A Parallel Approach for RegularSearch Algorithm

Jairo A. Lefebre-Lobaina(✉) and José Ruiz-Shulcloper

Universidad de Ciencias Informáticas, La Habana, Cuba
jairo.lefebre@gmail.com, jshulcloper@uci.cu

Abstract. The process of finding minimal subsets of features that can differentiate objects belonging to different classes, known as typical testors, is an exponential complexity problem. Several algorithms have been proposed to improve the search process efficiency by utilizing different properties and techniques, including parallelism. In this paper, we propose a parallel version of the RegularSearch algorithm to find all typical testors related to a supervised classification problem. The proposed algorithm is compared with the most recent algorithm in the literature, and the comparative analysis shows the advantages of our proposal over the compared methods, using both synthetic and real problem datasets.

Keywords: Feature selection · Typical testors · Reducts · Parallelization

1 Introduction

Supervised classification problems often involve structured data represented as features that describe a set of objects. However, as the number of features increases, it becomes more challenging to estimate the relevance of each feature and can impact the efficiency and accuracy of classification algorithms. Feature selection methods (*FS*) can help reduce the time required to train classification algorithms while maintaining accuracy [13]. One such approach to FS is the Test Theory, which was proposed by Zhuravlev [3].

Initially used for classification problems with two classes and objects described by Boolean features in the Pattern Recognition context, the concept of a *testor* was later extended to allow for its application in more complex situations [8]. Testors Theory has been successfully used to solve various problems, including text categorization [16], document clustering [11], fault diagnosis in steam turbines [5], and breast cancer diagnosis in medicine [4]. Moreover, it has been applied to estimate the relevance of each feature classification problems and for the *FS* process with mixed and incomplete data [20]. This is done by searching for *typical testors*, a concept related to *reducts* from Rough Sets Theory [9] and *minimal transversal* from Hyper Graph Theory [1].

The process of finding all *typical testors* (TT) has an exponential time complexity of 2^n, where n is the number of features describing the objects [8]. Also,

Y. Hernndez Heredia et al. (Eds.): IWAIPR 2023, LNCS 14335, pp. 72–83, 2024.
https://doi.org/10.1007/978-3-031-49552-6_7

several algorithms have been developed to improve the efficiency of this search process in different scenarios [19]. The algorithms for searching all *typical testors* can be divided into two main groups based on their search strategy: *external type* algorithms and *internal type* algorithms. The *external type* algorithms induce an order on the power set of features and use logical properties to minimize the number of comparisons carried out with most efficient approaches published in [6,12,15,17]. However, none of the recent approaches of mentioned algorithms have a proposed parallel version to find all typical testors.

On the other hand, the *internal type* algorithms focus their strategy on selecting comparisons between objects that fulfill certain properties and use these elements to build the *typical testor* set [2]. The updated version of YYC algorithm proposed by Piza et al. [14] is the latest and fastest approach for finding all typical testors in a parallel way.

In this paper, we propose a parallel approach for the internal type algorithm *RegularSearch* [10] to search for all typical testors related to a supervised classification problem. The rest of the paper is structured as follows: Sect. 2 presents the concepts used in this work, Sect. 3 describes the proposed algorithm in detail, Sect. 4 shows the results of the experimental study carried out, and finally, Sect. 5 presents our conclusions and proposals for future work.

2 Concepts Definition

Testor theory current research is based on three main concepts: the training matrix (TM), the difference matrix (DM), and the basic matrix (BM). The TM contains m objects, $\{O_1, O_2, ..., O_m\}$, each of which is described by n features, $R = \{x_1, x_2, \ldots, x_n\}$. We denote the value of feature x_i in object O_j as $x_i(O_j)$. The MD is a matrix that represents the difference or not (value 0) between each feature related to objects that belong to different classes.

Given a row $f_i \in MD$ and a feature $x_q \in R$, we denote the value of f_i in the feature x_q as $f_i[x_q]$. A row $f_i \in MD$ is considered a *sub-row* of $f_s \in MD$ if and only if for each $x_q \in R$, $f_i[x_q] \leq f_s[x_q]$ and there exists a feature x_p such that $f_i[x_q] < f_s[x_q]$. The matrix formed by all *sub-rows* in DM is called *basic matrix* (BM) and the rows in BM are a subset of the set of rows in DM [7].

Table 1. Transformations from TM to DM to BM.

TM		x_1	x_2	x_3	x_4
C_1	O_1	0	a	1	7
	O_2	1	b	0	4
	O_3	?	b	1	4
C_2	O_4	1	a	0	4
	O_5	1	b	1	7

DM	x_1	x_2	x_3	x_4
O_{14}	1	0	1	1
O_{15}	1	1	0	0
O_{24}	0	1	0	0
O_{25}	0	0	1	1
O_{34}	1	1	1	0
O_{35}	1	0	0	1

BM	x_1	x_2	x_3	x_4
O_{24}	0	1	0	0
O_{25}	0	0	1	1
O_{35}	1	0	0	1

In Table 1, we provide an example of an ME and the corresponding MD and MB. In this example, several data types are used in the presented features, including missing values (represented as ?), and all comparisons are made using a strict equality function with Boolean output. However, the testor theory has evolved to allow for more than two disjoint classes, different kinds of features, and comparison output values [8].

The density of value 1 is defined as $D(1) = U/(m_b * n)$, where m_b is the number of rows in BM and U is the number of 1 values contained in BM. $D(1)$ represents the minimum difference degree between objects that belong to different classes. If there are not many differences between objects that belong to different classes, the $D(1)$ value is low. For the BM presented in Table 1, we have $D(1) = 0.416$.

A subset of features T is a *testor*, if and only if when all features except those on T are removed from TM, the resulting sub matrix ME_t, contains no descriptions of equal objects that belong to different classes. This means that there are no zero rows in the sub matrix from the BM and DM formed by the features in T. T is a *typical testor* if and only if $\nexists T^{'}, T^{'} \subset T$ and $T^{'}$ is a testor [7]. For the TM presented in Table 1, and corresponding BM (and DM), the *typical testor* set is $T = \{\{x_1, x_2, x_3\}, \{x_2, x_4\}\}$.

2.1 Compatible and Regular Sets

Given two elements $a_{ij} = a_{pq} = 1$ from BM (or DM) in different rows and columns, the element a_{ij} is said to be *compatible* with a_{pq} if and only if $a_{pj} = a_{iq} = 0$. We will denote the compatibility relation between a_{ij} and a_{pq} as $a_{ij} \sim a_{pq}$. The set of elements $D = \{e_1, e_2, ..., e_r\}, D \in BM$ is a *compatible set* if and only if $r = 1$ or $e_t \sim e_p$ for $t \neq p$ and $t, p = 1, 2, ..., r$ [7]. Let MC be a sub-matrix of BM formed by the columns associated with the elements in D. For each feature $x_a \in MC$, there exists at least one row with value 0 in all features except in x_a.

From the previous definition, it can be deduced that any non-empty subset $Q \subset D$ is also a *compatible set*. Moreover, if Z is not a compatible set, any superset W that holds $Z \subset W$ will also not be a compatible set.

Let D be a *compatible set* and MC be the sub-matrix of BM formed by the columns associated with the elements in D. D is a *regular set* if and only if the corresponding features in MC satisfy the testor property, which is equivalent to saying that MC does not have any row with 0 values. Moreover, if D is a *regular set*, then the set of features $T \in MC$ is also a *typical testor* [7].

3 Parallel RegularSearch

In this work, is proposed a parallel version of the *RegularSearch* algorithm to identify all *typical testors* related to a supervised classification problem. The original *RegularSearch* algorithm was presented in [10] with the aim of finding all *regular sets* in the power sets of features from BM. The algorithm starts by

selecting the first feature in BM as an initial candidate and then incrementally adds new features during the search process.

In [10] is also proposed an arrangement process for BM to avoid comparisons with elements that will never be a testor. However, in order to provide a simpler description and better understanding of the parallel approach, we have simplified the sorting method.

The first step in the sorting process is to arrange the rows in BM in such a way that the first row in the matrix has the lowest number of elements with value 1. If there is more than one row that meets this criteria, any of them can be selected. The order of the remaining rows is not relevant. In the second step, the columns are sorted so that all elements with value 1 in the first row are grouped on the left side of BM. An example of this process is presented in Table 2, where BM^s is the sorting output.

Table 2. Sorting process example.

BM	x_1	x_2	x_3	x_4	x_5
O_1	0	0	1	1	1
O_2	1	0	0	0	1
O_3	0	1	0	1	0
O_4	1	1	1	0	0

BM^s	x_1	x_5	x_2	x_3	x_4
O_2	1	1	0	0	0
O_1	0	1	0	1	1
O_3	0	0	1	0	1
O_4	1	0	1	1	0

The search process begins with a *candidate set* of features denoted by C, where $|C| = 1$ and contains a single feature that, by definition, is a *compatible set*. Let MC be the submatrix from BM formed by the features in C. The algorithm continues to add new features to C incrementally, also checking that MC contains at least one subset of elements that holds the compatible set property and involves all the features in C. If MC contains a regular set (i.e., it has no zero rows), then C is added to the typical testor set (TT), and the search process for that candidate stops, because any superset will be a testor but will not hold the typicity property. If a feature $x_a \in BM$ is added to C, and the corresponding MC does not contain a *compatible set*, the search process stops because any superset from C will not be a *compatible set* either. In that situation, x_a is replaced with the feature x_{a+1} in C to continue the search process. If x_a is the last feature, the process stops.

All the steps mentioned above are described in the algorithm *GetRegularities* used by *RegularSearch* [10], and updated in Algorithm 1. This process is carried out in a recursive way. In every recursive call, the compatible set description contained in MC is generated. The behavior of the algorithm can be topologically represented as a depth-first tree search with a pre-order traversal strategy and a pruning process based on the compatible and regular sets properties. The main difference with [10] is that in the current proposal, *GetRegularitiesT*

initializes the sets TT, ZR, and OD to run independently in a parallel process call (*thread*), and returns a subset of the typical testor set.

To ensure proper data structuring between iterations, hashing-based dictionaries are utilized as a data structure to store tuples in the form of (x_i, r_i). A dictionary is denoted as $OD[x_i] = r_i$, where OD is the dictionary, x_i is a feature from BM, r_i is a set of row positions in BM, and $|OD|$ represents the number of stored tuples in OD. Additionally, the set of row positions in BM related to the feature x_i that only contains zeros is denoted as $z(x_i)$, while the remaining rows are denoted as $o(x_i)$.

Algorithm 1. *GetRegularitiesT*

input: BM, F, C, TT, ZR, OD
output: TT

```
1: define c_n as last element in C
2: if |C| = 1 then
3:     define TT = ∅
4:     ZR = z(c_n)
5:     OD[c_n] = o(c_n)
6: end if
7: for all f_j ∈ F, where c_n ∈ F and j > n do
8:     define Inf as ZR ∩ o(f_j)
9:     if |Inf| > 0 then
10:        define row_unitary as True
11:        define nOD as an empty dictionary
12:        for all x_i, r_i ∈ OD do
13:            nOD[x_i] = r_i − o(f_j)
14:            if |nOD[x_i]| = 0 then
15:                define row_unitary as False
16:                break loop
17:            end if
18:        end for
19:        if row_unitary then
20:            if |ZR| = 0 then
21:                add C to TT
22:            else
23:                nOD[f_j] = Inf
24:                define nC = C ∪ {f_j}
25:                define nZR = ZR − o(f_j)
26:                GetRegularitiesT(BM, F, nC, TT, nZR, nOD)
27:            end if
28:        end if
29:    end if
30: end for
31: return TT
```

Let TT be a typical testor set from BM, F the sorted set of features in BM, a sorted candidate set $C \neq \emptyset, C \subset F$, MC the submatrix from BM formed by the features in C, ZR the set of zero row positions in MC, and OD a dictionary to store the relation associated with feature $x_i \in C$ with the row position set from BM with zero value in all features except in x_i. Algorithm 1 describes the search for regular sets.

Let BM a basic matrix related to a supervised classification problem, and let TT the typical testor set in BM. The process to find all TT in a parallel way is described in Algorithm 2. It's important to highlight that, after the sorting process, the number of threads required will be the same as the number of non-zero values in the first row of BM.

Algorithm 2. *ParallelRegularSearch*

input: BM **output:** TT

1: **sort** BM according to 1 values contained in rows
2: **define** F **as** sorted set of features in BM
3: **define** $TList$ **as** a list of threads
4: **define** $TT = \emptyset$
5: **for all** $x_i \in F$ **do**
6: **define** r_1 **as** first row position in BM
7: **if** $r_1 \in z(x_i)$ **then**
8: **break loop**
9: **end if**
10: **define** $C = \{x_i\}$ **as** a sorted set of features
11: **if** $|z(x_i) = 0|$ **then**
12: **add** C to TT
13: **else**
14: **add** $GetRegularitiesT(BM, F, C, ZR, OD)$ as a thread call to $TList$
15: **end if**
16: **end for**
17: **for all** $tc_i \in TList$ **do**
18: **define** tt_i **as** the output of tc_i thread
19: $TT = TT \cup tt_i$
20: **end for**
21: **return** TT

The *ParallelRegularSearch* algorithm uses the *GetRegularitiesT* subroutine as the search strategy over the feature set in BM. Avoiding unnecessary comparisons is the core reason behind the efficiency of the *GetRegularitiesT* subroutine. Additionally, the tree-like search process ensures that every possible candidate is evaluated only once, which is the main characteristic that allows the parallelization of the searching process, increasing the efficiency without losing effectiveness. Since *GetRegularitiesT* does not compare $x_i \in F$ with previous features in F during the search process, *ParallelRegularSearch* provides each feature x_i as a new candidate for each thread. Each thread will return the typical testor subset, where x_i is the first feature, and the remaining features x_j hold $j > i$. Finally,

the set union of the different outputs will be the typical testor set associated with the input BM.

3.1 Execution Example

To show how the *ParallelRegularSearch* algorithm works, Table 3 describes each step executed using BM^s presented in Table 2 for a single thread call. Since the presented example is only for a single thread call the output will contain all typical testors that contains x_1.

Table 3. Execution steps description.

Step	C	Description
1	$\{x_1\}$	Take x_1 as candidate, due is not a typical testor and (by definition) is a compatible set
2	$\{x_1, x_5\}$	Add x_5 to C, due MC contains a compatible set, but C is not a testor
3	$\{x_1, x_5\}$	Ignore x_2, due MC corresponding to $\{x_2\} \cup C$ not contains a compatible set that include all features
4	$\{x_1, x_5\}$	Ignore x_3, due MC corresponding to $\{x_3\} \cup C$ not contains a compatible set that include all features
5	$\{x_1, x_5\}$	Ignore x_4, due MC corresponding to $\{x_4\} \cup C$ not contains a compatible set that include all features. Since there is no other combination the search process for $\{x_1, x_5\}$ stop (β)
6	$\{x_1, x_2\}$	Add x_2 to C, due MC contains a compatible set, but C is not a testor
7	$\boldsymbol{\{x_1, x_2, x_3\}}$	Add x_2 to C, due MC contains a compatible set and also a regular set, the current C is a typical testor and is added to TT set; since any super set of features form C will be not a typical testor (β)
8	$\{x_1, x_2\}$	Ignore x_4, due MC corresponding to $\{x_4\} \cup C$ not contains a compatible set that include all features. Since there is no other combination the search process for $\{x_1, x_2\}$ stop (β)
9	$\{x_1, x_3\}$	Add x_3 to C, due MC contains a compatible set, but C is not a testor
10	$\{x_1, x_3\}$	Ignore x_4, due MC corresponding to $\{x_4\} \cup C$ not contains a compatible set that include all features. Since there is no other combination the search process for $\{x_1, x_3\}$ stop (β)
11	$\boldsymbol{\{x_1, x_4\}}$	Add x_4 to C, due MC contains a compatible set and also a regular set, the current C is a typical testor and is added to TT set; since any super set of features form C will be not a typical testor (β)
12	–	Return the typical testors found since there is no possible feature to compare with after x_4. $TT = \{\{x_1, x_2, x_3\}, \{x_1, x_4\}\}$

In each step of the algorithm, a different candidate is tested, and the algorithm will not proceed to the next step if MC does not contain a compatible set with at least one element in each column. This is why the values in C in Table 3 can change without an explicit insertion or elimination; the algorithm backtracks (represented as β) to a previous state. When C satisfies the typical testor definition, the values are highlighted in bold.

In the sorted matrix, if the search process takes the feature x_2 as a new candidate in C, to find the possible typical testors with the remaining features (x_3 and x_4), any combination will not generate a testor because it will contain a row with zero values in MC. For this reason, only two threads are required to process that dataset.

The remaining steps using BM^s are carried out in parallel. Theoretically, if an infinite number of threads were available, the searching time would be the time required for the first thread to finish, because the search space is reduced by half in each following thread. Also, the amount of threads required is the same as the amount of ones in the first row of BM^s. The output of the second thread for searching the typical testors associated with x_5 in BM^s is $\{x_5, x_2\}, \{x_5, x_3, x_4\}\}$. The final TT set is the set union of each thread result:

$TT = \{\{x_1, x_2, x_3\}, \{x_1, x_4\}, \{x_5, x_2\}, \{x_5, x_3, x_4\}\}$.

4 Experimental Results

To demonstrate the advantages of the parallel approach for the *RegularSearch* algorithm (P-RS), several experiments were conducted using synthetic and real-world problem datasets. For the comparison process, the proposal was compared with the parallel version of the YYC algorithm (P-YYC).

Recent research has shown that modern algorithms exhibit variations in efficiency based on the $D(1)$ value [18]. To focus on the efficiency of the compared algorithms, the primary comparison criteria selected were the $D(1)$ value and the total time required to find all typical testors.

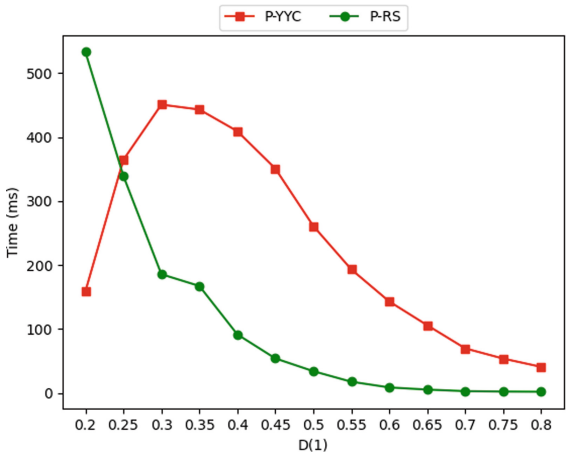

Fig. 1. Results for BMs with 250 rows and 20 columns.

To conduct experiments with synthetic matrices, several BMs of fixed dimensions were randomly generated. To match the expected $D(1)$ value, the synthetic

matrices were generated with a similar distribution of ones in each row. For each $D(1)$ value 50 matrices were generated, and then calculated the mean execution time, for a total of 650 BMs. All experiments were conducted on an Asus ROG Zephyrus G14 computer equipped with an AMD Ryzen^{TM} 7 4800HS processor and 16 GB of RAM.

As shown in Fig. 1, the *P-RS* algorithm has better results for higher $D(1)$ values, while *P-YYC* performs better for lower values. However, to show how the obtained values may change when using more complex datasets, two more test sets were conducted following the same comparison criteria.

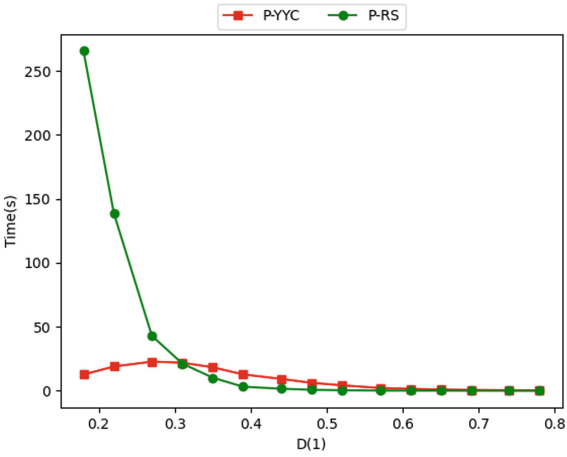

Fig. 2. Results for BMs with 250 rows and 30 columns.

For the next test with synthetic datasets, it was decided to increase the number of features while keeping the number of rows fixed. Is a know fact that the search space will increase and the complexity of the problem as the number of features increases. However, the Fig. 2 also shows that the relationship between the number of rows and columns may also affect the performance of the algorithms. Although *P-RS* still outperforms *P-YYC* for high $D(1)$ values with up to 9 s of difference, the results for lower densities are considerably worse. On the other hand, in the Fig. 3, it can be observed that *P-YYC* always performs worse than *P-RS*, with differences up to 13 s, when the number of rows is increased while keeping the number of columns fixed.

The second test set was carried out using well-known datasets from the UCI Machine Learning Repository[1] to evaluate the behavior of the compared algorithms on real-life problems. The only dataset that is not from UCI is the *Higher Education Students Performance Evaluation* (K-higher edu) dataset, which was

[1] UCI Machine Learning Repository https://archive.ics.uci.edu/.

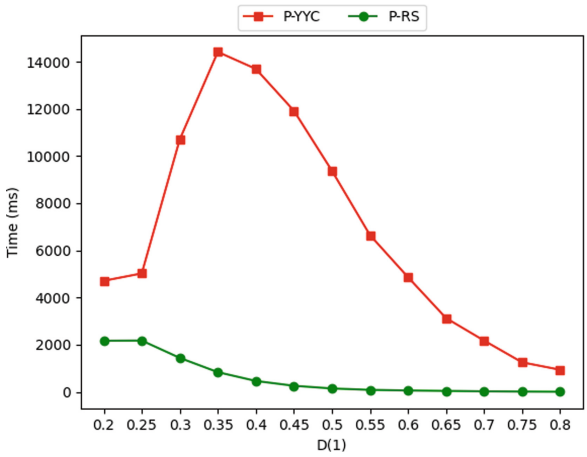

Fig. 3. Results for BMs with 1000 rows and 20 columns.

taken from Kaggle[2]. Since the previous experiments were conducted according to the $D(1)$ value, several datasets with different $D(1)$ values associated with BM were selected. The algorithms were implemented in Java using the *Runnable* interface and published on GitHub.[3].

Table 4. Dataset experimets.

Dateset	Rows	Cols	*D(1)*	P-YYC	P-RS
kr.vs.kp	29	36	0.02	**<1**	2
anneal	93	38	0.13	**30**	177
soybean	30	35	0.2	13	**6**
dermatology	1124	34	0.34	311791	**51969**
K-higher edu	2367	32	0.46	4272242	**172153**
cardiotocography	9751	35	0.54	23455339	**125839**
HCV-Egyptian	1442	28	0.67	15412	**159**
promoter	2761	57	0.75	26751971	**28994**
waveform5000	908	40	0.92	6	**<1**
DARWIN	1681	451	0.98	650407	**198**

The table presented in Table 4 provides a description of the BM associated with each dataset, sorted in ascending order according to the $D(1)$ value. Additionally, the table includes the execution time required for the algorithms *P-YYC*

[2] Kaggle dataset https://www.kaggle.com/datasets/mariazhokhova/higher-education-students-performance-evaluation.
[3] GitHub shared repository https://github.com/J41R0/RegularSearch.

and *P-RS*, presented in milliseconds. The results obtained are consistent with the experimentation using synthetic datasets, with differences of up to 7 h showing the superior performance of *P-RS* in BMs with high $D(1)$ value. Also, in all experiments can be appreciated that *P-YYC* for $D(1) > 0.5$ tends to improve the performance, but in any experiment get better results that *P-RS*.

The experimentation results clearly demonstrate that the algorithms exhibit different behavior depending on the proportion of rows and columns in the input BM. Therefore, further study is necessary for the precise selection of a typical testor finding algorithm according to the characteristics of the BM.

5 Conclusions

This paper presents a parallel approach to the internal type algorithm *RegularSearch* for finding all typical testors related to a supervised classification problem. The proposed algorithm parallelizes the search process of all typical testors associated with a feature with others on the left of the basic matrix, and the complete typical testor set is obtained by taking the union of all the subsets found.

The experiments conducted show that the proposed algorithm is more efficient than the parallel version of the YYC algorithm when $D(1)$ values are high. The results obtained with synthetic and real-life datasets confirm the behavior of the algorithm and its ability to handle real problems that generate a BM with high $D(1)$ value.

For future work, we suggest developing a methodology for comparing typical testor finding algorithms that considers the input matrix dimensions and $D(1)$ value. It would be interesting to investigate the relationship between the proposed approach and parallelization paradigms for BigData and HPC. Additionally, we propose developing a version of the algorithm that can take advantage of the computing capabilities of GPUs.

References

1. Alba-Cabrera, E., Godoy-Calderon, S., Lazo-Cortés, M.S., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A.: On the relation between the concepts of irreducible testor and minimal transversal. IEEE Access **7**, 82809–82816 (2019)
2. Alba-Cabrera, E., Ibarra-Fiallo, J., Godoy-Calderon, S., Cervantes-Alonso, F.: YYC: a fast performance incremental algorithm for finding typical testors. In: Bayro-Corrochano, E., Hancock, E. (eds.) CIARP 2014. LNCS, vol. 8827, pp. 416–423. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12568-8_51
3. Dmitriev, A., Zhuravlev, Y.I., Krendeliev, F.: About mathematical principles of objects and phenomena classification. Diskretni Analiz **7**, 3–15 (1966)
4. Gallegos, A., Torres, D., Álvarez, F., Soto, A.T.: Feature subset selection and typical testors applied to breast cancer cells. Res. Comput. Sci. **121**(1), 151–163 (2016)

5. Gómez, J.P., Hernández Montero, F.E., Gómez Mancilla, J.C.: Variable selection for journal bearing faults diagnostic through logical combinatorial pattern recognition. In: Hernández Heredia, Y., Milián Núñez, V., Ruiz Shulcloper, J. (eds.) IWAIPR 2018. LNCS, vol. 11047, pp. 299–306. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01132-1_34
6. Gómez, J.P., Montero, F.E.H., Sotelo, J.C., Mancilla, J.C.G., Rey, Y.V.: RoPM: an algorithm for computing typical testors based on recursive reductions of the basic matrix. IEEE Access **9**, 128220–128232 (2021)
7. 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 Recogn. Lett. **16**(12), 1259–1265 (1995)
8. Lazo-Cortes, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An overview of the evolution of the concept of testor. Pattern Recogn. **34**(4), 753–762 (2001)
9. Lazo-Cortés, M.S., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Sanchez-Diaz, G.: Are reducts and typical testors the same? In: Bayro-Corrochano, E., Hancock, E. (eds.) CIARP 2014. LNCS, vol. 8827, pp. 294–301. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12568-8_36
10. Lefebre-Lobaina, J.A., Ruiz-Shulcloper, J.: Regularsearch, a fast performance algorithm for typical testors computation (ND) (Under review)
11. Li, F., Zhu, Q.: Document clustering in research literature based on NMF and testor theory. JSW **6**(1), 78–82 (2011)
12. Lias-Rodriguez, A., Sanchez-Diaz, G.: An algorithm for computing typical testors based on elimination of gaps and reduction of columns. Int. J. Pattern Recognit. Artif. Intell. **27**(08), 1350022 (2013)
13. Mafarja, M., Qasem, A., Heidari, A.A., Aljarah, I., Faris, H., Mirjalili, S.: Efficient hybrid nature-inspired binary optimizers for feature selection. Cogn. Comput. 1–26 (2019)
14. Piza-Davila, I., Sanchez-Diaz, G., Lazo-Cortes, M.S., Noyola-Medrano, C.: Enhancing the performance of YYC algorithm useful to generate irreducible testors. Int. J. Pattern Recognit. Artif. Intell. **32**(01), 1860001 (2018)
15. Piza-Dávila, I., Sánchez-Díaz, G., Lazo-Cortés, M.S., Villalón-Turrubiates, I.: An algorithm for computing minimum-length irreducible testors. IEEE Access **8**, 56312–56320 (2020)
16. Pons-Porrata, A., Gil-García, R., 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). https://doi.org/10.1007/978-3-540-76725-1_67
17. Rodríguez-Diez, V., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Lazo-Cortés, M.S.: A new algorithm for reduct computation based on gap elimination and attribute contribution. Inf. Sci. **435**, 111–123 (2018)
18. Rodríguez-Diez, V., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Lazo-Cortés, M.S.: The impact of basic matrix dimension on the performance of algorithms for computing typical testors. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-López, J.A., Sarkar, S. (eds.) MCPR 2018. LNCS, vol. 10880, pp. 41–50. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92198-3_5
19. Sánchez-Díaz, G., Lazo-Cortés, M.S., Aguirre-Salado, C.A., Piza-Davila, I., Garcia-Contreras, J.P.: A review of algorithms to computing irreducible testors applied to feature selection. Artif. Intell. Rev. 1–22 (2022)
20. Solorio-Fernández, S., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F.: A survey on feature selection methods for mixed data. Artif. Intell. Rev. **55**(4), 2821–2846 (2022)