UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Parallelization Algorithm for the calculation of Typical Testors based on YYC

Performance evaluation in synthetic matrices and application to biodegradable molecules

Ariana Elizabeth Soria Salgado

Matemáticas

Trabajo de fin de carrera presentado como requisito para la obtención del título de Matemática

Quito, Septiembre de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN DE TRABAJO DE TITULACIÓN

Parallelization Algorithm for the calculation of Typical Testors based on YYC

Performance evaluation in synthetic matrices and application to biodegradable molecules

Ariana Elizabeth Soria Salgado

Nombre del profesor, Titulo académico: Julio Ibarra, M.Sc

Quito, Septiembre de 2022

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas. Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Nombres y apellidos:	Ariana Elizabeth Soria Salgado
Código:	00201791
Cédula de Identidad:	1718927203
Lugar y fecha:	Quito, Septiembre de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en `http://bit.ly/COPETheses`.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on `http://bit.ly/COPETheses`.

Dedicado a mi familia

Resumen

Dentro del área de reconocimiento de patrones, una de las principales preocupaciones es la de conseguir algoritmos que sean eficientes en términos de tiempo y de reconocimiento. Motivados por este objetivo, se han desarrollado teorías matemáticas que sirven como base fundamental para desarrollar algoritmos que reducen la cantidad de características necesarias para realizar buenas discriminaciones por clases. Una de estas teorías se llama: Teoría de Testores.

En el presente trabajo se propone un método innovador para hallar testores típicos, los cuales permiten la reducción de características para realizar procesos de clasificación o reconocimiento. El método está respaldado por fundamentos matemáticos que vienen de Teoría de Testores. El algoritmo propuesto consiste en dividir una matriz básica en bloques, luego se hallan los testores típicos de los bloques definidos y para encontrar los testores típicos de la matriz básica completa se prueban uniones entre los elementos de estos conjuntos obtenidos de los bloques. Se desarrolla un criterio para determinar cuándo las uniones de testores típicos de Bloques forman testores típicos de su matriz básica completa.

Para hallar los testores típicos de los bloques se optó por usar el algoritmo YYC. El desempeño del método es evaluado mediante el uso de matrices sintéticas. Se compararan los tiempos de ejecución del método tanto en su versión paralelizada como secuencial y se contraste con el algoritmo YYC usado para la matriz básica completa.

Se detallan las características de las matrices en las cuales el método propuesto resulta ser más eficiente. Se discuten las razones por las cuales el algoritmo puede llegar a ser deficiente y cómo podría ser implementado con algoritmos diferentes al YYC para obtener los testores típicos de los bloques en el proceso de paralelización. Finalmente, se analiza su desempeño en una base de datos obtenida del Repositorio UCI [23].

Palabras clave: reconocimiento de patrones, clasificación, testores, clasificación, testores típicos, redes neuronales, precisión, eficiencia computacional, tiempo de ejecución.

Abstract

Within the area of pattern recognition, one of the main concerns is to achieve algorithms that are efficient in terms of time and recognition. Motivated by this goal, mathematical theories have been developed that serve as a fundamental basis for developing algorithms that reduce the number of features needed to make good class discriminations. One of these theories is called: Theory of Testors.

In the present work, an innovative method is proposed to find typical testors, which helps reduce the number of features needed to carry out classification or recognition processes. The method is supported by mathematical foundations that come from Testor Theory. The proposed algorithm consists of dividing a basic matrix into blocks, then the typical testors of the defined blocks are found. To find the typical testors of the complete basic matrix, unions between the elements of these sets obtained from the blocks are tested. A criterion is developed to determine when the unions of typical testors of Blocks form typical testors of their complete basic matrix.

To find the typical testors of the blocks, it was decided to use the YYC algorithm. The performance of the method is evaluated through the use of synthetic basic matrices. The execution times of the method in both its parallelized and sequential versions are compared and contrasted with the YYC algorithm applied to the complete basic matrix.

The characteristics of the matrices in which the proposed method turns out to be more efficient are detailed. In addition, the reasons why the algorithm can become deficient and how it could be implemented with algorithms other than YYC to obtain the typical testors of the blocks in the parallelization process are discussed. Finally, its performance is analyzed in a database obtained from the UCI Repository [23].

Keywords: pattern recognition, classification, testors, typical testors, irreducible testors, neural networks, accuracy, computational efficiency, time execution.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Since childhood, humans are able of preforming complex cognitive tasks. Our brain allows us to memorize, learn, recognize and identify objects, people, animals, and others. The fast pace at which technology has evolved has allowed scientists and researchers to successfully implement these complex cognitive capabilities in computational algorithms.

Neural networks are a clear example of this. They can solve several types of classification problems such as hand-writing, face, and object recognition, among others.

Although the results obtained have great accuracy, computational efficiency may be considered a significant issue [1]. Briefly, a testor is a feature subset which allows complete differentiation of objects from different classes. A typical testor, in short words, is minimum set of characteristics that allow us to classify elements of different classes [5].

In general, logic-combinatry approachs in pattern recognition have become an auxiliar criterion for medical diagnosis. Testor theory can be applied in various fields of innovation. One of the most important is the field of health. There are works in which typical testors have been used successfully for the diagnosis or classification of diseases.

For example, typical testors have been used in order to find a minimum set of characteristics that best describes benign or malignant breast cancer cells [33]. Testor theory has helped to obtain a minimum combination of symptoms and a combination of equally discriminating characteristics (typical testors) that allow diagnoses of diseases [34].

In addition to using testor theory to identify reduced sets of significant features to discriminate objects of different classes, computationally we can speak of parallel algorithms.

In general, classical numerical methods have not explored the alternative of using multiple processors and other hardware alternatives to improve the efficiency of their algorithms [35].

Familiar algorithms can be reformed to create efficient parallel algorithms. For example, within linear algebra there are algorithms that turn out to be more efficient when parallelized [35]. For example, matrix multiplication can be transformed into a parallel algorithm. In particular, a three-dimensional case of matrix multiplication an algorithm for massively parallel processing system has been presented [36].

Chapter 2

MATERIALS AND METHODS

Real life problems often involve dealing with high-dimensional data-sets, but not every variable in the data-set have the same level of significance in the understanding of the problem of interest. Reducing the dimension of the original data can be transformed into a computational advantage in terms of efficiency. This can be done by using Testor Theory [1].

In mathematical terms, we start with a p-dimensional variable that contains all the information of an object $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_p)^T$. We want to find a lower dimensional representation, $\mathbf{s}$, $\mathbf{s} = (s_1, s_2, s_3, \ldots, s_k)^T$, $k < p$, of $\mathbf{x}$ (lower number of characteristics) that can still properly define the object we are dealing with.

Non-Statistical methods for dimensional reduction representations were proposed in the former Soviet Union and later in Cuba [1]. These proposed methods gave birth to what we now know as "Testor Theory". The concepts of Non-reducible Testor or typical testor were introduced first introduced by Yablonskii and Cheguis [2], [3]. Years later Dimitriev et. al. [4] applied this theory to classification problems.

In this chapter, the theoretical background is introduced in order to understand every step of the proposed algorithm. The proposed method and the experimental setup are also explained in detail.

2.1 Testor Theory

Let U be the universe of objects of our study, where each object has p characteristics that define it.

For each object of our universe, we have a set P=$\{x_1, x_2, x_3 \ldots, x_p\}$ of attributes, which is a p-dimensional variable. The objects of our universe can be classified into N disjoint classes, with $N > 1$.

Definition 2.1.1 By comparing feature by feature each pair of objects belonging to different classes we can construct a Differentiation Matrix M_D.

$$M_D = [m_{ij}]_{N \times p}, \qquad m_{ij} \in \{0, 1\}$$

If $m_{ij} = 0$, the pair of objects i are similar in the jth attribute. If m_{ij}=1, the pair of objects i are different in the jth attribute.

Here is a small example to show how to get a differentiation matrix given a dataset with two classes (named A and B). Class A has 3 members and class B has two:

Example 1

Class	Object	X_1	X_2	X_3	X_4
A	O_1	4	23	true	0.5
A	O_2	5	15	false	0.2
A	O_3	10	41	false	0.1
B	O_4	10	57	false	0.5
B	O_5	3	41	true	0.2

Let's compare each pair of objects of different classes, feature by feature. If the two objects for a feature j take the same value, in the differentiation matrix we put zero in this entry, otherwise we put one. We carry out this process until all possible pairs of objects of different classes have been compared.

The differentiation matrix for this example is shown below:

Comparison	X_1	X_2	X_3	X_4
O_1O_4	1	1	1	0
O_1O_5	1	1	0	1
O_2O_4	1	1	0	1
O_2O_5	1	1	1	0
O_3O_4	0	1	0	1
O_3O_5	1	0	1	1

Definition 2.1.2 Let f and g be any two rows of M_D. We say that f is *less than* g if $\forall i\ f_i \leq g_i$ and $\exists j$ such that $f_j \neq g_j$.

Definition 2.1.3 The row f is a *basic row* of M_D if there is no row g such that g is less than f in M_D.

Definition 2.1.4 The matrix M that only contains the basic rows of M_D is called the basic matrix of M_D.

Going back to Example 1, we will now obtain the basic matrix. If we look closely at the differentiation matrix of this example, we notice that row O_1O_5 and row O_2O_4 are the same. Therefore, they are comparable (as in Definition 2.1.2. Thus, one must be eliminated (no matter which one). This also happens for rows O_1O_4 and O_2O_5. On the other hand, row O_3O_4 is less than row O_1O_5 according to Definition 2.1.2. Finally, we can check that rows O_1O_4, O_3O_4 and O_3O_5 meet the characteristic of being basic rows according to Definition 2.1.3. Hence, the matrix that is only formed by these rows is a basic matrix that follows the Definition 2.1.4:

Comparison	X_1	X_2	X_3	X_4
O_1O_4	1	1	1	0
O_3O_4	0	1	0	1
O_3O_5	1	0	1	1

Definition 2.1.5 A feature subset $T \subseteq P$, $T = \{j_{k_1}, j_{k_2}, j_{k_3} \ldots, j_{k_s}\}$ is a testor if and only if when all features are eliminated, except those in T, there is not any pair of similar subdescriptions in different classes.

Therefore, a testor is a feature subset, which allows complete differentiation of objects from different classes [5].

If $T \subseteq P$, we define $M_{|T}$ as the matrix obtained from M by eliminating all the columns of M that do not belong to the set T.

In terms of M, we say that T is a testor if $M_{|T}$ does not have any rows of zeros.

Definition 2.1.6 The attribute $j_{k_r} \in T$ is typical with respect to T and M if $\exists q$, $q \in \{1, 2, 3, \ldots, s\}$ such that $f_{i_q j_{k_r}} = 1$ and $\forall l \neq q$, $f_{i_q j_{k_l}} = 0$.

Thus a set T has the typical property with respect to the matrix M if all its elements are typical attributes with respect to T and M.

Proposition 2.1.1 A set $T = \{j_{k_1}, j_{k_2}, j_{k_3} \ldots, j_{k_s}\} \subseteq P$ has the typical property with respect to the matrix M if and only if we can obtain an identity matrix in $M_{|T}$ by exchanging and eliminating some rows [6].

Going back to Example 1, we are going to show how Proposition looks in practice.

Within the set of all testors, there are some testors, which are irreducible. These kinds of testors are called typical testors [5]. The set of characteristics $T = \{X_1, X_4\}$ has the typical property and therefore we can visualize an identity matrix by taking only these columns of the basic matrix:

Comparison	X_1	X_4
O_1O_4	1	0
O_3O_4	0	1
O_3O_5	1	1

As can be seen, the identity matrix 2×2 is formed with rows O_1O_4 and O_3O_4 when we restrict the basic matrix to the set T.

Definition 2.1.7 A feature subset $T \subseteq P$ is a typical testor or irreducible testor if and only if T is a testor and there is no other testor τ such that $\tau \subset T$. This means that every feature in T is essential, so if we eliminated any, the resulting set is no longer a testor.

Once again, we return to Example 1. It has already been shown that the set $T = \{X_1, X_4\}$ has the typical property, now we will see that it is a typical testor. If we delete the column X_1, we are left with:

Comparison	X_1
O_1O_4	1
O_3O_4	0
O_3O_5	1

This can no longer be a testor since it has a row of zeros.Now if we remove the feature X_4, we get:

Comparison	X_4
O_1O_4	0
O_3O_4	1
O_3O_5	1

Similarly, a row of zeros is obtained. According to the Definition 2.1.7 the set T is a typical or irreducible testor of the basic matrix.

In terms of the basic matrix M, T is a typical testor of M if it is a testor and has the typical property with respect to M.

Proposition 2.1.2 Let $\Psi(M_D)$ be the set of all typical testor of M_D and $\Psi(M)$ be the set of all typical testor of M, with M the basic matrix of M_D then

$$\Psi(M_D) = \Psi(M).$$

By Proposition 2.1.2, we conclude that in terms of computational efficiency, it is better to work with M than with M_D because M has less or equal number of rows than M_D, which can make algorithms work faster to obtain the typical testors [6].

In the next section, the connections between Testor Theory and Hypergraph Theory, a highly studied area in the field of Mathematics, will be explored.

2.2 Connection between Testor Theory and Hypergraph Theory

Within the field of discrete mathematics, one of the most important topics is graph theory. Graph theory studies mathematical structures used to model pairwise relations

between objects through mathematical objects known as graphs. Moreover, within graph theory, graphs can be generalized. Hypergraphs are generalizations of graphs in which an edge can join any number of vertices. This allows more than only pairwise relations.

Berge was the first person to introduce Hypergraphs in 1973 [7], [8], [9]. Graphs only admit pairwise relationships, that is an edge can only join one vertex. On the other hand, Hypergraphs support multi-adic relationships so they become a more natural model of real life problems [7].

Definition 2.2.1 An hypergraph is an ordered pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a finite set of objects, and $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m\}$ is a family of subsets of $\mathcal{V}$.

Definition 2.2.2 The set $\mathcal{V}$ is the set of *vertices* or *nodes* and the elements of $\mathcal{E}$ are called *hyperedges*. So in a hypergraph, a hyperedge can links one or more vertices.

Definition 2.2.3 The incident matrix A of $\mathcal{H}$ is and $n \times m$ matrix whose rows correspond to the number of vertices of the hypergraphs and the columns correspond to the number of hyperedges of $\mathcal{H}$:

$$A = [a_{ij}]_{n \times m}, \qquad a_{ij} \in \{0, 1\}$$

In such a way that $a_{ij} = 1$ if $v_i \in \mathcal{E}_j$ and $a_{ij} = 0$ otherwise.

Definition 2.2.4 A hypergraph $\mathcal{H}$ is called simple if for every pair $(\mathcal{E}_i, \mathcal{E}_j)$, if $\mathcal{E}_j \subseteq \mathcal{E}_i$, then $j = i$.

Definition 2.2.5 Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A set $\tau \subseteq \mathcal{V}$ is called a transversal of $\mathcal{H}$ if it intersects all its hyperedges. In other words, $(\forall \mathcal{E}_i \in \mathcal{E})\ \tau \cap \mathcal{E}_i \neq \emptyset$.

A transversal τ is called minimal if no proper subset τ' of τ is a transversal of $\mathcal{H}$.

Definition 2.2.6 The transversal hypergraph $Tr(\mathcal{H})$ of a hypergraph $\mathcal{H}$ if the family of all minimal transversals of $\mathcal{H}$.

Until now, we presented definitions of two, apparently, different theories. But, there exists a relation between Testor Theory and Hypergraph Theory [10]. Based on the definitions of basic matrix, incidence matrix and simple hypergraph, the following theorem can be stated.

Theorem 2.2.1 A transposition over the incidence matrix of a simple hypergraph, results in a matrix that fulfills all required properties to be a basic matrix.

Theorem 2.2.1 allows us to obtain the typical testors of the basic matrix obtained by transposing the incidence matrix. The complete set of irreducible testors turns out to be the transversal hypergraph of the original hypergraph [10]. Futhermore, there is a relationship between Typical Testor and Minimal Transversal.

Theorem 2.2.2 Let $\Psi(B^*)=\{\tau_1, \tau_2, \ldots, \tau_s\}$ be the complete family of irreducible testors from a basic matrix B. Let $Tr(\mathcal{H})$ be the transversal hypergraph for a simple hypergraph $\mathcal{H}$ whose incidence matrix is exactly the transposed matrix of B, B^T, then $\Psi^*(B)=Tr(\mathcal{H})$.

Theorem 2.2.2 states the equivalence between computing typical testors from the transposed incidence matrix of a simple hypergraph $\mathcal{H}$ and minimal transversals of $\mathcal{H}$ [10]. The most important fact of this theorem is that the computation of irreducible testors and minimal transversal can be performed by using any known algorithm designed for one task or the other.

In Graph Theory, the Transversal Hypergraph Generation Problem is the problem of generating the set $Tr(\mathcal{H})$ of a given hypergraph $\mathcal{H}$. There exist several algorithms to fulfill this task such as the Kavvadias-Stavropoulos Algorithm (KS) [12], Berge's Algorithm [13], Khachiyan's Algorithm (improvement of Berge's Algorithm) [14], Simbolic Learning to improve the performance of transversal algorithms [27] and others.

In Testor Theory, algorithms have also been developed to find the set of irreducible testors of a given basic matrix B. The Binary-Recursive Algorithm (BR) [15], BT Algorithm [16], LEX Algorithm [18], YYC [17], among others.

It should be noted that although it is true that there is a relationship between typical testors and minimum transversal, in this paper we will use the mathematical language of Testor Theory. The objective of this section is to note that this relationship must be taken into account since it allows transferring the results of typical testors to this area, which is also highly studied today.

Next, the characteristics of the algorithm to find typical testors that have motivated this work will be detailed: Algorithm YYC. Key concepts on which their programming is based are introduced and resumed. In addition, we detail how it will be used in the proposed approach to find irreducible testors.

2.3 YYC Algorithm

The YYC Algorithm was introduced in 2014 by Eduardo Alba, Salvador Godoy, Julio Ibarra and Fernando Cervantes as a new computacional technique to find the complete set of typical testors of any basic matrix [17]. The algorithm is based on the strategy of breaking down the calculation of typical testors up to some i row, instead of calculating the typical testors of the whole matrix at once. Once the algorithm generates irreducible testors up to some row i, with respect to the submatrix constituted up to the first i rows,

it uses this set to build a set of irreducible testors up to row $i+1$ and so on.

One of the most novel contributions of the YYC Algorithm is that it introduces the concept of compatible sets [17], [18] into its programming and this algorithm can be parallelized to improve computational efficiency [29]. Moreover, the characteristic of the YYC algorithm to build the typical testors row by row motivated the idea of parallelization for the proposed algorithm, since this calculation of typical testors can be done separately in a partition by rows of the basic matrix.

2.3.1 Compatible Sets

Definition 13. Let B be a basic matrix and E a set of elements. E is said to form a compatible set if under some row and column rearrangement, those elements shape into an identity matrix [17].

If some elements of a basic matrix B form a compatible set, then the corresponding subset form a typical testor of B if and only if the submatrix corresponding to those columns has no rows of zeros [17]. This statement guarantees the typicity and testor conditions.

In the work of Eduardo Alba, Salvador Godoy, Julio Ibarra and Fernando Cervantes [17], a function to find compatible sets is proposed. Based on their work, we propose the following algorithm 1:

Algorithm 1 Find Compatible Set

Input: T_j indices of testor columns, x_m index of column of basic matrix B and f the number of row of B.

Output: `True` if the set $\{T_j, x_m\}$ form a typical testor (compatible set) and `False` if they do not.

1: Define $U = \{T_j, x_m\}$ ▷ We want to check if this new set is a typical testor or not.
2: Define SM as a submatrix of B up to the f-th row with the columns define by U.
3: Redefine SM including only unique basic rows. ▷ This will simplify the number of iterations of the algorithm to check if there exists a compatible set or not.
4: Define n_b as the number of basic rows.
5: **if** $n_b \geq length(U)$ **then**
6: **for** `<i=1 up to the last column of` SM`>` **do**
7: **if** the sum of the all the columns in SM in the row i is equal to zero **then**
8: **return** `False`
9: **return** `True`
10: **else**
return `False`

2.3.2 YYC Algorithm-Code

In this subsection we will present our own version of the YYC Algorithm 2.3.2 implemented in *Matlab 2022a*. The code is based on the work of the authors Eduardo Alba, Salvador Godoy, Julio Ibarra and Fernando Cervantes [17].

Algorithm 2 YYC Algorithm

Input: B basic matrix.
Output: Ψ complete set of typical testors of B.

Read only the first row of basic matrix B. Define the vector *ones*.
for <j=1 up to the last column of B> **do**
    if the first row of B has a 1 in that column **then**
        Add i to the vector *ones*. ▷ For the first row, each column with a value 1 is a typical testor for that row.
Let $\Psi = ones$.
for <i=2 up to the last row of B> **do**
    Define $\Psi^* = \emptyset$.
    for <each $\tau_j \in \Psi^*$> **do**
        if $\exists x_p \in \tau_j \; [r_i[x_p] = 1]$ **then**
            Add τ_j to Ψ^*
        else
            Define the vector *onescol*.
            Using the function *find*, search for all x_p in r_i such that $r_i[x_p] = 1$ and store it in *onescol*. ▷ *find* is a function of Matlab.
            for <each x_p in *onescol*> **do**
                if *FindCompatibleSet*(τ_j, x_p) is `True` **then**
                    Add $\tau_j \cap \{x_p\}$ to Ψ^*
    Let $\Psi = \Psi^*$
    return Ψ

The YYC Algorithm has been reported to become slower as the number of compatible sets or the number of rows grow. This has been one of the main motivations for this work.

In the next section work we introduced a new approach for finding typical testors. An innovative way of applying YYC will be presented as part of a new algorithm to find typical testors of a basic matrix.

2.4 Blocks Algorithm Description

During this work we present a new approach to obtain typical testors of a basic matrix B. The strategy is inspired on the phrase "divide and conquer". Based on typical testors of blocks of B we will find typical testors of all of B. In addition to seeking to propose a more efficient algorithm to obtain irreducible testors of a given basic matrix, we want to propose one that can be parallelized.

Definition 2.4.1 A *block* B of M is a submatrix of M consisting of all columns but only some rows.

Proposition 2.4.1 Let B be a basic matrix. Define B_1 as a block of B. Then B_1 is a smaller basic matrix.

Proof: Because B is a basic matrix, it is composed only by basic (incomparables) rows. Thus if we choose just some rows of B but all of its columns, we end up with a smaller matrix that fulfills all the properties to be a basic matrix since its formed only by basic rows. □

2.4.1 Two Blocks of a Basic Matrix

In this section, we explore relationships between testors of two different blocks of a basic matrix. Although we will only work with two blocks, in the case we have more the ideas can be generalized.

Proposition 2.4.2 Let B be a basic matrix, T_1 and T_2 be testors of B then $T_1 \cup T_2$ is also a testor of B.

Proof: Define the set $T = T_1 \cup T_2$. By Definition 5, $B_{|T_1}$ and $B_{|T_2}$ do not have any rows of zeros. Therefore when we considered the matrix $B_{|T}$ it will be a submatrix of B that will no have any rows of zeros, i.e. T is a testor of B. □

Proposition 2.4.3 Let B be a basic matrix. Suppose T_1 and T_2 are any typical testors of two different blocks of B, with $T_1 \neq T_2$ and $T_1 \cap T_2 \neq \emptyset$, then the intersections of typical testors of blocks will not form typical testors of B.

Proof: Let $T = T_1 \cap T_2$. Without loss of generality assume there are one or more features that have been eliminated from T_1, then since T_1 is a minimal set if we eliminated any feature the resulting set is no longer a testor. Thus $B_{|T}$ will have one or more rows of zeros, i.e. T is not a testor of B. □

Following a similar argument as above, the operation of difference of sets is not of our interest because it can eliminate essencial features. From now on, we will focus only

on typical testors of B that are the result of the union of typical testors of the blocks of B.

Proposition 2.4.4 Let B be a basic matrix, B_1 and B_2 be blocks of B. Suppose T_1 is a typical testor of B_1 and T_2 is a typical testor of B_2. If T_1=T_2 then T_1 is a typical testor of B.

Proof: By Proposition 2.4.2, we know that T_1 is a testor of B since $T_1 \cup T_2 = T_1$. Because T_1 has the typical property in B_1 and B_2, then by Proposition 2.1 we can find an identity matrix in $B_{|T_1}$. Hence, T_1 has the typical property with respect to B and is a testor so T_1 must be a typical testor of B. □

2.4.2 YYC in the Blocks Approximation

The YYC algorithm has been designed to find the complete set of typical testors of a basic matrix, and since it builds typical testors up to a row i, it needs to reach the last row of the basic matrix to find its irreducible testors. This causes the algorithm to slow down if the number of rows is large. Also, since it uses the concept of finding compatible sets, the algorithm becomes slower as the number of compatible sets grows.

To deal with these disadvantages, we will apply the Blocks Approximation. We will partition a basic matrix into two blocks. We will apply the YYC algorithm to each of its blocks to obtain the complete set of typical testors of the blocks and then using these two sets as inputs we will find the typical testors of the entire matrix.

2.4.3 Blocks Approximation Algorithm

Given we divide the basic matrix into two blocks and we know the entire sets of typical testors of the blocks, we can test unions of elements of these sets to find irreducible testors.

By Proposition 2.4.2, we know that the union of typical testors of different blocks is a testor of the whole matrix. Thus, we only need to verify that the union has the typical property with respect to the entire basic matrix. To test whether the union results in a typical testor we can make use of Proposition 2.1 and formulate an extended version for our proposed method:

Proposition 2.4.5 Let B be a basic matrix. Suppose T_1 and T_2 are any typical testors of two different blocks of B. The set $T = T_1 \cup T_2$ has the typical property with respect to the matrix B if and only if we can obtain an identity matrix in $B_{|T}$ by exchanging and eliminating some rows. Futhermore, T is a typical testor of B.

Recall that mathematical unions of sets cannot guarantee the uniqueness property, we need to take this consideration into account to determine that each typical testor obtained by the union of typical testors of the blocks is unique in the final result of the blocks function. With this idea in mind, we proceed to write the algorithm 3.

Algorithm 3 Blocks Algorithm

Input: TB_1 complete set of typical testors of B_1 and TB_2 complete set of typical testors of B_2.

Output: TT complete set of tyical testors of the entire basic matrix B.

1: Define TT=$\emptyset$.
2: Define l_1 as the number of elements in TB_1.
3: Define l_2 as the number of elements in TB_2.
4: **for** <i=1 up to l_1> **do**
5: **for** <j=1 up to l_2> **do**
6: **if** Find compatible set between the testor i of the set TB_1 and the testor j of the set TB_2 **then** ▷ Programming based on Proposition 2.4.5.
7: **if** the union of testor i of the set TB_1 and the testor j of the set TB_2 is not in the set TT **then**
8: Add the union of testor i of the set TB_1 and the testor j of the set TB_2 to TT.
return TT

2.4.4 Parallelization of Blocks Algorithm

Technological innovation has advanced on a large scale in recent decades and one of the most recent areas of computational development is programming or parallel computing. Currently, the existence of computers with multicore processors or hyper-threading makes parallel computing more economically feasible [30].

Although in everyday programming it is not so common to apply parallel programming for computationally intensive processes, its use is highly recommended. This is why one of the advantages of the way the proposed algorithm works is that it can be parallelized and improve its performance.

For the Block approximation, the parallelization occurs in the phase in which the complete set of testors typical of the Blocks that have been defined is found. Instead of obtaining the complete set of typical testors sequentially, we can perform this task simultaneously by taking advantage of the available cores.

Due to the fact that this work is a first proposal of a more efficient algorithm, the proposed parallelization will serve for two blocks as the first stage.

Using *Matlab 2022a* and *Parallel Computing Toolbox*, the following algorithm can be defined that will perform the calculation of complete typical testors of the two blocks simultaneously.

Algorithm 4 Parallel Blocks Aproximation

Input: B complete basic matrix, $n = 2$ number of blocks, *ind* number of the last row of *Block* 1.

Output: $T = \{TT_1, TT_2\}$. Where TT_1 is the complete set of tyical testors of the B_1 and TT_2 is the complete set of tyical testors of *Block* 2.

1: Define the set T=$\emptyset$
2: Define *Block* 1: B_1 = BM(1:ind,:)
3: Define *Block* 2: B_2 = BM(ind+1:end,:)
4: **parfor** `<i=1 up to n>` ▷ *parfor* is a Matlab command, which executes for-loop iterations in parallel on workers.
5: Apply Algorithm 2.3.2 on B_i an saved in $T(i)$
return T

Now we present a diagram detailing step by step and graphically the process of the algorithm proposed in this work.

The first step is to divide the basic matrix into Blocks. It is important to mention that the Blocks have that are formed are disjoint and the number of rows that each block has can vary. For computational convenience the rows are chosen in an orderly manner.

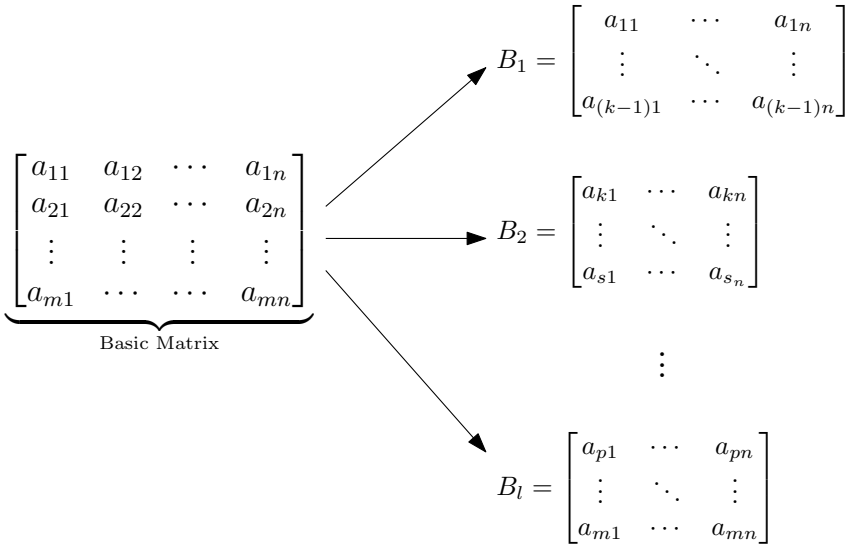

Figure 2.1: Step 1. Define Blocks

The second step consists in obtaining the set of typical testors for each block. This is the part of the algorithm that is going to be parallelized so that the calculations are done simultaneously.

$$
\begin{array}{lcl}
[\text{ Block } 1\ ] & \overset{YYC}{\longmapsto} & T_1 = \{\text{Typical testors of } B_1\} \\
[\text{ Block } 2\ ] & \overset{YYC}{\longmapsto} & T_2 = \{\text{Typical testors of } B_2\} \\
\vdots & & \vdots \\
[\text{ Block } l\ ] & \overset{YYC}{\longmapsto} & T_l = \{\text{Typical testors of } B_l\}
\end{array}
$$

Figure 2.2: Step 2. Find the typical testors for each block

The last step consists of taking all the sets of typical testors obtained in Step 2 and testing unions of irreducible testors of these sets to obtain typical testors for the complete basic matrix.

$$
\left.\begin{array}{l} T_1 \\ T_2 \\ \vdots \\ T_l \end{array}\right\} \text{Test unions typical testors of Blocks} \quad \overset{\text{Blocks Algorithm}}{\longmapsto} \quad T_F = \{\text{Typical testors of } B\}
$$

Figure 2.3: Step 3. Test unions of typical testors from Blocks

Once it has been detailed how the proposed algorithm works in each of its stages and the mathematical concepts that motivated its creation are clear, it is possible to move on to the experimentation phase in order to explore the strengths and limitations of the algorithm.

2.5 Experimental setup

With the aim of testing the proposed approach we used develop a set of synthetic basic matrices using operators [6], [19] and we used a real–world dataset taken from UCI machine learning repository [22].

2.5.1 Synthetic Basic Matrices

To test the proposed method, we construct a set of synthetic basic matrices that will help us to make the conclusions of this work. The advantage of using synthetic basic matrices is that we can know a prior the exact number of typical testors or minimal transversals [6], [11], [19]. So is a good practice to test algorithms with a set of synthetics matrices and this practice has become more common in recent years.

Definition 2.5.1 Let A and B be two basic matrices:

$$A = [a_{ij}]_{m\times n}, \quad B = [b_{ij}]_{m\times n'}$$

We define φ-Operator that acts over A and B in the following way:

$$\varphi(A, B) = [A\ B]$$

We call this operation *simple fusion*.

Definition 2.5.2 Let A and B be two basic matrices:

$$A = [a_{ij}]_{m\times n}, \quad B = [b_{ij}]_{m'\times n'}.$$

Then

$$\theta(A, B) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_{11} & \cdots & b_{1n'} \\ a_{11} & a_{12} & \cdots & a_{1n} & b_{21} & \cdots & b_{2n'} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{11} & a_{12} & \cdots & a_{1n} & b_{m'1} & \cdots & b_{m'n'} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_{11} & \cdots & b_{1n'} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_{m'1} & \cdots & b_{m'n'} \end{bmatrix}$$

We call this operation *combinatory fusion*.

Definition 2.5.3 Let A and B be two basic matrices:

$$A = [a_{ij}]_{m\times n}, \quad B = [b_{ij}]_{m'\times n'}.$$

We define the operator γ as:

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

One the most important things about using these operators in basic matrices is that the resultant matrix will be a basic matrix. All the operators are said to preserve the property of rows incomparability [6].

Because the resultant matrix is still a basic matrix and the arguments of the operators are basic matrices, then the operators are associative [19].

Notation: $\mathcal{O}^N(A)$ means the operator $\mathcal{O}$ has been aplied N times to the basic matrix A.

Let A be a basic matrix and $\Psi^*(A)$ be the complete set of irreducible testors of A. Let $\mathcal{C}_A = \{x_1, \cdots, x_n\}$ be the set of columns in A and let $x_j \in \mathcal{C}_A$. We denote $[x_j]_N = \{x_j, x_{j+n}, \cdots, x_{j+(N-1)n}\}$ the class of all columns in $\varphi^N(A)$ exactly equal to x_j. Given $S \subseteq \mathcal{C}_A$ and $S = \{x_{j_1}, x_{j_2}, \cdots x_{j_s}\}$, $[S]_N$ will denote the family of subsets of columns from $\varphi^N(A)$ that can be obtained by replacing one or more columns in S with another column in the same class.

Moreover, if A and B are basic matrices such that their sets of typical testors $\Psi^*(A)$ and $\Psi^*(B)$ are known, then the following propositions can be proved [19]:

Proposition 2.5.1 $\Psi^*(\varphi^N(A)) = \{[T]_N \mid T \in \Psi^*(A)\}$.

This proposition states that the set of typical testors of a N times concatenated matrix with itself is the set of all classes of typical testors of A [19]. Thus, we can previously know the number of typical testors of the matrix $\Psi^*(\varphi^N(A))$.

Proposition 2.5.2 $\Psi^*(\theta(A,B)) = \Psi^*(A) \cup \Psi^*(B)$.

The set of typical testors of the matrix $\theta(A,B)$ is the union of the complete set of testors of A and B.

Proposition 2.5.3 $\Psi^*(\gamma(A,B)) = \{T_A \cup T_B \mid T_A \in \Psi^*(A) \ \& \ T_B \in \Psi^*(B)\}$.

These three propositions give rise to 3 important corollaries about the number of testors typical of matrices to which operators have been applied [19], [6]:

Corollary 1 The number of typical testors of the matrix $\varphi^N(A)$ is given by:

$$\left|\Psi^*(\varphi^N(A))\right| = \sum_{T \in \Psi^*(A)} N^{|T|}$$

Corollary 2 The number of typical testors of the matrix $\theta(A, B)$ is:

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

Corollary 3 The number of typical testors of the matrix $\gamma(A, B)$ can be obtained using the formula:

$$|\Psi^*(\gamma(A, B))| = |\Psi^*(A)| \cdot |\Psi^*(B)|$$

We can now define three matrices on which the operators will act and will allow us to evaluate the proposed algorithm.

We define the matrix A, which has 4 rows and 5 columns as follows:

$$A = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

This matrix has the following set of typical testors:

$$\Psi^*(A) = \{\{x_1, x_2\}, \{x_1, x_3, x_4\}, \{x_1, x_5\}, \{x_2, x_5\}, \{x_3, x_5\}\}$$

Then we define the matrix B with the same dimensions as A, as follows:

$$B = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Which has the following set of typical testors:

$$\Psi^*(B) = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_2, x_5\}, \{x_1, x_3, x_5\}, \{x_1, x_4, x_5\}\}$$

Finally, we define the identity matrix I_5 of dimensions 5×5 to run the propose method and test it.

$$I_5 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The complete set of typical testors of I_5 is:

$$\Psi^*(I_5) = \{\{x_1, x_2, x_3, x_4, x_5\}\}$$

The basic matrices defined above will serve as the basis for performing operations with the operators defined previously to obtain a set of test matrices that is varied enough to allow us to make relevant conclusions about the Blocks algorithm. We will find the typical testors of the test matrices in two ways, using the Blocks algorithm and using YYC.

Once the set of test matrices is obtained, we will characterize each of them to then apply the YYC algorithm and record its execution time in seconds. Then we will use the block algorithm. This algorithm will be applied to two different partitions in order to be able to conclude if the way in which we build the blocks is relevant or not. For the first partition we will take the top half of the rows for *Block* 1 and the bottom half of the rows for *Block* 2 . For Partition 2, a proportion of 1/4 of the top rows of the complete basic matrix will be used for *Block* 1 and the rest of the rows for *Block* 2.

For each partition we must record the time it takes for the YYC algorithm to obtain the typical testors of *Block* 1 and then the time it takes to find the typical testors of *Block* 2. Then these sets will be the arguments for the block algorithm. For which it is also necessary to register the time it takes to execute. It must be taken into account that this process will be sequential and that the total execution time will be the sum of these times measured in seconds.

On the other hand, it is necessary to mention that the codes used for this work are implemented in the *Matlab* 2022*a* platform and the experiments carried out were executed on a MacBook Pro (13-inch, M1, 2020) personal computer with macOS Monterey operating system and memory of 8GB.

2.5.2 Parallel Computing and Basic Synthetic Matrices

After performing the sequential tests for the Blocks Algorithm (3), tests will be done implementing parallelization using algorithm 4. The time it takes for the sequential and parallel Blocks algorithm and the YYC algorithm to find the complete set of irreducible testors of a basic synthetic matrix will be compare.

2.5.3 Dataset from UCI Machine Learning Repository

To test the performance of the algorithm proposed in this work in application problems, a dataset from the UCI Repository [22] will be used. The UCI Machine Learning Repository is a collection of open acces databases that are used by the machine learning community for the empirical analysis of machine learning algorithms. The archive was created as an ftp archive in 1987 by David Aha and fellow graduate students at UC Irvine. Since that time, it has been widely used by students, educators, and researchers all over the world as a primary source of machine learning data sets [22].

The dataset selected from UCI Repository is called QSAR biodegradation. This data set was built in the Milano Chemometrics and QSAR Research Group. They study chemical stricture and biodegradation molecules [23].

This database contains 1055 instances, each with 41 features that are molecular descriptors. There are two classes: ready biodegradable molecules (356) and non-ready (699). Therefore, the labels for the classes are: RB (Ready Biodegradable) and NRB (Not ready biodegradable).

A treatment process must be carried out before the Blocks algorithm can be applied. The differentiation matrix described in Section 2 in definition 2.1.1 must be implemented.

Once the differentiation matrix M_D is obtained, it must be reduced to a basic matrix. When the basic matrix is obtained, the separation into two blocks is carried out. For simplicity, only one partition will be used: that of dividing the matrix in half (refered as Partition 1). With the blocks of the basic matrix defined, the YYC algorithm is applied to obtain the complete sets of irreducible testors for each block.

With the set of typical testors, one neural network will be trained by using only the characteristics of one typical testor chosen. Another neural network will be trained using the full set of features. Neural networks were programmed in Python 3.9 [37] using libraries: Tensor [39] and Keras [38]. The execution times of the Block Algorithm were recorded both in its parallel and sequential version.

Chapter 3

RESULTS AND DISCUSSION

In this chapter the Blocks Algorithm is applied to a set of basic synthetic matrices and also to a real base taken from the UCI repository [22]. The execution times of the Blocks Algorithm are recorded and the possible reasons for its fast or inefficient execution compared to the YYC Algorithm are discussed.

For each matrix of the set of synthetic test matrices elaborated in Section 2 and detailed in 2.5.1, there is a section in which a reduced table is shown with the most relevant results that allow us to perform an analysis of the efficiency of the algorithm proposed by each operator.

After performing the sequential tests for the Block Algorithm, we implement the parallelization described in algorithm 4 for a reduced set of the synthetic basic matrices defined in Section 2.5.1.

At the end of this chapter, the efficiency of the Block Algorithm for a real database is analyzed as part of application problems.

3.1 Evaluation Results Synthetic Matrices

The tables presented in this section are simplified versions of the results obtained. The first column tells us the number of times the operator $\mathcal{O}$ was applied to the basic matrix M. Columns two, three and four are characteristic of the basic matrix $X = \mathcal{O}^N(M)$. Then TT column refers to the number of typical testor for the matrix X. The column labeled YYC records the time in seconds that it took this algorithm to find all the typical

testors of X. The column labeled Partition 1 tells us the total time (in seconds) the Blocks Algorithm ran with the first partition. The Partition 2 column indicates the total time the Blocks Algorithm was executed with the second partition.

The complete tables of results can be found in the Annexes for each matrix for each applied operator.

3.1.1 Basic Matrix I_5 Experiments

I_5 is a basic matrix of dimension 5x5 with a density of ones of xxxx. This matrix has a single typical testor of cardinality 5. To this matrix we will apply each of the operators defined in this section.

Operator γ applied to I_5

By applying the γ operator recursively to the matrix I_5 the resulting matrix will increase the number of rows and columns alike but will keep a single typical testor. The density of the resulting matrix will be less than that of I5.

Now the table with results is presented:

$\gamma^N(I_5)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	5	5	0.2	1	0.005368	0.031299	0.031299
2	10	10	0.1	1	0.005935	0.019170	0.024763
3	15	15	0.0667	1	0.006892	0.013305	0.010420
4	20	20	0.05	1	0.006095	0.007149	0.006432
5	25	25	0.04	1	0.010865	0.009869	0.008085
6	30	30	0.0333	1	0.017441	0.010271	0.013504
8	40	40	0.025	1	0.031454	0.015096	0.009752
10	50	50	0.02	1	0.030361	0.015599	0.011771

Table 3.1: Results for I_5 using the γ operator.

Based on the results obtained, in this case it can be seen that the Block algorithm is much more efficient than YYC algorithm when this operator is applied and N increases.

It is observed in graph 3.1 below that a break point at N=4, where the block algorithm becomes more efficient. This can be explained by the fact that the matrix has low density and only one typical testor. This fact favors the Blocks algorithm since there

are not many unions to test, which is evidenced on the tables in the Annex Section 5: 5.1 and 5.4. In them, the number of testors obtained in each block for each partition is recorded. On the other hand, there is no clear pattern in terms of the efficiency of one partition or the other.

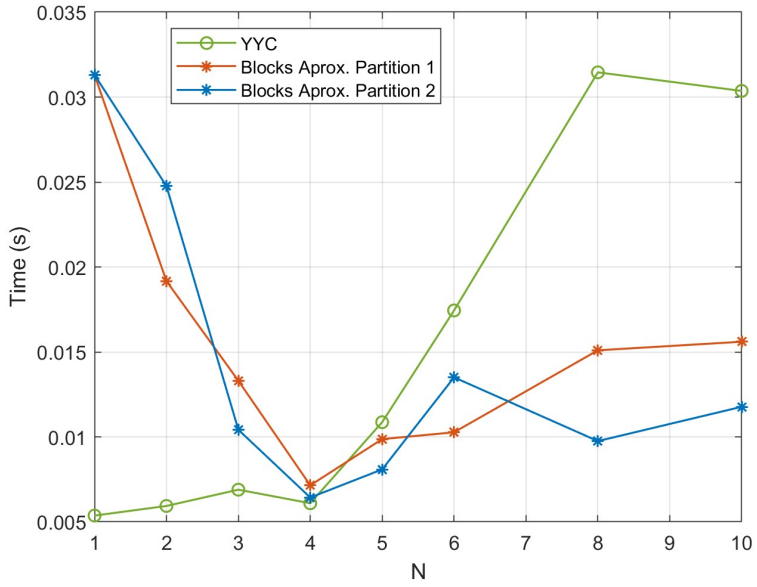

Figure 3.1: Results for I_5 using the γ operator.

Operator θ applied to I_5

By applying the θ operator recursively to the matrix I_5 the resulting matrix will increase the number of rows and columns. The resulting matrices will have a very low density, a large number of rows compared to the number of columns, and few typical testors.

Following the table with results is presented:

$\theta^N(I_5)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	5	5	0,2	1	0,003416	0,031299	0,031299
2	25	10	0,08	2	0,059171	0,07695	0,025822
3	125	15	0,024	3	0,05546	0,094993	0,071902
4	625	20	0,0064	4	0,288344	0,291037	0,329386
5	3125	25	0,0016	5	4,557188	3,835996	4,153209
6	15625	30	0,000384	6	425,640116	143,165474	166,028161

Table 3.2: Results for I_5 using the θ operator.

Below is a graph where you can see the results obtained for this matrix when applying this operator:

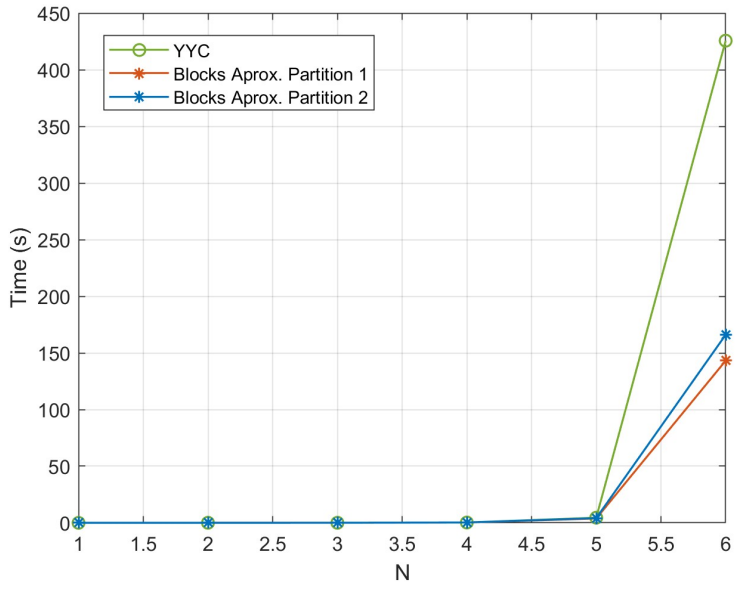

Figure 3.2: Results for I_5 using the θ operator.

The Blocks Algorithm turns out to be more efficient than the YYC as this operator is applied. In this case the number of iterations that the Blocks algorithm had to perform was relatively small since the number of typical testors found in each of the blocks (regardless of which partition) was no more than 11, which was computationally more efficient than perform the full iterations over the number of rows. To see these values in more detail, you can refer to the Annexes 5 section in the 5.2 and 5.5 tables. On the other hand, no definite pattern is observed as to which partition turns out to be more efficient. The Blocks algorithm for the two partitions was able to find the complete set of irreducible testors for the matrix $\theta^N(I_5)$.

Operator φ applied to I_5

This operator generates matrices in which the number of rows does not vary. For this reason we will only apply Partition 1 (half blocks) for the Blocks Algorithm. The number of columns increases as we apply the operator. The generated matrices keep their density constant but have a large number of typical testors.

Next the table with results is presented:

$\varphi^N(I_5)$	Rows	Columns	Density	TT	YYC	Partition 1
1	5	5	0,2	1	0,003416	0,031299
2	5	10	0,2	32	0,014124	0,034755
3	5	15	0,2	243	0,02497	0,330449
4	5	20	0,2	1024	0,049103	4,168582
5	5	25	0,2	3125	0,115917	37,331452
6	5	30	0,2	7776	0,235226	232,989829
8	5	40	0,2	32768	0,742539	14400<
10	5	50	0,2	100000	2,226354	36000<

Table 3.3: Results for I_5 using the φ operator.

Now a graph with the results is shown:

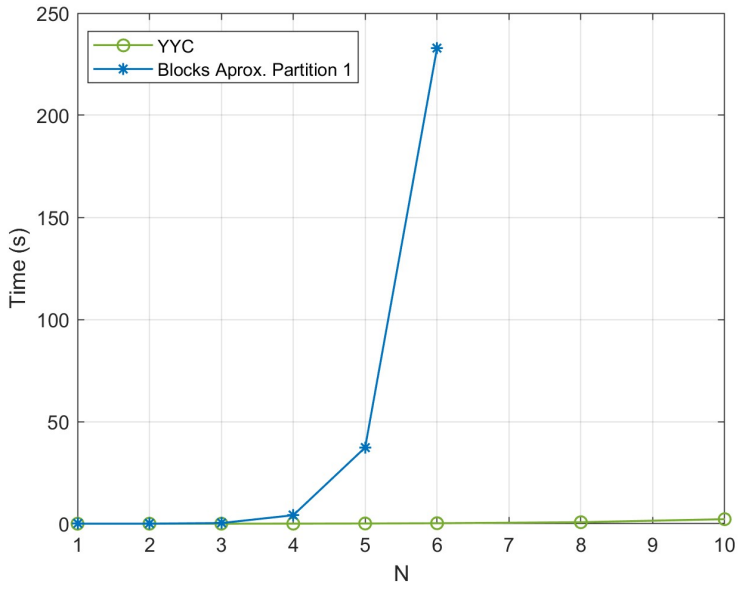

Figure 3.3: Results for I_5 using the φ operator.

For this case the Blocks algorithm turns out to be very impractical. These matrices are relatively small in dimension so the YYC algorithm runs very efficiently and in less time than Blocks. This is due to the fact that there are a large number of typical testors in matrices that do not exceed 5 rows.

The Block algorithm turns out to be computationally inefficient since it must perform a large number of iterations to find the unions of irreducible testors of each Block that will

be typical testors of the entire matrix. For example, for $N = 5$, the Blocks algorithm must perform about 3125 iterations, which takes much longer than the 5 iterations performed in YYC. For a more detailed report of these values, you can go to the Annexes Section 5, in the table 5.3. The Blocks algorithm found the complete set of typical testors for the matrices $\varphi^N(I_5)$.

3.1.2 Basic Matrix A Experiments

The basic matrix A has dimension 4x5 with a density of ones of $0,55$. This matrix has 5 typical testors, 4 of cardinality 2 and one of cardinality 3. To this matrix we will apply the operators defined in this section.

Operator θ applied to A

The generated matrices maintain a large number of rows compared to the number of columns. The density gradually decreases and the number of testors increases progressively as we apply the operator. Below is the table with the recorded times:

$\theta^N(A)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	4	5	0,55	5	0,029548	0,160351	0,160351
2	16	10	0,3438	10	0,037219	0,080564	0,045398
3	64	15	0,1289	15	0,040026	0,066707	0,070502
4	256	20	0,043	20	0,168861	0,228052	0,222462
5	1024	25	0,0134	25	1,982099	1,493741	1,391179
6	4096	30	0,004	30	35,945845	22,669527	21,262211
7	16384	35	0,0012	35	762,991133	445,540981	419,779761

Table 3.4: Results for A using the θ operator.

For this group of matrices, the Blocks algorithm turns out to be much more efficient than the YYC algorithm since the number of rows is large compared to the number of typical testors per Block. This means that it is computationally faster to check which unions of irreducible testors give typical testors of the entire basic matrix than to search for row-by-row irreducible testors for the entire basic matrix.

One of the reasons why the proposed algorithm turns out to be more efficient is that the number of typical testors of the blocks turns out to be small compared to the number of rows of the blocks. Furthermore, as N grows the Blocks algorithm is increasingly favored by this fact.

This can be seen in the comparative graph of the algorithms times vs. the number of times the operator below:

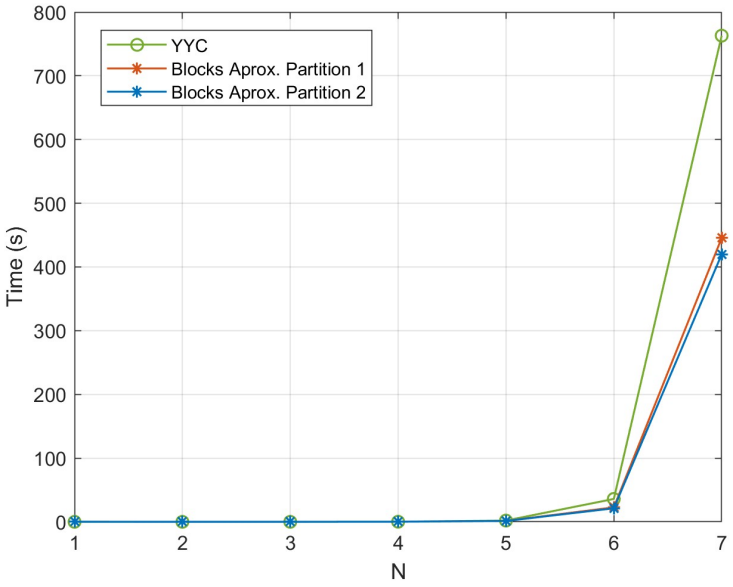

Figure 3.4: Results for A using the θ operator.

For example, for $N = 6$, the YYC algorithm must perform 4096 iterations while the Block Algorithm performs around 784.

To observe the much more detailed report of these values, you can go to the Annexes Section 5 in table 5.6 and in table 5.8. Furthermore, for each Partition the Blocks algorithm was able to find the complete set of irreducible testors for the matrix $\theta^N(A)$. And, in this particular case Partition 2 turns out to be approximately 1% more efficient than Partition 1. Which indicates that there is no significant gain in terms of efficiency when performing one partition or the other.

Operator φ applied to A

This operator generates matrices in which the number of rows does not vary. For this reason we will only apply Partition 1 (half blocks) for the Blocks Algorithm. The number of columns increases as we apply the operator. The generated matrices keep their density constant but have a large number of typical testors regarding the dimension of the resulting matrices. The table with the experimental results obtained is presented:

$\varphi^N(A)$	Rows	Columns	Density	TT	YYC	Partition 1
1	4	5	0,55	5	0,029548	0,160351
2	4	10	0,55	24	0,00652	0,041484
3	4	15	0,55	63	0,008336	0,0782
4	4	20	0,55	128	0,010053	0,17203
5	4	25	0,55	225	0,01791	0,413693
6	4	30	0,55	360	0,025413	0,736256
8	4	35	0,55	768	0,044329	14400<
10	4	40	0,55	1400	0,064266	36000<

Table 3.5: Results for A using the φ operator.

A comparative graph between the execution time of the algorithms vs. the number of times the operator was applied is displayed:

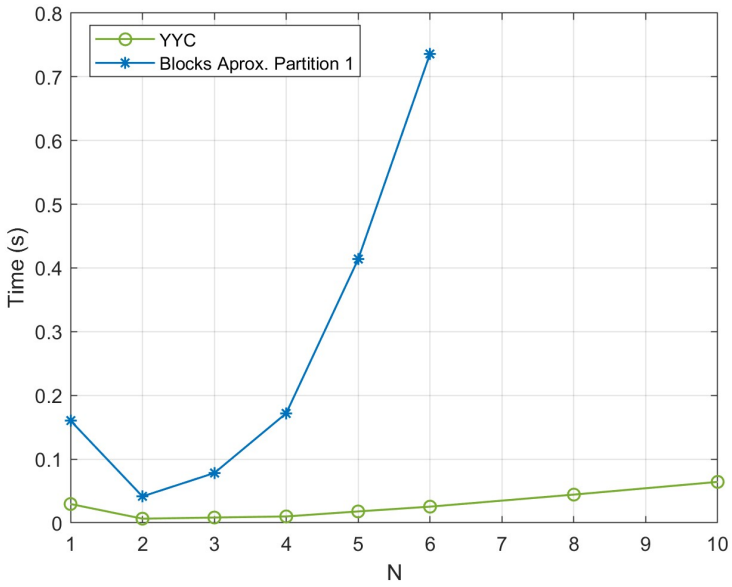

Figure 3.5: Results for A using the φ operator.

For this group of matrices, the most efficient algorithm turns out to be the YYC. This is because the number of rows is small (four rows) but these matrices have a large number of irreducible testors. Furthermore, each Block has a large number of typical testors, which implies that the number of unions to be tested to determine the irreducible testors of the complete basic matrix is greater than iterating over the rows, which makes this process less efficient. However, despite being inefficient, the Block Algorithm managed

to find the complete set of testors typical of $\varphi^N(A)$.

3.1.3 Basic Matrix B Experiments

The basic matrix B has dimension 4x5 with a density of ones of $0,50$. This matrix has 5 typical testors, three testors of cardinality 2 and two of cardinality 3. To this matrix we will apply the operators defined in this section.

It is worth mentioning that to obtain matrices A and B, an algorithm was used to reduce a matrix with random binary inputs to a basic matrix.

Operator θ applied to B

When we apply this operator to matrix B, we obtain matrices with a large number of rows compared to the number of columns. And the amount of testors as well as the density are small.

Now the reduced table of results is presented to compare the algorithms:

$\theta^N(B)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	4	5	0,5	5	0,00475	0,030756	0,030756
2	16	10	0,3125	10	0,008634	0,033356	0,040685
3	64	15	0,1172	15	0,066651	0,092834	0,092404
4	256	20	0,0391	20	0,552101	0,391459	0,399752
5	1024	25	0,0122	25	11,563568	5,175375	5,608347
6	4096	30	0,0037	30	303,740156	121,208505	135,161253

Table 3.6: Results for B using the θ operator.

The Blocks algorithm turned out to be much more efficient than the YYC algorithm. This fact is explained by looking at the number of rows in these matrices vs. the number of typical testors per block and in the entire basic matrix. The number of rows per block turns out to be larger than the number of typical testors, which indicates that it takes more time to finish the iterations by rows than to test the unions of typical testors of the blocks. To observe these values in greater detail, you can go to the Annexes Section 5 in the tables: 5.9 and 5.11.

In this case, the Blocks algorithm found (for each partition) the full set of typical testors and Partition 1 turned out to be about 1% more efficient. An efficiency greater

than 1% does not give us great indications that in practice it is really beneficial to use Partition 1.

A graph is shown where the efficiency of each algorithm is evidenced:

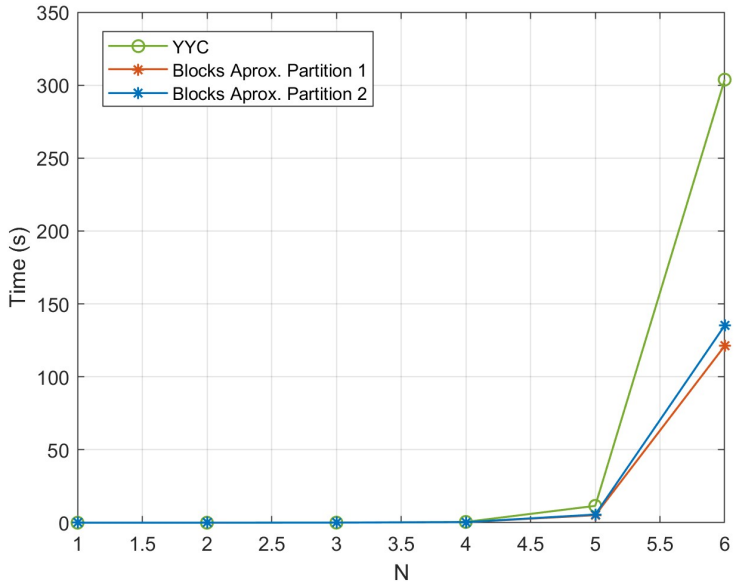

Figure 3.6: Results for B using the θ operator.

Operator γ applied to B

Applying this operator to this basic matrix results in a small progressive increase in the number of columns and rows in the resulting matrices. The density progressively decreases while the number of typical testors increases greatly.

$\gamma^N(B)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	4	5	0,5	5	0,00475	0,030756	0,030756
2	8	10	0,25	25	0,011609	0,026731	0,022235
3	12	15	0,1667	125	0,037024	0,12944	0,132935
4	16	20	0,125	625	0,107366	1,566974	1,574165
5	20	25	0,1	3125	0,438356	44,119327	44,082993
6	24	30	0,0833	15625	22,81533	21600<	21600<

Table 3.7: Results for B using the γ operator.

A graph is made using the results obtained:

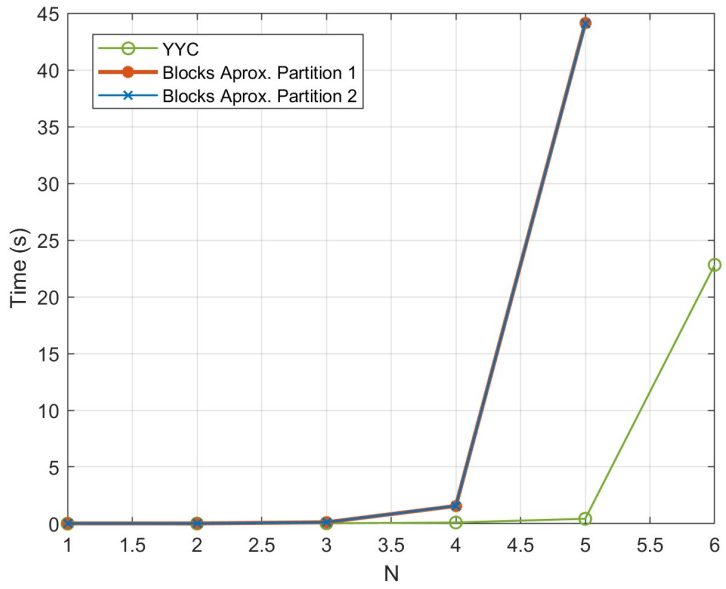

Figure 3.7: Results for B using the γ operator.

Since the number of rows in this family of matrices is small compared to the number of testors, the YYC algorithm turns out to be much more efficient. The Block Algorithm turns out to be impractical since there are a large number of iterations to perform due to the large number of typical testors found in each Block for each of the Partitions made. For more detail, review the 5.10 and 5.12 tables in the Annexes Section 5.

As seen in the graph 3.7 there is no clear evidence that there is a significant difference in terms of efficiency in terms of one Partition or the other. The Blocks algorithm found the complete set of typical testors for each matrix of this family.

3.1.4 Basic Matrix M_1 Experiments

From the basic matrices defined in section 2.6.1, we proceed to define another basic matrix resulting from the application of an operator in order to obtain greater variety in the experimentation.

Let M_1 be the following:

$$M_1 = \theta(A,B) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_10 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

M_1 is a basic matrix of dimension 16x10 with a density of ones of $0,3281$. This matrix has 10 irreducible testors, seven have cardinality 2 and three have cardinality 3. To this matrix we will apply each of the operators defined in this section.

Operator γ applied to M_1

When the γ operator is applied to the matrix M_1, matrices with a progressive increase in the number of rows and columns are obtained. The density decreases but the number of typical testors increases greatly.

Now the table with the results is displayed:

$\gamma^N(M_1)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	16	10	0,3281	10	0,093533	0,242545	0,242545
2	32	20	0,1641	100	0,128282	0,149861	0,199808
3	48	30	0,1094	1000	0,84736	4,880646	5,24383
4	64	40	0,082	10000	10,647505	338,500018	385,16331
5	80	50	0,0656	100000	135,544081	18000<	18000<

Table 3.8: Results for M_1 using the γ operator.

Using the information obtained we can make a graph to better visualize the results:

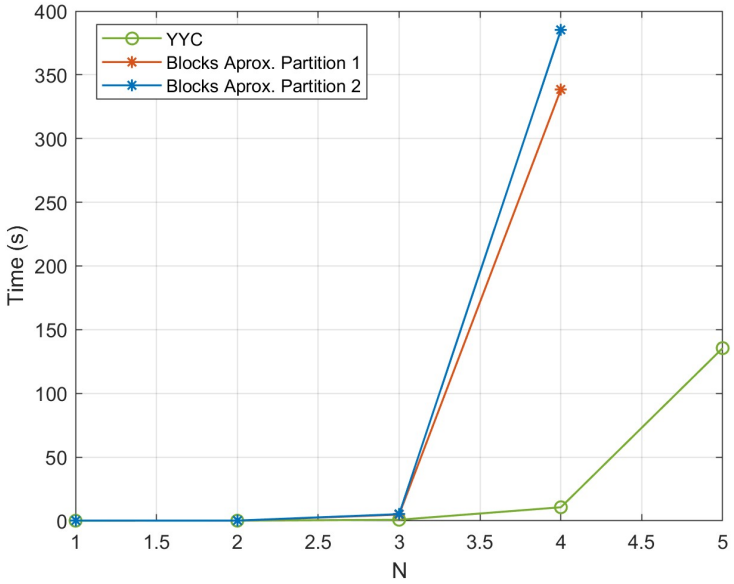

Figure 3.8: Results for M_1 using the γ operator.

The Blocks algorithm turned out not to be the most efficient in this case. The YYC algorithm was able to obtain the complete set of typical testors without problems since the matrices treated are relatively small in dimension despite having a large number of irreducible testors. As can be seen in the Annexes section 5 in the table 5.13 and in the table 5.16, as N increases for each block, the number of typical testors found by Block no matter what partition has been made. This implies a large number of iterations that the Block Algorithm must perform, which affects its computational efficiency and makes it impractical in this type of matrix.

It should be noted that the Block Algorithm was able to find the complete set of typical testors of the complete basic matrix. Although the Block Algorithm was not the most efficient, Partition 1 turned out to be much faster computationally than Partition 2.

Operator θ applied to M_1

In this case, when applying this operator, we obtain resulting matrices whose number of rows increases much more than the number of columns. The density is low and the number of irreducible testors increases progressively.

Now the table with the results is displayed:

$\theta^N(M_1)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	16	10	0,3281	10	0,093533	0,242545	0,242545
2	256	20	0,041	20	0,29709	0,34286	0,287528
3	4096	30	0,0038	30	100,458358	62,073514	58,046076

Table 3.9: Results for M_1 using the θ operator.

Using the results obtained, a comparative graph of the execution time of the algorithms can be made:

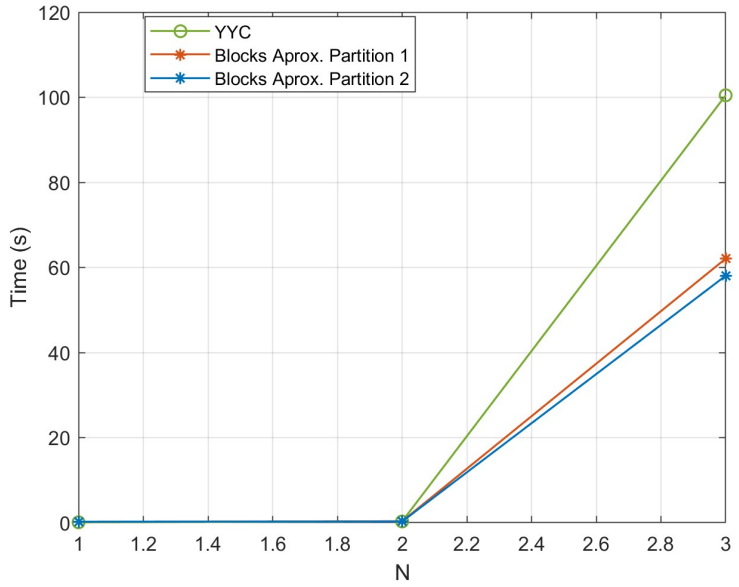

Figure 3.9: Results for M_1 using the θ operator.

To obtain the set of typical testors by applying the θ operator iteratively on the matrix M_1, it turns out to be much more efficient to use the YYC Algorithm. The proposed algorithm was able to find the complete set of typical testors of the entire matrix in less time than the YYC algorithm. This fact may be due to the fact that the matrices have a large number of rows but few typical testors in general and few irreducible testors in their blocks. This implies that computationally fewer iterations are done to check which unions of typical Block testors give testors for the entire basic matrix than to search for testors row by row of the entire matrix. Also since the $N = 2$ case, Partition 2 turned out to be more efficient.

Operator φ applied to M_1

By applying this operator the number of rows and the density remain constant in the resulting matrices. But the number of typical testors greatly increases.

Following is the table with the results obtained:

$\varphi^N(M_1)$	Rows	Columns	Density	TT	YYC	Partition 1
1	16	10	0,3281	10	0,093533	0,242545
2	16	20	0,525	52	0,035331	0,136302
3	16	30	0,525	144	0,075568	0,598597
4	16	40	0,525	304	0,197799	2,12069
5	16	50	0,525	550	0,408643	6,354836
6	16	60	0,525	900	0,807671	16,539908

Table 3.10: Results for M_1 using the φ operator.

Below is a comparative graph of the algorithms:

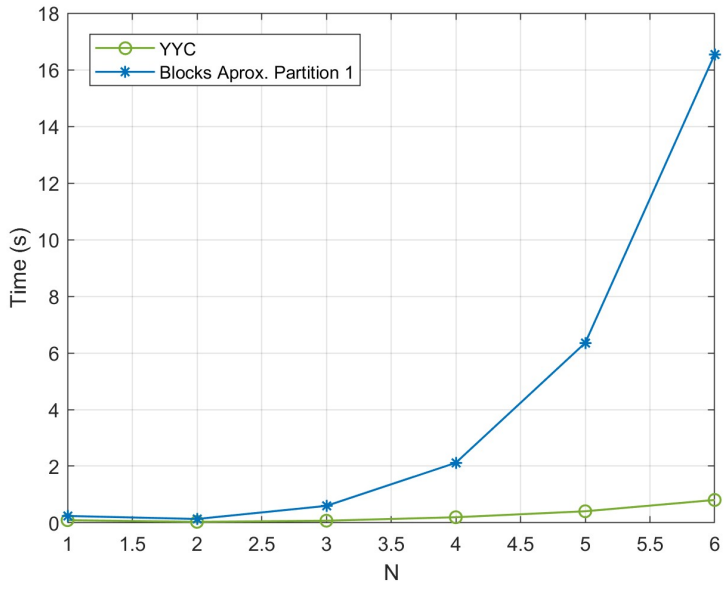

Figure 3.10: Results for M_1 using the φ operator.

Because there are a large number of typical testors within each block, the Block Algorithm is impractical. It is much more computationally efficient to do all 16 row

iterations than to check which joins will give us a typical testor for the entire matrix. However, the Blocks algorithm found the complete set of irreducible testors for $\varphi^N(M_1)$. To verify the number of testors found for each Block as well as more information on these results, you can visit the Annexes Section 5, in particular the table 5.15.

3.1.5 Basic Matrix M_2 Experiments

Using the matrices defined in section 2.6.1, let M_2 be the following basic matrix:

$$M_2 = \gamma(A,B) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & X_9 & X_{10} \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

M_2 is a basic matrix of dimension 8x10 with a density of ones of $0,2625$. This matrix has 25 irreducible testors. M_2 has twelve typical testors of cardinality 4, eleven typical testors of cardinality 5, and two testors of cardinality 6. To this matrix we will apply each of the operators defined in this section.

Operator γ applied to M_2

By applying this operator to M_2, resulting matrices with a greater number of rows and columns are obtained. The number of typical testors increases very fast and they are matrices with a large number of testors in relation to their size. The density progressively decreases.

In this case, the Block Algorithm does not turn out to be efficient to find the typical testors of the matrices of the family $\gamma^N(M_2)$ as can be seen in graph 3.13 and table 3.13.

This can be explained by the fact that they are relatively small dimension matrices but have a large number of typical testors. If you look at the tables 5.18 and 5.21 in the Annexes Section 5, you can see that in each block there is a large number of irreducible testors (regardless of which partition it is). Which implies that a greater number of

iterations must be carried out to test the unions that will serve as typical testors for the entire matrix than to traverse the entire matrix through all its rows. It is worth mentioning that the Block Algorithm was able to find the complete set of typical testors for the basic matrix when it was executed. Furthermore, with the recorded results it is not possible to identify if there is a Partition that is much more efficient than the other.

$\gamma^N(M_2)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	8	10	0,2625	25	0,013787	0,015825	0,015825
2	16	20	0,1313	625	0,157914	1,717367	1,624205
3	24	30	0,0875	15625	2,165658	938,675906	1165,59522
4	32	40	0,0656	390625	66,21621	25200<	25200<

Table 3.11: Results for M_2 using the γ operator.

A comparative graph of the efficiency (measured in seconds) of the tested algorithms is available:

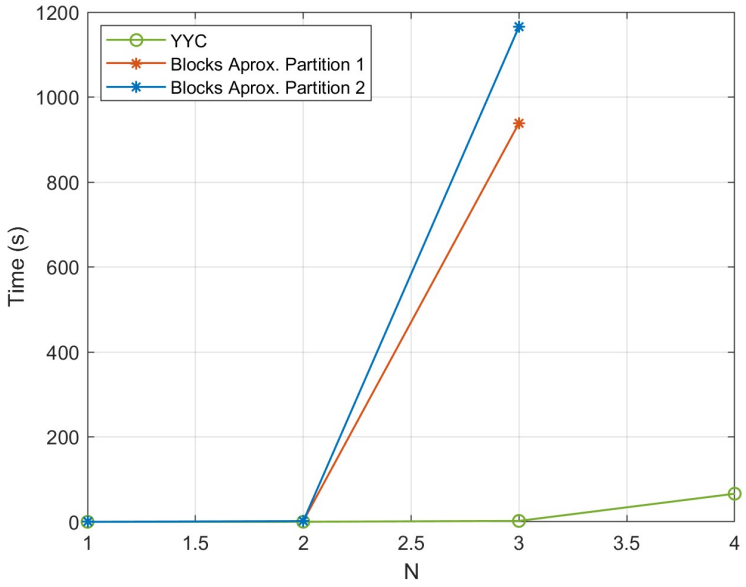

Figure 3.11: Results for M_2 using the γ operator.

Operator θ applied to M_2

If the θ operator is applied to M_2, the density decreases but both the number of rows and columns increases in small ranges. However, the number of typical testors increases

sharply and stops the size of the resulting matrices.

Following is the table with the results obtained:

$\theta^N(M_2)$	Rows	Columns	Density	TT	Time	Partition 1	Partition 2
1	8	10	0,2625	25	0,013787	0,015825	0,015825
2	64	20	0,0820	50	0,11095	1,60673	1,585995
3	512	30	0,0154	75	2,18726	949,077128	1174,87477
4	4096	40	0,0026	100	117,796907	21600<	21600<

Table 3.12: Results for M_2 using the θ operator.

Now a graph is presented that helps to compare the efficiency of the tested algorithms:

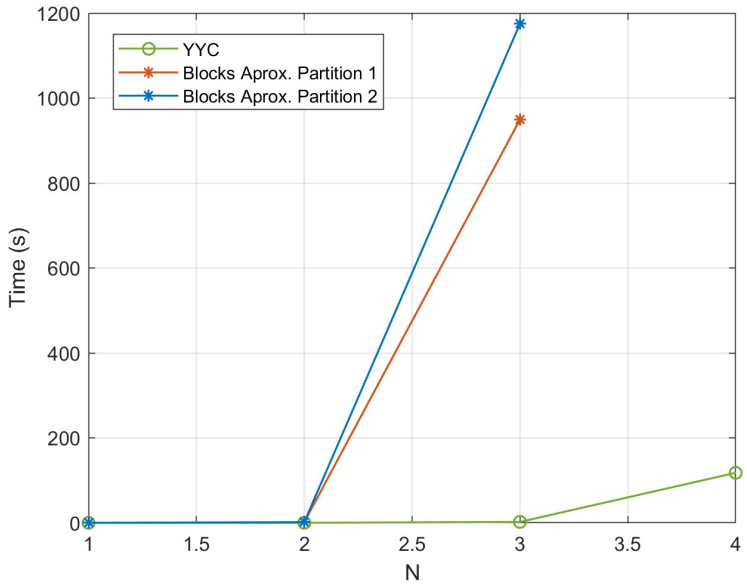

Figure 3.12: Results for M_2 using the θ operator.

The Block Algorithm turns out to be inefficient and the best computational performance to find the typical testors of the matrices of the family $\theta^N(M_2)$ is presented by the YYC algorithm. This is because the arrays are small in dimension and have a large number of testors. As can be seen in the Appendix Section 5 in the tables 5.19 and 5.22 for any partition, as N grows the number of typical testors for each block whose joins must be tested grows on a large scale. Which generates that there is a large number of iterations that the Block Algorithm must perform to obtain the typical testors of the entire matrix.

For the number of results obtained in this case, there is no clear pattern that one partition turned out to be much more efficient than the other. When the Block Algorithm was fully executed, it was able to find the complete set of typical testors for the matrix $\theta^N(M_2)$.

Operator φ applied to M_2

By applying this operator the number of rows and the density remain constant in the resulting matrices. The number of columns increases from 10 to 10. And the number of typical testors increases greatly for the dimensions of the resulting matrices.

$\varphi^N(M_2)$	Rows	Columns	Density	TT	Time	Partition 1
1	8	10	0,2625	25	0,013787	0,015825
2	8	20	0,2625	672	0,068181	1,817258
3	8	30	0,2625	5103	0,262985	100,825489
4	8	40	0,2625	22528	0,957787	1974,79877

Table 3.13: Results for M_2 using the φ operator.

Below is a graph based on the results in the table 3.13:

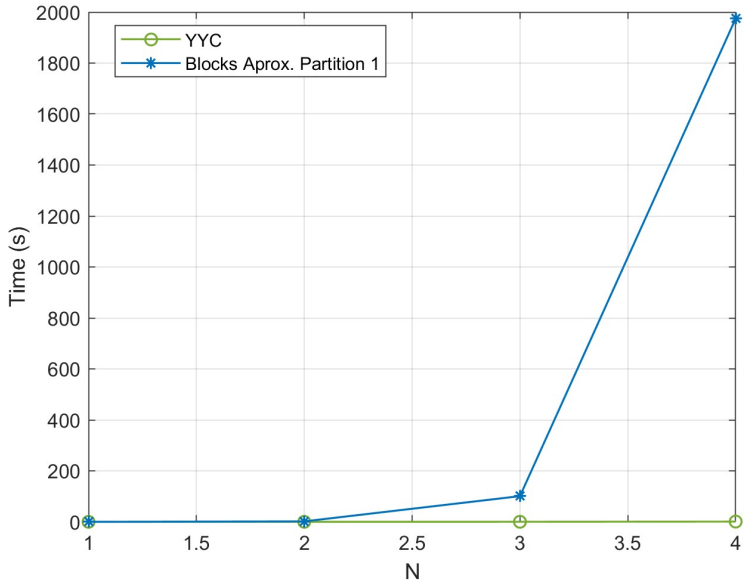

Figure 3.13: Results for M_2 using the φ operator.

The YYC Algorithm presented a better performance in this case. Although the Blocks Algorithm was able to find the full set of typical testors for $\varphi^N(M_2)$, its performance was quite slow compared to the YYC Algorithm. As can be seen in the table 5.20 of the Annexes Section 5, the number of typical testors found per block was high. This implies that a large amount of time must be spent testing each of the possible joins, which greatly slows down the execution time of the Blocks Algorithm. These observations can be verified by looking at the graph shown below.

3.1.6 Basic Matrix M_3 Experiments

A new basic matrix M3 is defined using the operators defined in section 2.6.1:

$$M_3 = \varphi(A, B) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

M_3 is a basic matrix of dimension 4x10 with a density of ones of $0,5250$. This matrix has twenty-four irreducible testors, fifteen of cardinality 2 and nine of cardinality 3. To this matrix we will apply each of the operators defined in this section.

Operator γ applied to M_3

By applying this operator to this basic matrix, a set of resulting matrices is obtained whose number of rows and columns grows progressively and density decreases as N increases. They are small matrices in relation to the number of irreducible testors they have.

Following is the table with the results obtained:

$\gamma^N(M_3)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	4	10	0,525	24	0,049014	0,060935	0,060935
2	8	20	0,2625	576	0,063097	1,358063	1,701933
3	12	30	0,175	13824	0,968452	863,766025	848,675431
4	16	40	0,1313	331776	26,494519	21600<	21600<

Table 3.14: Results for M_3 using the γ operator.

Below is a comparative graph of the computational efficiency of the algorithms based on the results of the table above:

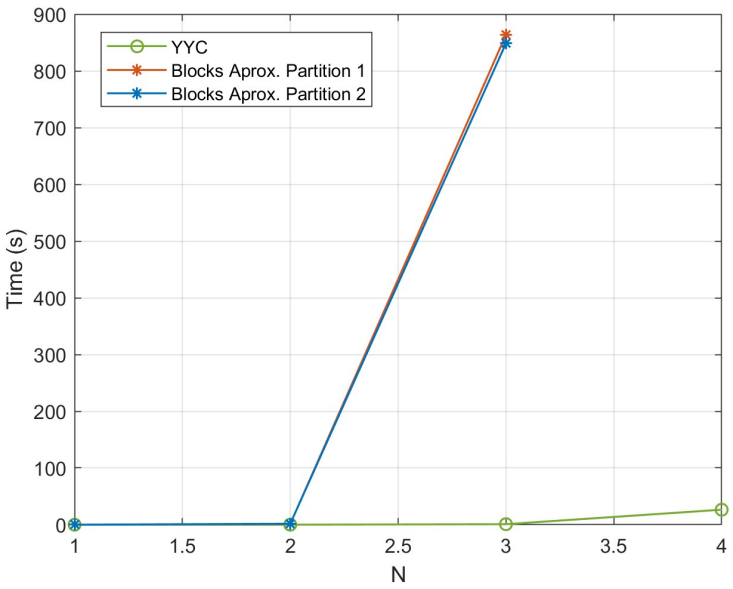

Figure 3.14: Results for M_3 using the γ operator.

Of the tested algorithms, the most efficient is YYC. This algorithm found in much smaller execution times the complete set of irreducible testors for each matrix $\gamma^N(M_3)$.

Although the Blocks Algorithm was able to accomplish the same goal, the time it finished performing this task was slowed down by the number of typical testors found in each Block for each partition, as can be seen in the tables 5.23 and 5.26 of the Annexes Section. This means that many union tests must be performed in order to find typical testors of the entire matrix. On the other hand, the matrices of the family $\gamma^N(M_3)$ have a very small number of rows compared to the number of irreducible testors. This fact makes it a better strategy to search for irreducible testors by row than to search for them in the possible unions of typical testors of the blocks. On the other hand, there is no clear evidence that one of the two partitions was more efficient than the other.

Operator θ applied to M_3

If the θ operator is applied to the basic matrix M_3, the resulting matrices are obtained whose number of rows grows much faster than the number of columns. The density drops and the number of testors increases as N increases.

Now the table with the results is displayed:

$\theta^N(M_3)$	Rows	Columns	Density	TT	YYC	Partition 1	Partition 2
1	4	10	0,525	24	0,049014	0,060935	0,060935
2	16	20	0,525	48	0,08538	0,137144	0,127581
3	64	30	0,2461	72	0,41647	0,45491	0,438584
4	256	40	0,082	96	14,806416	6,03836	6,984725
5	1024	50	0,0256	120	806,866905	265,149842	326,694497

Table 3.15: Results for M_3 using the θ operator.

A comparative graph of the performance of the algorithms is displayed:

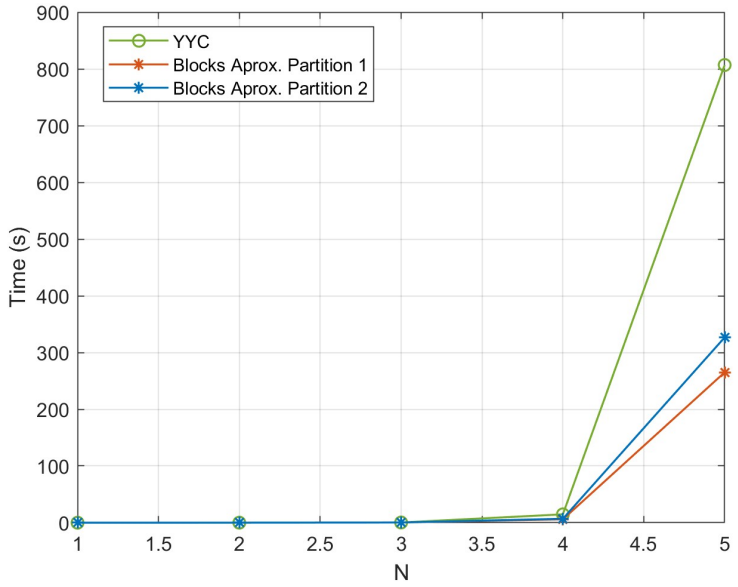

Figure 3.15: Results for M_3 using the θ operator.

The Blocks Algorithm turns out to be the one with the best computational performance to find the typical testors of the $\theta^N(M_3)$ matrices for each N. This is a consequence of the fact that the matrices are of large dimension compared to the number of irreducible testors they have. As can be seen in the results table, as N grows the number of rows grows at a large scale, which slows down the YYC algorithm. If it is observed in the Annexes Section 5, the table 5.24 for Partition 1, it is evident that the number of typical testors found for each block is less than the number of rows of the block since the case $N = 3$ which favors the performance of the proposed algorithm. The Block Algorithm was able to find the complete set of typical testors for the matrix

$\theta^N(M_3)$ for each N. Also, Partition 2 turned out to be more efficient than Partition 1 in this particular case.

Operator φ applied to M_3

By applying this operator the number of rows and the density remain constant in the resulting matrices. The number of columns increases from 10 to 10. And the number of typical testors increases greatly for the dimensions of the resulting matrices.

Next is the complete table of results:

$\varphi^N(M_3)$	Rows	Columns	Density	TT	YYC	Partition 1
1	4	10	0,525	24	0,049014	0,060935
2	4	20	0,525	132	0,011605	0,187471
3	4	30	0,525	378	0,0283447	0,89801
4	4	40	0,525	816	0,048724	3,724487
5	4	50	0,525	1500	0,072615	11,831504

Table 3.16: Results for M_3 using the φ operator.

Now a comparative graph of the performance of the algorithms:

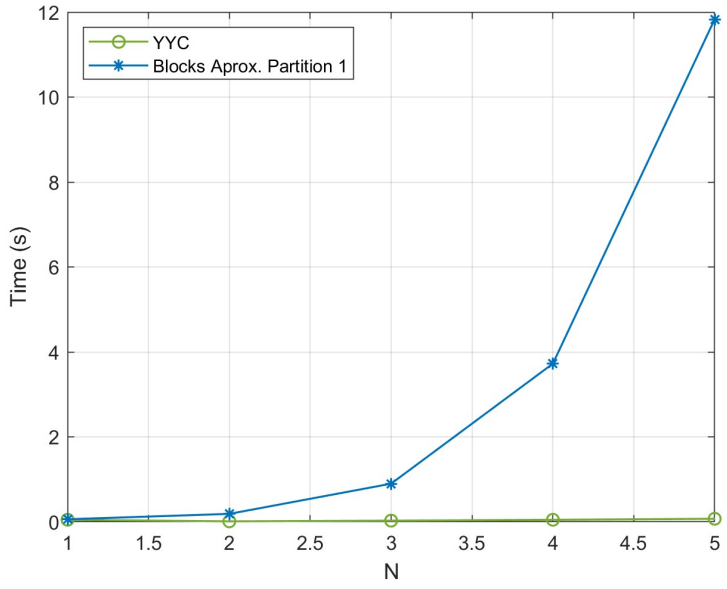

Figure 3.16: Results for M_3 using the φ operator.

The best algorithm to apply in this case is the YYC. Its execution time was always less than the time recorded for the Blocks Algorithm. This result can be explained by observing the table 5.25 in the Annexes Section 5, where it is evident that for each block there were a large number of typical testors. That is, there were a lot of joins to test. So, for this case it is better to search for the typical testors by row than to test unions of typical testors of the blocks. It should be noted that the Block Algorithm was able to find the complete set of testors typical of the $\varphi^N(M_3)$ matrices for each N.

3.2 Parallel Computing and Basic Synthetic Matrices

As mentioned in the previous section, the efficiency of the Blocks algorithm drops considerably when the number of unions of typical testors to be tested is very large. This drawback of the method could be combated by not using the full set of typical testors for the blocks but the parallelization process will not help the proposed method much in this case.

Thus, in order to properly analyze the efficiency of parallelizing the algorithm we will focus only in tests with basic synthetic matrices in which the Sequential Blocks Algorithm has turned out to be more efficient than the YYC Algorithm. This is because if the Blocks Algorithm in its first sequential phase turned out to be very inefficient compared to the YYC algorithm, parallelization is unlikely to become a significant improvement.

The basic matrices I_5, A, B, M_1 and M_2 will be used. As before, we will apply operators to these matrices. Also, by using some of the results of the previous section we can more completely compare the improvement of parallel computing.

Secondly, since there is no conclusive evidence that it is relevant (in terms of computational efficiency) to perform Partition 1 or Partition 2, for this stage of experimentation we will use only Partition 1.

We will now explain the meanings of the labels of the columns of the tables that contain the results for this section. The first column tells us the number of times the operator $\mathcal{O}$ was applied to the basic matrix M. Columns two, three and four are characteristic of the basic matrix $X = \mathcal{O}^N(M)$. Then TT column refers to the number of typical testors for the matrix X. The column labeled YYC records the time in seconds that it took to this algorithm to find all the typical testors of X. The column labeled *Sequential* tells us the total time (in seconds) the Blocks Algorithm took to execute sequentially. The column labeled *Parallel* records the total time measured in seconds that it took for the Blocks Algorithm to execute completely when parallelization is applied by using algorithm 4.

3.2.1 Basic Matrix I_5 Experiments Parallelization

Due to the favorable results for the Sequential Blocks Algorithm (3) detailed in the previous section for the matrix I_5, tests are performed using the Parallelized Blocks Algorithm (4) making use of two of the three operators defined in Section 2.5.1.

Operator γ applied to I_5

The γ-Operator was recursively applied to the matrix I_5, the complete basic matrix was divided into two blocks trying to have the same number of rows. Below is the table with the experimental results obtained in this case:

$\gamma^N(I_5)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	5	5	0.2000	1	0.00537	0.03130	0.06524
2	10	10	0.1000	1	0.00594	0.01917	0.06323
3	15	15	0.0667	1	0.00689	0.01330	0.05930
4	20	20	0.0500	1	0.00610	0.00715	0.04792
5	25	25	0.0400	1	0.01087	0.00987	0.05457
6	30	30	0.0333	1	0.01744	0.01027	0.05623
8	40	40	0.0250	1	0.03145	0.01510	0.06147
10	50	50	0.0200	1	0.03036	0.01560	0.04705

Table 3.17: Results parallelization for I_5 using the γ operator.

In this case, the parallelization of the algorithm turns out to be inefficient. This may be due to the fact that the dimension of the matrices obtained by applying the operator θ on the matrix I_5 is relatively small in dimension. Being small in size, it is not computationally efficient to distribute tasks to workers and to apply paralleling computation.

In other words, for the time it takes for the computer to distribute the tasks to its workers, it takes more time than carrying out the process sequentially.

This result may be an indicator that in the case of small-dimensional matrices, the parallelization process would not necessarily be more efficient. In fact, the Blocks Algorithm could be used sequentially and avoid the computer having to distribute tasks internally. As before, the complete set of typical testors was found.

These observations can be seen if we make a comparative graph of the efficiency of the algorithms based on table 3.17:

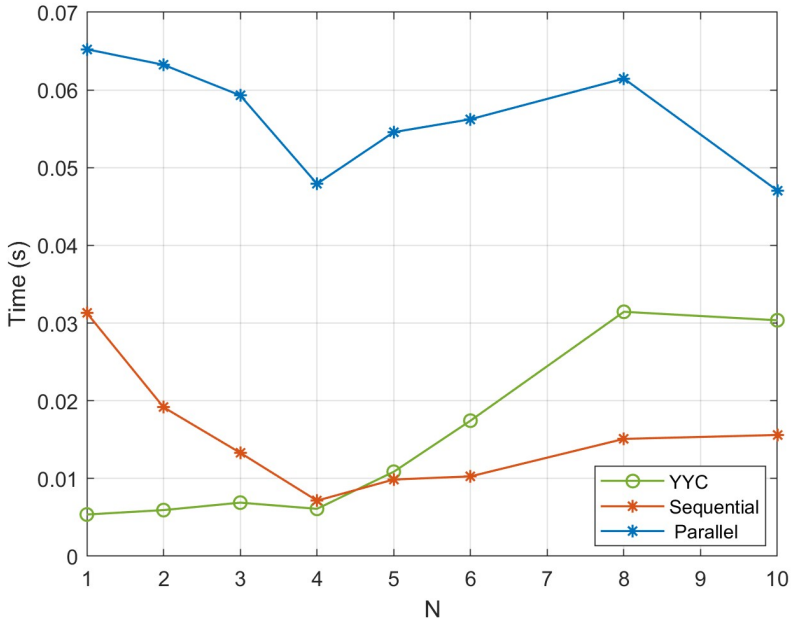

Figure 3.17: Parallelization Results for I_5 using the γ operator.

Operator θ applied to I_5

When we apply the θ-Operator recursively to the matrix I_5, the following experimental results are obtained:

$\theta^N(I_5)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	5	5	0.200000	1	0.003416	0.031299	0.089988
2	25	10	0.080000	2	0.059171	0.076950	0.113450
3	125	15	0.024000	3	0.055460	0.094993	0.116216
4	625	20	0.006400	4	0.288344	0.291037	0.239069
5	3125	25	0.001600	5	4.557188	3.835996	2.545781
6	15625	30	0.000384	6	425.640116	143.165474	52.509040

Table 3.18: Results parallelization for I_5 using the θ operator.

In this case, the parallelization of the algorithm turns out to be very successful. Of the three algorithms tested, the most efficient is the Parallelized Blocks Algorithm.

A graph obtained from the experimental results is shown in Figure 3.18. In this graph, it is evident that as N becomes larger, the efficiency of the parallelized algorithm

also increases. That is, the distribution of tasks within the multicores turns out to be efficient. This makes sense when looking at the dimension of the matrices on which the algorithm preformed better. As before, the complete set of typical testors was found.

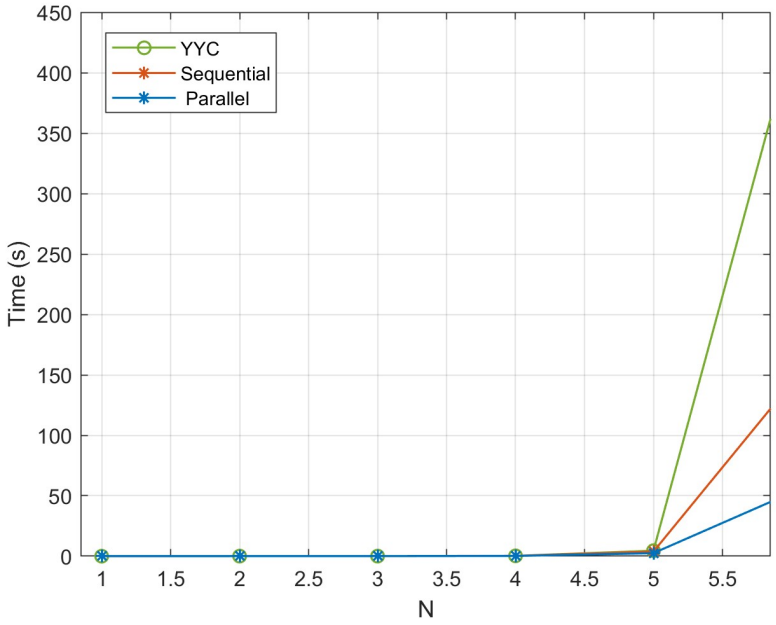

Figure 3.18: Parallelization Results for I_5 using the θ operator.

3.2.2 Basic Matrix A Experiment Parallelization

If the operator θ is applied to the basic matrix A recursively and the Blocks sequential algorithm, its parallelized version and the YYC algorithm are applied, the following execution times dependent on N are obtained:

$\theta^N(A)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	4	5	0.5500	5	0.02955	0.16035	0.20046
2	16	10	0.3438	10	0.03722	0.08056	0.12627
3	64	15	0.1289	15	0.04003	0.06671	0.12347
4	256	20	0.0430	20	0.16886	0.22805	0.18971
5	1024	25	0.0134	25	1.98210	1.49374	0.99190
6	4096	30	0.0040	30	35.94585	22.66953	14.12854
7	16384	35	0.0012	35	762.99113	445.54098	276.71959

Table 3.19: Results parallelization for A using the θ operator.

Next, a comparative graph of the performance of the algorithms:

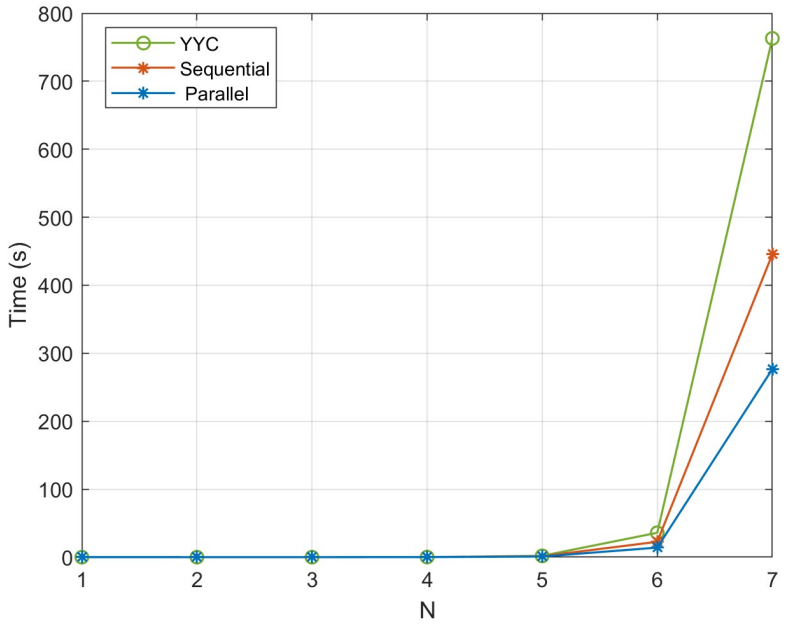

Figure 3.19: Parallelization Results for A using the θ operator.

As evidenced in table 3.19 and in graph 3.19, the best performance for this family of matrices is carried out by the Parallelized Blocks algorithm. It is efficient that the tasks are distributed to the multicores that the computer has and this improves the efficiency of the sequential algorithm.

3.2.3 Basic Matrix B Experiment Parallelization

Next the experimental results obtained for B matrix when θ operator is applied:

$\theta^N(B)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	4	5	0.5000	5	0.004750	0.030756	0.078621
2	16	10	0.3125	10	0.008634	0.033356	0.070697
3	64	15	0.1172	15	0.066651	0.092834	0.108287
4	256	20	0.0391	20	0.552101	0.391459	0.293576
5	1024	25	0.0122	25	11.563568	5.175375	3.289233
6	4096	30	0.0037	30	303.740156	121.208505	77.532677

Table 3.20: Results parallelization for B using the θ operator.

The following is a comparative graph of the performance of the algorithms:

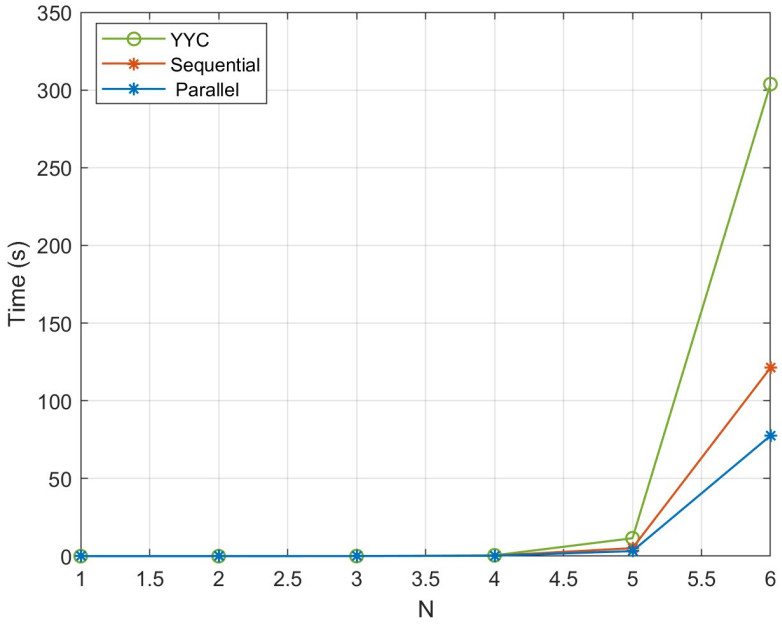

Figure 3.20: Parallelization Results for B using the θ operator.

The algorithm that manages to obtain the complete set of typical testors for this family of basic matrices faster is that of Parallelized Blocks. In other words, it is worth doing the parallelization process as it does increase the efficiency of the sequential method discussed in the previous section.

3.2.4 Basic Matrix M_1 Experiment Parallelization

Below are the results obtained for M_1:

$\theta^N(M_1)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	16	10	0.3281	10	0.093533	0.242545	0.281115
2	256	20	0.041	20	0.29709	0.34286	0.266416
3	4096	30	0.0038	30	100.458358	62.073514	38.160306

Table 3.21: Results parallelization for M_1 using the θ operator.

Due to the large amount that is obtained, even when applying the θ operator to

this basic matrix a few times, the process of parallelizing the Blocks algorithm greatly increases the efficiency of the method. This can be visually evidenced in the following graph:

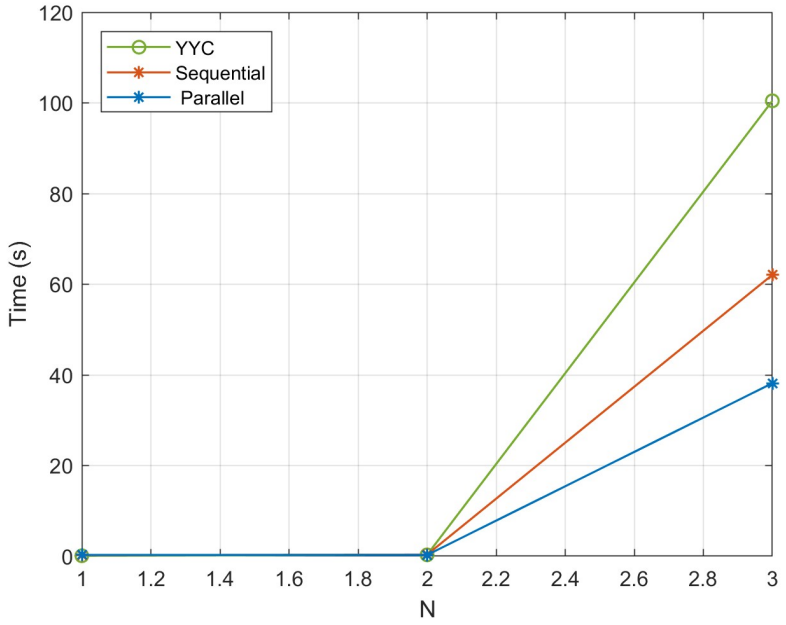

Figure 3.21: Parallelization Results for M_1 using the θ operator.

3.2.5 Basic Matrix M_3 Experiment Parallelization

Following is the table with the results obtained:

$\theta^N(M_3)$	Rows	Columns	Density	TT	YYC	Sequential	Parallel
1	4	10	0.525	24	0.049014	0.060935	0.094418
2	16	20	0.525	48	0.08538	0.137144	0.162983
3	64	30	0.2461	72	0.41647	0.45491	0.369814
4	256	40	0.082	96	14.806416	6.03836	3.80871
5	1024	50	0.0256	120	806.866905	265.149842	151.159575

Table 3.22: Results parallelization for M_3 using the θ operator.

In this case, implementing parallelization in the Blocks Algorithm turns out to be very helpful. The method greatly increases efficiency, which means that the distribution of tasks in multicores does present an improvement.

A graph is presented to observe the efficiency curves of the tested algorithms:

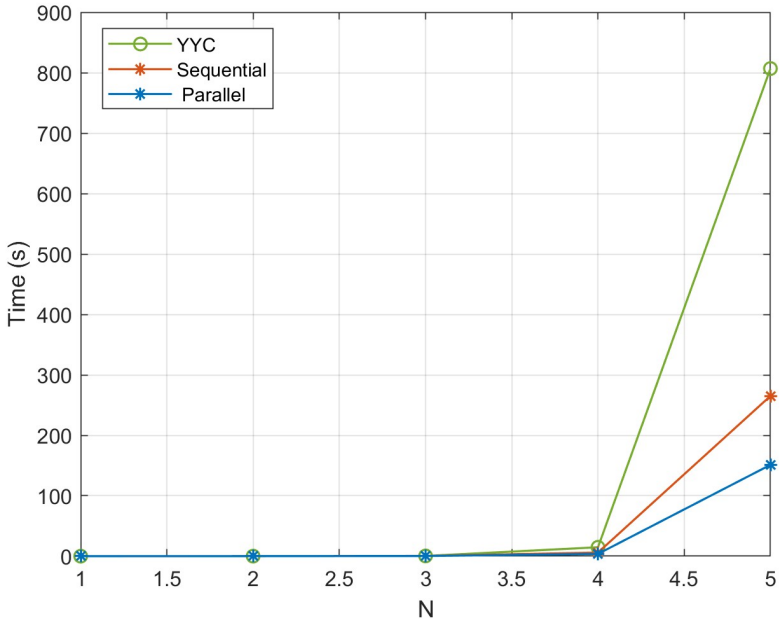

Figure 3.22: Parallelization Results for M_3 using the θ operator.

3.3 Application to QSAR biodegradation Data Set

First, the classification neural network was designed using the full number of base features. Of the 1055 instances, 500 were used for the training process and 555 for testing. Because the base is not balanced, 250 random items from each class were taken for the training process. The remaining elements of each class were part of the test set. The basic matrix obtained in this case had a dimension of 706×41.

In the training stage, the Aaccuracy of the neural netword was $90,00\%$. What turns out to be acceptable considering that the base is not equitable in the number of elements per class. For the test stage, the neural network had 85.77% accuracy. Below is the confusion matrix obtained for the neural network using all the features:

		Predicted	
		RB	NRB
Actual	RB	87,74%	12,26%
	NRB	14,70%	85,30%

Table 3.23: Confusion Matrix for the model trained with all attributes

The percentage of elements of the RB class and NRB class that are misclassified as NRB are due to the fact that the base does not have an equal distribution of elements of classes. As they are 356 ready biodegradable molecules and 699 non-ready. But this is a risk of working with real bases, that a priori we don't know if it will be possible to obtain an equitable percentage of training for each class.

This preamble helps us to analyze the database and the neural network. Now the proposed method and the results obtained for classification using only selected features are applied.

Below is a table that records the ejecution time (measured in seconds) to find the typical testors of the basic matrix for the biodegradable base QSAR [23]:

Algorithm	Total Execution Time
YYC	18.8944
Sequential Blocks Algorithm	10.6227
Parallel Blocks Algorithm	7.9238

Table 3.24: Executions times

Once the typical testors for this base were obtained, the same neural network used aboved was trained using only the characteristics belonging to one single testor of length six. We preserved the structure of the neural network that allowed us to obtain the table 3.3.

Out of a total of 41 characteristics, the new classification model only needs 6 to be able to predict. In other words, we reduce by approximately 85% the number of characteristics necessary to know if a molecule is ready or not ready biodegradable.

The characteristics selected for the neural network are the following:

- Number of heavy atoms
- Frequency of C-N at topological distance 3

- Frequency of C-O at topological distance 3
- Number of CRX3
- Leading eigenvalue from adjacency matrix (Lovasz-Pelikan index)
- Second Mohar index from Laplace matrix

It is worth mentioning that to obtain the detailed list of all the characteristics, you can go to the reference [23].

In general, the new model for the training set obtains an accuracy of 84,20%. For the test set, we obtained 81,57% of accuracy.

Below is the confusion matrix obtained after training the neural network only with the selected features using Testor Theory:

		Predicted	
		RB	NRB
Actual	RB	88,68%	11,32%
	NRB	19,60%	80,40%

Table 3.25: Confusion Matrix for the model trained with only selected features

Although in this case we are still affected by the inequality of elements by class of the base, the percentage of well classified for RB increases. This is favorable since it is the class of which there are fewer members. On the other hand, the number of well classified if an element belongs to the NRB class has been reduced about 5%. This is not entirely inefficient considering that we only used 6 of 41 variables.

Chapter 4

CONCLUSIONS AND FUTURE WORK

Regarding the performance of the proposed algorithm on the set of basic synthetic test matrices, it can be concluded that it is not appropriate to apply it in all cases. The success cases turn out to be when the matrix has a large number of rows, or is generally large in dimension, but has relatively few typical testors or the goal is not to find the complete set of irreducible testors. Regarding the parallelization of the proposed method, it can be concluded that it is successful when the dimension of the matrix is large.

Regarding the application on a real basis, we conclude that the method allows to perform acceptable classifications using neural networks and greatly reducing the number of attributes needed to classify. This is an advantage of great importance when working with massive databases with many various and looking for quick results in the short term for classification.

One of the advantages of the method proposed but yet to be explore is that when using the Blocks algorithm in real databases as an application form, the number of typical testors that to find can be limited. In real application problems, it is not necessary to find the complete set of typical testors. For example, in classification problems it is enough to find one typical testor that efficiently classifies the classes and thus a neural network can be trained more efficiently without using the full set of features.

On the other hand, it is possible to improve much more the performance of the proposed algorithm. In future works, the idea would be to try defining more than two blocks for large arrays. And then test the unions. In this case applying parallelization could be even more profitable for the method.

Also, the set of typical testors could for each block could also be limited. In other words, not to use the complete set but only a part of it, taking into account that to train a neural network by trial and error it is not necessary to obtain the complete set. This could reduce the time it takes for the Blocks Algorithm to run as it would have fewer unions of typical testors to test.

Another possible improvement would be the fact of parallelizing more the algorithm. By this I mean to parallelized the YYC as mentioned in [29].

Also as part of future work, it would be interesting to use other types of algorithms to find typical testors in the Blocks. Depending on the characteristics of the basic matrices, the most efficient algorithm for that case could be chosen and used in Step 1 of the Blocks Algorithm. The rest of the method process would remain the same. With this, we want to note that this proposed method can be used with algorithms other than YYC and could be beneficial in computational terms.

Chapter 5

ANNEXES

5.1 Full Tables of Results for the Set of Synthetic Test Matrices

In this section we present the complete tables of results obtained during the experimentation part of the Blocks Algorithm for all the basic matrices defined in this work.

The number of rows of each block is shown, the time that the YYC algorithm took to obtain all the testors for each block, the time that the Blocks algorithm took to execute to obtain the typical testors of the basic matrix, the total time of this process and the percentage of typical testors obtained that coincide with those of the complete basic matrix obtained previously. Time is measured in seconds.

5.1.1 Complete Results for I_5

We present the complete tables of results obtained for the matrix I_5.

Tables of Results of I_5 with Partition 1

Below is the table with the complete results for matrix I_5 using Partition 1 and the Blocks algorithm for the γ operator.

$\gamma^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	1	0.004228	0.025588	0.031299	100%
	B2	3	1	0.001483			
2	B1	5	1	0.002940	0.015140	0.019170	100%
	B2	5	1	0.001090			
3	B1	8	1	0.002422	0.009060	0.013305	100%
	B2	7	1	0.001823			
4	B1	10	1	0.001936	0.002708	0.007149	100%
	B2	10	1	0.002505			
5	B1	13	1	0.003732	0.001409	0.009869	100%
	B2	12	1	0.004728			
6	B1	15	1	0.003496	0.003586	0.010271	100%
	B2	15	1	0.003189			
8	B1	20	1	0.005202	0.005900	0.015096	100%
	B2	20	1	0.003994			
10	B1	25	1	0.007096	0.004726	0.015599	100%
	B2	25	1	0.003777			

Table 5.1: Complete results for I_5 using the γ operator.

Now the table with the complete results for matrix I_5 using the Partition 1 and the Blocks algorithm for the θ is presented.

$\theta^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	1	0,004228	0,025588	0,031299	100%
	B2	3	1	0,001483			
2	B1	12	3	0,013435	0,052456	0,07695	100%
	B2	13	3	0,011059			
3	B1	62	5	0,025168	0,046309	0,094993	100%
	B2	63	5	0,023516			
4	B1	312	7	0,094012	0,046236	0,291037	100%
	B2	313	7	0,150789			
5	B1	1563	9	1,382594	0,248859	3,835996	100%
	B2	1562	9	2,204543			
6	B1	7813	11	84,24754	6,133656	143,165474	100%
	B2	7812	11	52,784278			

Table 5.2: Complete results for I_5 using the θ operator.

Following, we present the table with the complete results for matrix I_5 using Partition 1 and the Blocks algorithm for the φ operator.

$\varphi^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	1	0,004228	0,025588	0,031299	100%
	B2	3	1	0,001483			
2	B1	2	4	0,00262	0,023827	0,034755	100%
	B2	3	8	0,008308			
3	B1	2	9	0,001376	0,325295	0,330449	100%
	B2	3	27	0,003778			
4	B1	2	16	0,001733	4,158029	4,168582	100%
	B2	3	64	0,00882			
5	B1	2	25	0,002016	37,320405	37,331452	100%
	B2	3	125	0,009031			
6	B1	2	36	0,004124	232,970714	232,989829	100%
	B2	3	216	0,014991			
8	B1	2	64	0,00428	14400<	-	-
	B2	3	512	0,030842			
10	B1	2	100	0,012028	36000<	-	-
	B2	3	1000	0,045941			

Table 5.3: Complete results for I_5 using the φ operator.

Tables with Results of I_5 with Partition 2

Below is the table with the complete results for matrix I_5 using Partition 2 and the Blocks algorithm for the γ operator.

$\gamma^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	1	0,004228	0,025588	0,031299	100%
	B2	3	1	0,001483			
2	B1	3	1	0,001305	0,021207	0,024763	100%
	B2	7	1	0,002251			
3	B1	4	1	0,001176	0,006567	0,01042	100%
	B2	11	1	0,002677			
4	B1	5	1	0,000965	0,002408	0,006432	100%
	B2	15	1	0,003059			
5	B1	6	1	0,001982	0,00176	0,008085	100%
	B2	19	1	0,004343			
6	B1	8	1	0,005526	0,001636286	0,013504286	100%
	B2	22	1	0,006342			
8	B1	10	1	0,002052	0,002739	0,009752	100%
	B2	30	1	0,004961			
10	B1	12	1	0,002241	0,004696	0,011771	100%
	B2	38	1	0,004834			

Table 5.4: Complete results for I_5 using the operator γ.

Now the table with the complete results for matrix I_5 using the Partition 2 and the Blocks algorithm for the θ is presented.

$\theta^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	1	0,004228	0,025588	0,031299	100%
	B2	3	1	0,001483			
2	B1	6	3	0,001825	0,014337	0,025822	100%
	B2	19	3	0,00966			
3	B1	32	5	0,006821	0,017784	0,071902	100%
	B2	93	5	0,047297			
4	B1	156	7	0,056168	0,044925	0,329386	100%
	B2	469	7	0,228293			
5	B1	782	9	0,441939	0,273044	4,153209	100%
	B2	2343	9	3,438226			
6	B1	3906	11	15,520486	7,742165	166,028161	100%
	B2	11719	11	142,76551			

Table 5.5: Complete results for I_5 using the θ operator.

5.1.2 Complete Results for A

We present the complete tables of results obtained for the matrix A.

Tables of Results of A with Partition 1

Below is the table of the complete results for matrix A using Partition 1 and the Blocks algorithm for the θ operator.

$\theta^N(A)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	3	0,004093	0,152553	0,160351	100%
	B2	2	3	0,003705			
2	B1	8	8	0,006703	0,06968	0,080564	100%
	B2	8	8	0,004181			
3	B1	32	13	0,02154	0,035069	0,066707	100%
	B2	32	13	0,010098			
4	B1	128	18	0,091338	0,066581	0,228052	100%
	B2	128	18	0,070133			
5	B1	512	23	0,824677	0,155648	1,493741	100%
	B2	512	23	0,513416			
6	B1	2048	28	13,4796	0,65395	22,669527	100%
	B2	2048	28	8,535977			
7	B1	8192	33	274,900442	6,248284	445,540981	100%
	B2	8192	33	164,392255			

Table 5.6: Complete results for A using the θ operator.

Following, we present the table with the complete results for matrix A using Partition 1 and the Blocks algorithm for the φ operator.

$\varphi^N(A)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	3	0,004093	0,152553	0,160351	100%
	B2	2	3	0,003705			
2	B1	2	10	0,001239	0,038489	0,041484	100%
	B2	2	8	0,001756			
3	B1	2	21	0,001994	0,073963	0,0782	100%
	B2	2	15	0,002243			
4	B1	2	36	0,004188	0,164785	0,17203	100%
	B2	2	24	0,003057			
5	B1	2	55	0,005292	0,40576	0,413693	100%
	B2	2	35	0,002641			
6	B1	2	78	0,005193	0,728804	0,736256	100%
	B2	2	48	0,002259			
8	B1	2	136	0,010563	10800<	-	-
	B2	2	80	0,00514			
10	B1	2	210	0,013255	28800<	-	-
	B2	2	120	0,008555			

Table 5.7: Complete results for A using the φ operator.

Table of Results of A with Partition 2

Below is the table with the complete results for matrix A using Partition 1 and the Blocks algorithm for the θ operator.

$\theta^N(A)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	3	0,004093	0,152553	0,160351	100%
	B2	2	3	0,003705			
2	B1	4	8	0,004893	0,032878	0,045398	100%
	B2	12	8	0,007627			
3	B1	16	13	0,008998	0,038179	0,070502	100%
	B2	48	13	0,023325			
4	B1	64	18	0,042162	0,06365	0,222462	100%
	B2	192	18	0,11665			
5	B1	256	23	0,203067	0,13454	1,391179	100%
	B2	768	23	1,053572			
6	B1	1024	28	2,30793	0,715668	21,262211	100%
	B2	3072	28	18,238613			
7	B1	4096	33	41,602925	6,950251	419,779761	100%
	B2	12288	33	371,226585			

Table 5.8: Complete results for A using the θ operator.

5.1.3 Complete Results for B

Tables of Results of B with Partition 1

Below is the table of the complete results for matrix B using Partition 1 and the Blocks algorithm for the θ operator.

$\theta^N(B)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	5	0,002954	0,025685	0,030756	100%
	B2	2	2	0,002117			
2	B1	8	10	0,006173	0,023826	0,033356	100%
	B2	8	7	0,003357			
3	B1	32	15	0,029414	0,041003	0,092834	100%
	B2	32	12	0,022417			
4	B1	128	20	0,203483	0,06655	0,391459	100%
	B2	128	17	0,121426			
5	B1	512	25	3,218218	0,157563	5,175375	100%
	B2	512	22	1,799594			
6	B1	2048	30	77,883362	0,828213	121,208505	100%
	B2	2048	27	42,49693			

Table 5.9: Complete results for B using the θ operator.

Following is the table with the complete results and the Blocks algorithm for the γ operator.

$\gamma^N(B)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	5	0,002954	0,025685	0,030756	100%
	B2	2	2	0,002117			
2	B1	4	5	0,001681	0,023282	0,026731	100%
	B2	4	5	0,001768			
3	B1	6	25	0,006473	0,120388	0,12944	100%
	B2	6	10	0,002579			
4	B1	8	25	0,008278	1,550225	1,566974	100%
	B2	8	25	0,008471			
5	B1	10	125	0,018182	44,087545	44,119327	100%
	B2	10	50	0,0136			
6	B1	12	125	0,05613	21600<	-	-
	B2	12	125	0,035807			

Table 5.10: Complete results for B using the γ operator.

Tables of Results of B with Partition 2

Below is the table with the complete results for matrix B using Partition 1 and the Blocks algorithm for the θ operator.

$\theta^N(B)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	5	0,002954	0,025685	0,030756	100%
	B2	2	2	0,002117			
2	B1	4	8	0,001688	0,02976	0,040685	100%
	B2	12	9	0,009237			
3	B1	16	13	0,011469	0,040069	0,092404	100%
	B2	48	14	0,040866			
4	B1	64	18	0,067944	0,064226	0,399752	100%
	B2	192	19	0,267582			
5	B1	256	23	0,632752	0,167011	5,608347	100%
	B2	768	24	4,808584			
6	B1	1024	28	13,07284	0,911034	135,161253	100%
	B2	3072	29	121,177379			

Table 5.11: Complete results for B using the θ operator.

Following is the table with the complete results for matrix B using Partition 2 and the Blocks algorithm for the γ operator.

$\gamma^N(B)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	5	0,002954	0,025685	0,030756	100%
	B2	2	2	0,002117			
2	B1	2	5	0,001882	0,016967	0,022235	100%
	B2	6	10	0,003386			
3	B1	3	5	0,001442	0,121124	0,132935	100%
	B2	9	50	0,010369			
4	B1	4	5	0,001672	1,542139	1,574165	100%
	B2	12	125	0,030354			
5	B1	5	15	0,002105	43,989237	44,082993	100%
	B2	15	500	0,091651			
6	B1	6	25	0,003593	21600<	-	-
	B2	18	1250	0,19521			

Table 5.12: Complete results for B using the γ operator.

5.1.4 Complete Results for M_1

We present the complete tables of results obtained for the matrix M_1.

Tables with Results of M_1 with Partition 1

Below is the table with the complete results for matrix M_1 using Partition 1 and the Blocks algorithm for the γ operator.

$\gamma^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	8	8	0,009885	0,225385	0,242545	100%
	B2	8	8	0,007275			
2	B1	16	10	0,021829	0,106078	0,149861	100%
	B2	16	10	0,021954			
3	B1	24	80	0,065116	4,746237	4,880646	100%
	B2	24	80	0,069293			
4	B1	32	100	0,073914	338,340506	338,500018	100%
	B2	32	100	0,085598			
5	B1	40	800	0,506273	18000<	-	-
	B2	40	800	0,592656			

Table 5.13: Complete results for M_1 using the γ operator.

Now the table with the complete results for matrix M_1 using the Partition 1 and the Blocks algorithm for the θ is presented.

$\theta^N(M_1)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	8	8	0,009885	0,225385	0,242545	100%
	B2	8	8	0,007275			
2	B1	128	18	0,152795	0,079967	0,34286	100%
	B2	128	18	0,110098			
3	B1	2048	28	38,064186	0,75932	62,073514	100%
	B2	2048	28	23,250008			

Table 5.14: Complete results for M_1 using the θ operator.

Following, we present the table with the complete results for matrix M_1 using Partition 1 and the Blocks algorithm for the φ operator.

$\varphi^N(M_1)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	8	8	0,009885	0,225385	0,242545	100%
	B2	8	8	0,007275			
2	B1	8	38	0,015327	0,111082	0,136302	100%
	B2	8	36	0,009893			
3	B1	8	102	0,041917	0,526178	0,598597	100%
	B2	8	96	0,030502			
4	B1	8	212	0,067842	1,99583	2,12069	100%
	B2	8	200	0,057018			
5	B1	8	380	0,102456	6,179356	6,354836	100%
	B2	8	360	0,073024			
6	B1	8	618	0,182065	16,227804	16,539908	100%
	B2	8	588	0,130039			

Table 5.15: Complete results for M_1 using the φ operator.

Tables with Results of M_1 with Partition 2

Below is the table with the complete results for matrix M_1 using Partition 2 and the Blocks algorithm for the γ operator.

$\gamma^N(I_5)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	8	8	0,009885	0,225385	0,242545	100%
	B2	8	8	0,007275			
2	B1	8	8	0,002799	0,132712	0,199808	100%
	B2	24	80	0,064297			
3	B1	12	10	0,005406	4,718812	5,24383	100%
	B2	36	800	0,519612			
4	B1	16	10	0,0095	384,324964	385,16331	100%
	B2	48	1000	0,828846			
5	B1	20	80	0,035205	18000<	-	-
	B2	60	8000	8,403873			

Table 5.16: Complete results for M_1 using the γ operator.

Now the table with the complete results for matrix M_1 using the Partition 2 and the Blocks algorithm for the θ is presented.

$\theta^N(M_1)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	8	8	0,009885	0,225385	0,242545	100%
	B2	8	8	0,007275			
2	B1	64	18	0,046841	0,076909	0,287528	100%
	B2	192	18	0,163778			
3	B1	1024	28	6,68468	0,782022	58,046076	100%
	B2	3072	28	50,579374			

Table 5.17: Complete results for M_1 using the θ operator.

5.1.5 Complete Results for M_2

We present the complete tables of results obtained for the matrix M_2.

Tables with Results of M_2 with Partition 1

Below is the table with the complete results for matrix M_2 using Partition 1 and the Blocks algorithm for the γ operator.

$\gamma^N(M_2)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	4	5	0,00195	0,010342	0,015825	100%
	B2	4	5	0,003533			
2	B1	8	25	0,029793	1,677296	1,717367	100%
	B2	8	25	0,010278			
3	B1	12	125	0,028583	938,615053	938,675906	100%
	B2	12	125	0,03227			
4	B1	16	625	0,086856	25200<	-	-
	B2	16	625	0,106558			

Table 5.18: Complete results for M_2 using the γ operator.

Now the table with the complete results for matrix M_2 using the Partition 1 and the Blocks algorithm for the θ is presented.

$\theta^N(M_2)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	4	5	0,00195	0,010342	0,015825	100%
	B2	4	5	0,003533			
2	B1	32	30	0,008957	1,591246	1,60673	100%
	B2	32	30	0,006527			
3	B1	256	55	0,029354	949,01814	949,077128	100%
	B2	256	55	0,029634			

Table 5.19: Complete results for M_2 using the θ operator.

Following, we present the table with the complete results for matrix M_2 using Partition 1 and the Blocks algorithm for the φ operator.

$\varphi^N(M_2)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	4	5	0,00195	0,010342	0,015825	100%
	B2	4	5	0,003533			
2	B1	4	24	0,00328	1,809769	1,817258	100%
	B2	4	28	0,004209			
3	B1	4	63	0,005971	100,809195	100,825489	100%
	B2	4	81	0,010323			
4	B1	4	128	0,011858	1974,770176	1974,798767	100%
	B2	4	176	0,016733			

Table 5.20: Complete results for M_2 using the φ operator.

Tables with Results of M_2 with Partition 2

Below is the table with the complete results for matrix M_2 using Partition 2 and the Blocks algorithm for the γ operator.

$\gamma^N(M_2)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	4	5	0,00195	0,010342	0,015825	100%
	B2	4	5	0,003533			
2	B1	4	5	0,002277	1,587369	1,624205	100%
	B2	12	125	0,034559			
3	B1	6	25	0,004047	1165,424106	1165,595216	100%
	B2	18	1250	0,167063			
4	B1	8	25	0,003025	25200<	-	-
	B2	24	15625	2,165193			

Table 5.21: Complete results for M_2 using the γ operator.

Now the table with the complete results for matrix M_2 using the Partition 2 and the Blocks algorithm for the θ is presented.

$\theta^N(M_2)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	4	5	0,00195	0,010342	0,015825	100%
	B2	4	5	0,003533			
2	B1	16	28	0,001405	1,552421	1,585995	100%
	B2	48	40	0,032169			
3	B1	128	53	0,004316	1174,698747	1174,874773	100%
	B2	384	65	0,17171			

Table 5.22: Complete results for M_2 using the θ operator.

5.1.6 Complete Results for M_3

We present the complete tables of results obtained for the matrix M_3.

Tables with Results of M_3 with Partition 1

Below is the table with the complete results for matrix M_3 using Partition 1 and the Blocks algorithm for the γ operator.

$\gamma^N(M_3)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	14	0,003765	0,051675	0,060935	100%
	B2	2	7	0,005495			
2	B1	4	24	0,003326	1,352554	1,358063	100%
	B2	4	24	0,002183			
3	B1	6	336	0,031085	863,711086	863,766025	100%
	B2	6	168	0,023854			
4	B1	8	576	0,063481	21600<	-	-
	B2	8	576	0,065862			

Table 5.23: Complete results for M_3 using the γ operator.

Now the table with the complete results for matrix M_3 using the Partition 1 and the Blocks algorithm for the θ is presented.

$\theta^N(M_3)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	14	0,003765	0,051675	0,060935	100%
	B2	2	7	0,005495			
2	B1	8	38	0,018294	0,10857	0,137144	100%
	B2	8	31	0,01028			
3	B1	32	62	0,13427	0,221068	0,45491	100%
	B2	32	55	0,099572			
4	B1	128	86	3,529959	0,506778	6,03836	100%
	B2	128	79	2,001623			
5	B1	512	110	170,282129	1,727866	265,149842	100%
	B2	512	103	93,139847			

Table 5.24: Complete results for M_3 using the θ operator.

Following, we present the table with the complete results for matrix M_3 using Partition 1 and the Blocks algorithm for the φ operator.

$\varphi^N(M_3)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	14	0,003765	0,051675	0,060935	100%
	B2	2	7	0,005495			
2	B1	2	52	0,00493	0,178556	0,187471	100%
	B2	2	22	0,003985			
3	B1	2	114	0,006755	0,887681	0,89801	100%
	B2	2	45	0,003574			
4	B1	2	200	0,008401	3,711539	3,724487	100%
	B2	2	76	0,004547			
5	B1	2	310	0,014673	11,809716	11,831504	100%
	B2	2	115	0,007115			

Table 5.25: Complete results for M_3 using the φ operator.

Tables with Results of M_3 with Partition 2

Below is the table with the complete results for matrix M_3 using Partition 2 and the Blocks algorithm for the γ operator.

$\gamma^N(M_3)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	14	0,003765	0,051675	0,060935	100%
	B2	2	7	0,005495			
2	B1	2	14	0,001872	1,67779	1,701933	100%
	B2	6	168	0,022271			
3	B1	3	21	0,002914	848,469166	848,675431	100%
	B2	9	2880	0,203351			
4	B1	4	24	0,003894	21600<	-	-
	B2	12	13824	0,968085			

Table 5.26: Complete results for M_3 using the γ operator.

Now the table with the complete results for matrix M_3 using the Partition 2 and the Blocks algorithm for the θ is presented.

$\theta^N(M_3)$		Rows	TT	YYC	Blocks	Total Time	Comparisons
1	B1	2	14	0,003765	0,051675	0,060935	100%
	B2	2	7	0,005495			
2	B1	4	30	0,00506	0,104147	0,127581	100%
	B2	12	38	0,018374			
3	B1	16	54	0,044593	0,189301	0,438584	100%
	B2	48	62	0,20469			
4	B1	64	78	0,488683	0,515952	6,984725	100%
	B2	192	86	5,98009			
5	B1	256	102	18,141178	1,720689	326,694497	100%
	B2	768	110	306,83263			

Table 5.27: Complete results for M_3 using the θ operator.

Bibliography

[1] Vázquez R.,Godoy S. 2007. *Using testor theory to reduce the dimension of neural network models.* Special Issue in Neural Networks and Associative Memories. 28, 93-103.

[2] Cheguis I.A., Yablonskii S.V. 1955. *About testors for electrical outlines.* Usp. Mat. Nauk,4(66): 182-184.

[3] Cheguis I.A., Yablonskii S.V. *Logical methods for controlling electrical systems.* Trudy MIAN ime V.A. Steklova, LI :270-360, 1958.

[4] Dimitriev A. N., Zhuravlev Y.I. and Krendeliev F.P. 1966. *About mathematical principles of objects and phenomena classification.* Diskretni Analiz 7.

[5] Martínez J. F., Santos J. A., Carrasco A. 2004. *Feature Selection using Typical Testors applied to Estimation of Stellar Parameters.* Computación y Sistemas [en linea], 8(1), 15-23. ISSN: 1405-5546. Disponible en: `https://www.redalyc.org/articulo.oa?id=61580103`

[6] Alba, E., & Santana, R. 2010. *Generación de matrices para evaluar el desempeño de estrategias de búsqueda de testores típicos.* ACI Avances En Ciencias E Ingenierías, 2(2). `https://doi.org/10.18272/aci.v2i2.23`

[7] Ouvrard, X. 2020. *Hypergraphs: An introduction and review.* arXiv.org. Retrieved June 1, 2022, from `https://arxiv.org/abs/2002.05014`

[8] Berge C. 1970. *Graphes et hypergraphes.* Dunod, Paris.

[9] Berge C. 1973. *Graphs and hypergraphs.* North-Holland publishing company Amsterdam.

[10] Alba E., Godoy S., Lazo M., Trinidad F. & Carrasco J. 2019. *On the Relation Between the Concepts of Irreducible Testor and Minimal Transversal.* IEEE Access, 7:82809– 82816.

[11] Guevara I. J. 2021. *Implementación Computacional del Algoritmo LEX para el cálculo de testores típicos usando aprendizaje simbólico* [Tesis previa a la obtención del título de Ingeniera Matemática]. Escuela Politécnica Nacional.

[12] Kavvadias D. J., Stavropoulos C. E. 2005. *An Efficient Algorithm for the Transversal Hypergraph Generation.* Journal of Graph Algorithms Appl. Vol 9, 239-264.

[13] Berge C. 1989. *Hypergraphs: Combinatorics of Finite Sets.* vol. 45. Amsterdam, The Netherlands: Elsevier.

[14] Khachiyan L., Boros E., Elbassioni K., Gurvich V. 2005. *A new algorithm for the hypergraph transversal problem.* Computing and Combinatorics. Berlin, Germany: Springer, pp. 767–776.

[15] Lias-Rodríguez A., Pons-Porrata A. 2009. *BR: A new method for computing all typical testors.* Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (Lecture Notes in Computer Science), vol. 5856. New York, NY, USA: Springer, pp. 433–440.

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

[17] Alba E., Ibarra J., Godoy S., Cervantes F. 2014. *YYC: A fast performance incremental algorithm for finding typical testors.* Iberoamerican Congress on Pattern Recognition, pp. 416–423. Springer.

[18] Santiesteban-Algaza, Y., Pons-Porrata, A. *LEX: A New Algorithm for the Calculus of all Typical Testors.* Vol. 1, pp. 85–95.

[19] Alba E., Ibarra J., Godoy S. 2016. *Generating synthetic test matrices as a benchmark for the computational behavior of typical testor-finding algorithms.* Pattern Recognition Letters, 80:46–51.

[20] Eiter T., Gottlob G. 2002. *Hypergraph Transversal Computation and Related Problems in Logic and AI.* European Workshop on Logics in Artificial Intelligence, 549–564, Springer.

[21] Bache, K., Lichman, M. 2013. *UCI machine learning repository.* `https://archive.ics.uci.edu/ml/index.php`

[22] Dua, D. Graff, C. 2019. *UCI Machine Learning Repository.* Irvine, CA: University of California, School of Information and Computer Science. `[http://archive.ics.uci.edu/ml`

[23] Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., Consonni, V. 2013. *Quantitative Structure - Activity Relationship models for ready biodegradability of chemicals.* Journal of Chemical Information and Modeling, 53, 867-878. `https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation`

[24] Gallegos, A., Torres, D., Álvarez, F., Torres, A. 2017. *Identificación de Características de Células de Cáncer de Mama por medio de testores típicos.* Research in Computing Science, 140(1), 43–54. https://doi.org/10.13053/rcs-140-1-4

[25] Lazo-Cortes, M., Ruiz-Shulcloper, J., Alba-Cabrera, E. 2001. *An overview of the evolution of the concept of testor.* Pattern Recognition, 34(4), 753–762. https://doi.org/10.1016/s0031-3203(00)00028-5

[26] G. Sánchez, M. Lazo, O. Fuentes. 1999. *Genetic algorithm to calculate minimal typical testors.* Proceedings of the IV Iberoamerican Symposium on Pattern Recognition. 207-214.

[27] González-Guevara V., Godoy-Calderon S., E. Alba-Cabrera E., Calvo H. 2019. *Symbolic Learning for Improving the Performance of Transversal-Computation Algorithms.* IEEE Access, vol. 7, pp. 19752-19761, doi: 10.1109/ACCESS.2019.2895296.

[28] Sanchez-Diaz G., Lazo-Cortes M.S., Aguirre-Salado C.A. et al. 2022. *A review of algorithms to computing irreducible testors applied to feature selection.* Artif Intell Rev.

[29] Piza-Davila I., Sanchez-Diaz G., Lazo-Cortes M. S., Noyola-Medrano, C. 2017. *Enhancing the performance of YYC algorithm useful to generate irreducible Testors.* International Journal of Pattern Recognition and Artificial Intelligence, 32(01), 1860001. https://doi.org/10.1142/s0218001418600017

[30] Rauber, T., Rünger, G. 2013. *Parallel programming* (pp. 169-226). Berlin, Germany:: Springer.

[31] Agarwal, R. C., Balle, S. M., Gustavson, F. G., Joshi, M., Palkar, P. 1995. *A three-dimensional approach to parallel matrix multiplication.* IBM Journal of Research and Development, 39(5), 575-582.

[32] Gupta, A., Kumar, V. 1993. *Scalability of parallel algorithms for matrix multiplication.* In 1993 International Conference on Parallel Processing-ICPP'93 (Vol. 3, pp. 115-123). IEEE.

[33] Gallegos, A., Torres, D., Álvarez, F., Soto, A. T. 2016. *Feature Subset Selection and Typical Testors Applied to Breast Cancer Cells.* Res. Comput. Sci., 121(1), 151-163.

[34] Ortíz-Posadas, M. R., Martínez-Trinidad, J. F., Ruiz-Shulcloper, J. 1996. *A new approach to differential diagnosis of diseases.* International journal of bio-medical computing, 40(3), 179-185.

[35] Heller, Don 1978. *A Survey of Parallel Algorithms in Numerical Linear Algebra.* SIAM Review, 20(4), 740–777. https://doi.org/10.1137/1020096

[36] Agarwal, R. C., Balle, S. M., Gustavson, F. G., Joshi, M., Palkar, P. 1995. *A three-dimensional approach to parallel matrix multiplication.* IBM Journal of Research and Development, 39(5), 575-582.

[37] Van, R. G., Drake, F. 2019. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[38] Francois Chollet et al. 2015. *Keras.* `https://github.com/fchollet/keras`

[39] Abadi M., Agarwal A., et al. 2015. *TensorFlow: Large-scale machine learning on heterogeneous systems*. Software available from `tensorflow.org`