ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

IMPLEMENTACIÓN COMPUTACIONAL DEL ALGORITMO LEX PARA EL CÁLCULO DE TESTORES TÍPICOS USANDO APRENDIZAJE SIMBÓLICO

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA MATEMÁTICA

PROYECTO DE INVESTIGACIÓN

INGRID JOHANA GUEVARA ROMERO
ingrid.guevara@epn.edu.ec

Director: EDUARDO ALBA CABRERA, PH.D.
ealba@usfq.edu.ec

Codirector: DIEGO FERNANDO RECALDE CALAHORRANO, PH.D.
diego.recalde@epn.edu.ec

QUITO, MAYO DE 2021

DECLARACIÓN

Yo INGRID JOHANA GUEVARA ROMERO, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

Ingrid Johana Guevara Romero

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por INGRID JOHANA GUEVARA ROMERO, bajo nuestra supervisión.

Eduardo Alba Cabrera, PH.D.
Director del Proyecto

Diego Fernando Recalde Calahorrano, PH.D.
Codirector del Proyecto

AGRADECIMIENTOS

A Dios por brindarme todas las oportunidades que me llevaron a salir adelante

A mi director Eduardo Alba de la Universidad San Francisco de Quito, por depositar su confianza en mí y ofrecerme su apoyo y guía .

A Salvador Godoy del Instituto Politécnico Nacional por su ayuda y valiosos consejos durante este proyecto.

A mi codirector Diego Recalde por supervisar el desarrollo de este trabajo.

A la Universidad San Francisco de Quito, por su colaboración con el equipo para realizar este trabajo.

A la Escuela Politécnica Nacional y a mis profesores por la formación que me brindaron.

DEDICATORIA

A mis padres Emerson y Marithza, a mis hermanos Diana y Sebastián, por su sacrificio, por sus palabras de aliento y por creer en mí.

A mi novio Boris, por estar presente tanto en los momentos más difíciles como en los más felices e inspirarme a ser una mejor persona.

May the force be with you.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El algoritmo LEX para el cálculo de testores típicos ha sido de gran utilidad para aplicaciones en reconocimiento de patrones. La reciente publicación sobre la relación entre testores típicos y transversales mínimos, potencialmente aumente la utilidad y aplicación del algoritmo dentro de los campos de hipergrafos y minería de datos. Desafortunadamente, la alta complejidad temporal que los algoritmos en ambas áreas poseen sigue siendo una dificultad. Por ello, constantemente se busca alternativas que puedan ayudar a sobrellevar problemas difíciles. En este trabajo se propone la inclusión de un comportamiento de aprendizaje simbólico dentro de la implementación del algoritmo LEX. El aprendizaje simbólico incorporado es una estrategia general que optimiza el proceso de búsqueda, y por tanto la eficiencia de los algoritmos de testores típicos y transversales mínimos. Además, el desempeño del algoritmo resultante es evaluado utilizando como referencias matrices de prueba cuidadosamente diseñadas.

Abstract

The LEX algorithm for computing typical testors has been notoriously useful in the context of pattern recognition applications. The recently published relationship between typical testors and minimal transversals, potentially increases this algorithm's usefulness and applicability within the hypergraphs and data mining fields. Unfortunately, the high time-complexity of algorithms in both areas still remains a major obstacle. Therefore, alternatives that can help overcome difficult problems are constantly being researched. In this paper we propose the inclusion of a symbolic learning behavior into the implementation of the LEX algorithm. The incorporated symbolic learning is a general strategy for optimizing the search process, and thus improving the efficiency of typical testors and minimal transversal algorithms. In addition, the performance of the resulting algorithm is assessed using carefully designed benchmark test matrices.

Capítulo 1

Introducción

A mediados de los 50, el concepto de testor apareció con el objetivo de desarrollar métodos lógicos matemáticos para encontrar fallas en circuitos eléctricos en los trabajos de Yablonskii y Cheguis [8, 9]. Desde entonces, el concepto de testor típico o también conocido como irreducible se ha ampliado y generalizado de varias formas [11]. En el contexto de reconocimiento de patrones comenzó a aplicarse como una herramienta útil para resolver una amplia variedad de problemas prácticos como diagnóstico médico [10], categorización de texto [28], resumen de documentos [29] y agrupación de documentos [26].

Encontrar el conjunto de todos los testores típicos en una matriz básica A denotado $\Psi^*(A)$, ha sido un problema estudiado durante años con mucho énfasis en los últimos diez. La bibliografía recoge muchos trabajos relacionados con este problema [10], [5], [27], [32]. Estos algoritmos, a pesar de su alta complejidad, han ayudado a abordar nuevos problemas prácticos interesantes.

Por otro lado, tenemos a la teoría de grafos, un campo muy conocido por su gran relevancia dentro de las matemáticas discretas durante las últimas décadas debido a su versatilidad al momento de modelar una amplia gama de fenómenos [24],[25], [34]. En particular, el concepto de transversal mínimo de un hipergrafo [7] es de gran utilidad, y ha sido utilizado en áreas como: inteligencia artificial, teoría de la fiabilidad, bases de datos, programación en enteros y la teoría del aprendizaje [13], [19]; y se han propuesto varios algoritmos para su cálculo [21], [14].

Las recientes publicaciones de la convergencia teórica entre los conceptos de Testor Típico e Hipergrafo Transversal [2] y [16] abren nuevas posibilidades para el estudio, desarrollo y aplicación de los mismos. En particular, los algoritmos para el cálculo del conjunto de todos los testores típicos y de transversales mínimos, ahora

pueden ser aplicados indistintamente en cualquiera de las áreas convergentes. No obstante, todos los algoritmos que se reportan en la literatura en cualquiera de los dos contextos anteriormente mencionados poseen una elevada complejidad temporal que limita la posibilidad de aplicar estos conceptos en situaciones que requieran el manejo de gran cantidad de datos, casos que paradójicamente serían los más interesantes y útiles. Esta limitación ha fomentado la búsqueda de alternativas que permitan un aumento del rendimiento de estos algoritmos mediante la reducción sustancial de su costo computacional.

El algoritmo LEX [5] es un caso particular de estos algoritmos, diseñado inicialmente para el cálculo de testores típicos. Su nombre se debe a que implementa un orden similar al orden lexicográfico para recorrer los subconjuntos de columnas de la matriz booleana que recibe como entrada. Este algoritmo tiene una estrategia eficiente que evita la revisión de muchos conjuntos de columnas.

En [16] se introduce una nueva estrategia para mejorar la eficiencia de los algoritmos de búsqueda de testores típicos denominada aprendizaje simbólico. El aprendizaje simbólico guía y optimiza el proceso de búsqueda de los algoritmos para el cálculo de testores típicos a través de reglas para dividir su espacio de búsqueda, ofreciendo un aprovechamiento más completo del conocimiento que estos adquieren.

Debido a su reciente publicación, esta estrategia ha sido probada únicamente en los algoritmos BR y YYC, elegidos para mostrar su funcionamiento en el mismo artículo donde el aprendizaje simbólico es propuesto. De ahí surge la motivación para incorporar esta estrategia al anteriormente mencionado algoritmo LEX. Incorporar el aprendizaje simbólico a LEX además se justifica por el hecho de que, tanto BR como YYC poseen propiedades muy diferentes.

En resumen, en este proyecto se propone una implementación del algoritmo LEX incorporando la estrategia de aprendizaje simbólico para mejorar la eficiencia de cálculo del conjunto de todos los testores típicos de una matriz booleana.

Comenzaremos con el capítulo 2, que está conformado por los conceptos relevantes para este estudio. Las nociones básicas de la teoría de testores se exponen en la sección 2.1, como son la definición de testor, matriz de diferencias, matriz básica, y testores típicos; en la sección 2.2 se encuentran los conceptos necesarios dentro de la teoría de hipergrafos, tales como matriz de incidencia, hipergrafo simple, transversales y transversales mínimos. La sección 2.3 contiene los teoremas fundamentales para establecer la relación entre testor típico y transversal mínimo. En 2.4 se encuen-

tran las definiciones y propiedades a partir de las cuales se construyó el algoritmo LEX. Además, en la sección 2.5 se detalla en qué consiste el aprendizaje simbólico; se incluyen varios conceptos, tanto en términos de testores típicos como en términos de transversales mínimos, que sirven para explicar su funcionamiento. A través de estas definiciones, es posible hacer una partición del espacio de búsqueda de cualquier algoritmo. Usando esta descripción, se derivan cuatro reglas importantes para lograr la implementación de la estrategia. Por último, es necesario entender cómo crear matrices sintéticas de prueba para algoritmos de búsqueda de testores típicos, lo cual se esclarece en la sección 2.6.

Una vez que el lector haya revisado los conceptos básicos de la teoría de testores e hipergrafos, el algoritmo LEX, los fundamentos del aprendizaje simbólico y la generación de matrices de prueba, llegará al capítulo 3 donde se presenta el objetivo principal del proyecto, la incorporación del aprendizaje simbólico en LEX; que incluye el desarrollo de esta, así como el algoritmo resultante puesto a prueba.

Primeramente, se determinan las etapas del algoritmo original en las cuales se puede utilizar la estrategia de aprendizaje simbólico y se presenta el código del nuevo algoritmo, al cual hemos denominado LEX*. Para las pruebas se ha diseñado un conjunto de matrices, hemos elegido cuatro matrices base que pueden encontrarse en el apéndice A, a partir de las cuáles generamos las matrices sintéticas que se usaron en el experimento. Mediante estas matrices creamos tres matrices con estructuras diferentes, variando el número de filas y columnas que cada una posee. A las tres matrices aplicamos tres operadores, descritos en 2.6; de modo que, entre las matrices generadas existan algunas con gran densidad de testores típicos y otras de grandes dimensiones, pero escasa cantidad de testores típicos. El número de filas, columnas y testores típicos resultante de cada matriz creada se especifican en la sección 3.3.1.

Los resultados en la sección 3.3.2 están distribuidos de la siguiente manera: para cada pareja matriz operador se dispone de una figura a la izquierda que representa la comparación del tiempo de ejecución del algoritmo original y el tiempo del algoritmo nuevo, la figura a la derecha representa el total de revisiones de LEX en contraste con el total de revisiones de LEX*, ambas figuras van acompañadas de un cuadro con los datos arrojados en los experimentos.

Para terminar, el capítulo 4 abarca las conclusiones obtenidas a partir de las pruebas realizadas. El apéndice B se compone del código del algoritmo LEX*, junto con las funciones complementarias necesarias, todo implementado en el lenguaje

de programación Matlab versión R2019b. Para los experimentos hemos usado una unidad de alto procesamiento Nvidia DGX Workstation.

Capítulo 2

Marco Teórico

2.1. Teoría de Testores

La teoría de testores es un instrumento útil para la selección y evaluación de rasgos en el reconocimiento de patrones, que se ha aplicado para resolver una variedad de problemas como diagnóstico médico [10], categorización de texto [28], resumen de documentos [29] y agrupación de documentos [26]. En esta sección presentaremos las definiciones básicas detrás de esta teoría.

Denotemos por U el universo de objetos de estudio, cada uno de ellos descritos en términos de los rasgos $R = \{x_1, \ldots, x_n\}$ y agrupados en las clases $K_1, \ldots, K_r$ con $r \geq 2$, las cuales no son necesariamente disjuntas. Sea $M \subset U$ una muestra de entrenamiento de objetos $O_1, \ldots, O_m$ tal que $M = \bigcup_{i=1}^{r} K_i^{'}$ donde $K_i^{'} \subseteq K_i$. Cada rasgo x_i toma valores en un conjunto M_i $(i = 1, \ldots, n)$.

Sea D un operador de descripción

$$D\colon U \to \prod_{i=1}^{n} M_i$$

D asocia a cada elemento O en U un punto en $M_1 \times \ldots \times M_n$ denotado por $D(O) = (x_1(O), \ldots, x_n(O))$, llamado descripción del objeto; donde $x_i(O) \quad i = 1, \ldots, n$ representa el valor del rasgo x_i para el objeto O.

DEFINICIÓN 2.1 (Criterio de comparación de disimilaridad) *Se denomina criterio de comparación del rasgo x_i a la función $C_i\colon M_i \times M_i \to \{0, 1\}$, tal que si C_i es un criterio de disimilaridad se cumple que $Ci(x_i(O_k), x_i(O_j))$ es igual a 1 si se considera*

que los objetos O_k y O_j ($k \neq j$) son diferentes en el rasgo x_i y 0 si son iguales.

EJEMPLO 2.2 *Observemos los siguientes criterios de comparación de disimilaridad*

1. *Criterio de comparación de igualdad estricta:*

$$Ci(x_i(O_k), x_i(O_j)) = \begin{cases} 0 & si\ x_i(O_k) = x_i(O_j), \\ 1 & caso\ contrario \end{cases}$$

2. *Criterio de comparación de error admisible:*

$$Ci(x_i(O_k), x_i(O_j)) = \begin{cases} 0 & si\ |x_i(O_k) - x_i(O_j)| \leq \epsilon, \\ 1 & caso\ contrario \end{cases}$$

con $\epsilon > 0$ y $k, j = 1, \ldots, n$.
Notemos que este último es solo aplicable a rasgos numéricos.

DEFINICIÓN 2.3 (Matriz de aprendizaje) *Sea $\overline{\alpha}(O) = (\overline{\alpha_1}(O), \ldots, \overline{\alpha_r}(O))$ el r-uplo de pertenencia de un objeto O a las clases $K_1', \ldots, K_r'$. Donde*

$$\overline{\alpha_j}(O) = \begin{cases} 1 & si\ O \in K_j' \\ 0 & caso\ contrario \end{cases}$$

La matriz de aprendizaje (MA) es aquella que contiene las descripciones de los objetos de la muestra de entrenamiento y el r-uplo de pertenencia. Es decir, el tuplo $(D(O_1), \overline{\alpha}(O_1), \ldots, D(O_m), \overline{\alpha}(O_m))$.

EJEMPLO 2.4 *Consideremos la siguiente matriz de aprendizaje con objetos pertenecientes a 3 clases*

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	$\overline{\alpha}$		
$\mathbf{O_1}$	1.3	10	True	azul	1	0	0
$\mathbf{O_2}$	-1.3	13	True	rojo	1	0	0
$\mathbf{O_3}$	2.4	16	False	azul	0	1	0
$\mathbf{O_4}$	-2.4	19	False	blanco	0	1	0
$\mathbf{O_5}$	2.6	10	True	verde	0	0	1
$\mathbf{O_6}$	-2.6	20	False	rojo	0	0	1

DEFINICIÓN 2.5 (Matriz de Diferencias) *La matriz de diferencias de MA (denotada MD) es una matriz booleana que se obtiene de comparar todos los objetos de clases*

distintas. Está formada por las m' filas $S_{ij} = (\delta_1^{ij}, \ldots, \delta_{ij}^n)$ con $i, j = 1, \ldots, m$, $i \neq j$; donde $\delta_p^{ij} = C_p(x_p(O_i), x_p(O_j))$ con $p = 1, \ldots, n$ y $m' = \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |K_i'||K_j'|$.

Dicho de otra forma, cada fila en una matriz de diferencia representa la comparación de dos objetos de diferentes clases para cada uno de los rasgos considerados. Generalmente una MD contiene mucha información redundante, por lo que comúnmente es reducida para aplicaciones.

DEFINICIÓN 2.6 *Sean a_{i_p}, a_{i_q} filas de una MD, a_{i_p} es subfila de a_{i_q} si $\forall j \quad a_{i_q,j} = 0 \Rightarrow a_{i_p,j} = 0$ y además $\exists j_0$ tal que $a_{i_q,j_0} = 1 \wedge a_{i_p,j_0} = 0$*

Usando la definción de subfila podemos construir una matriz más pequeña que conserve la información más importante de una MD.

DEFINICIÓN 2.7 *Sea i_q una fila de una MD, i_q es básica si no existe ninguna fila de MD que sea una subfila de i_q. La Matriz Básica (MB) de MD es aquella compuesta únicamente de filas básicas, sin repetir ninguna de ellas.*

EJEMPLO 2.8 *La matriz de diferencias del ejemplo 2.4 es la siguiente*

Filas	**Objetos comparados**		
1110	O_1	con	O_3
1111	O_1	con	O_4
1001	O_1	con	O_5
1111	O_1	con	O_6
1111	O_2	con	O_3
1111	O_2	con	O_4
1101	O_2	con	O_5
1110	O_2	con	O_6
0111	O_3	con	O_5
1101	O_3	con	O_6
1111	O_4	con	O_5
0001	O_4	con	O_6

Para comparar los objetos en los rasgos x_1 y x_2 usamos el criterio de comparación de error admisible con $\epsilon = 0{,}5$ y $\epsilon = 2$ respectivamente, para los rasgos x_3 y x_4 usamos el criterio de igualdad estricta.
Su matriz básica esta dada por:

$$\mathbf{MB} = \begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 & X_4 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \right] \end{array}$$

Notaremos como $MB_{|\tau}$ a la sub-matriz obtenida eliminando de MB las columnas que no pertenecen al subconjunto de rasgos $\tau \subseteq R$.

DEFINICIÓN 2.9 (Testor y Testor Típico) *El subconjunto de rasgos $\tau \subseteq R$ de una MB es un testor si $MB_{|\tau}$ no contiene ninguna fila compuesta exclusivamente de ceros. τ es un testor típico (TT), o también llamado irreducible, si al remover cualquier rasgo de τ aparece al menos una fila de ceros.*

Al conjunto de todos los testores de una matriz básica A se la denota como $\psi(A)$ y al conjunto de todos los testores típicos como $\psi^*(A)$.

2.2. Teoría de Hipergrafos

Uno de los campos más notorios de las matemáticas discretas es la teoría de grafos, y un concepto particularmente interesante es el de transversales mínimos dentro de un hipergrafo, cuyas aplicaciones incluyen inteligencia artificial, teoría de la fiabilidad, bases de datos, programación en enteros y la teoría del aprendizaje [19],[13]. Esta sección se enfoca en exponer las nociones más relevantes de la teoría de hipergrafos, para posteriormente vincular el concepto de transversal mínimo y testor irreducible.

En teoría de grafos un hipergrafo es la generalización del concepto tradicional de grafos, donde una arista no se restringe a conectar únicamente dos vértices.

DEFINICIÓN 2.10 (Hipergrafo) *Un hipergrafo $\mathcal{H}$ es un par ordenado $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ donde $\mathcal{V} = \{v_1, \ldots, v_n\}$ es un conjunto finito de objetos y $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ un recubrimiento de $\mathcal{V}$ ($\mathcal{E}_i \neq \emptyset, i = 1, \ldots, m$ y $\cup_{i=1}^{m} \mathcal{E}_i = \mathcal{V}$).*

Los elementos de $\mathcal{V}$ se los conoce como vértices o nodos, y los elementos de $\mathcal{E}$ como hiperaristas.

EJEMPLO 2.11 *Consideremos el hipergrafo $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ con $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, v_6.v_7\}$ y $\mathcal{E} = \{\mathcal{E}_1 = (v_1, v_2, v_3), \mathcal{E}_2 = (v_2, v_3), \mathcal{E}_3 = (v_3, v_5, v_6), \mathcal{E}_4 = (v_4)\}$*

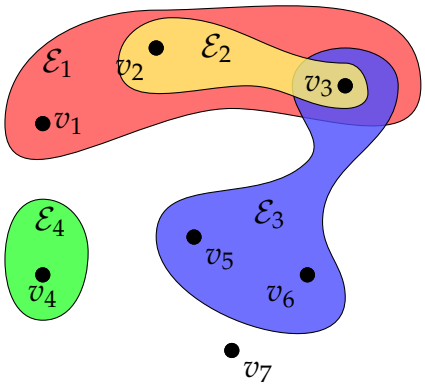


Figura 2.1: Representación gráfica de un hipergrafo

DEFINICIÓN 2.12 (Matriz de incidencia) *La matriz de incidencia* $A = [a_{ij}]_{m\times n}$ *de un hipergrafo* $\mathcal{H}$ *es aquella cuyas filas y columnas corresponden a los vértices e hiperaristas de* $\mathcal{H}$ *respectivamente, tal que* $a_{ij} = 1$ *si* $v_i \in \mathcal{E}_j$ *y* $a_{ij} = 0$ *caso contrario.*

DEFINICIÓN 2.13 (Hipergrafo Simple) *Un hipergrafo* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *se llama simple si para cada par* $(\mathcal{E}_i, \mathcal{E}_j)$ *se cumple que si* $\mathcal{E}_j \subseteq \mathcal{E}_i \Rightarrow j = i$.

DEFINICIÓN 2.14 (Transversal) *Sea* $H = (\mathcal{V}, \mathcal{E})$, $\tau \subseteq \mathcal{V}$ *es un transversal de* $\mathcal{H}$ *si* $\forall \mathcal{E}_i \in \mathcal{E} \quad \tau \cap \mathcal{E}_i \neq \emptyset$. *Un transversal* τ *es llamado mínimo si ningún subconjunto propio* τ' *de* τ *es un transversal de* $\mathcal{H}$.

DEFINICIÓN 2.15 *El hipergrafo transversal de* $\mathcal{H}$ *denotado por* $Tr(\mathcal{H})$ *es la familia de todos los transversales mínimos de* $\mathcal{H}$.

2.3. Relación entre Testor Típico y Transversal Mínimo

A primera vista, la teoría de testores y la teoría de hipergrafos aparentan ser diferentes. Pero en [2] las definiciones de matriz básica, matriz de incidencia e hipergrafo simple permiten crear una relación de ambos conceptos mediante el siguiente teorema:

TEOREMA 2.16 *Al transponer una matriz de incidencia de un grafo simple, la matriz resultante cumple con todos los requisitos para ser una matriz básica.*

Este resultado da lugar a lo siguiente:

TEOREMA 2.17 *Sea* $\psi^*(B) = \{\tau_1, \ldots, \tau_s\}$ *la familia de testores típicos de una matriz básica* B. *Sea* $Tr(\mathcal{H})$ *el hipergrafo transversal de un hipergrafo simple* $\mathcal{H}$ *cuya matriz*

de incidencia es $B^\intercal$, entonces $\psi^*(B) = Tr(\mathcal{H})$.

Es decir, los transversales mínimos de un hipergrafo simple $\mathcal{H}$ y los testores típicos de la transpuesta de su matriz de incidencia $\mathcal{H}$ son los mismos.

2.3.1. Implicaciones

La mayor consecuencia de esta equivalencia es el hecho de que se puede aplicar cualquier algoritmo conocido para el cálculo de testores típicos o transversales mínimos, indistintamente entre sí.

Otros efectos de esta relación yacen en la propiedad de dualidad [23] que establece que $Tr(Tr(\mathcal{H})) = \mathcal{H}$, es decir:

1. El hipergrafo transversal de un hipergrafo simple es también un hipergrafo simple.
2. El hipergrafo transversal de un hipergrafo dual es el hipergrafo original.

En teoría de testores la propiedad de dualidad da lugar a los siguientes lemas:

LEMA 2.18 *Sea $\psi^*(B) = \{\tau_1, \ldots, \tau_s\}$ la familia de testores típicos de una matriz básica B, entonces la matriz conformada por estos $[\psi^*(B)]$ es también una matriz básica.*

LEMA 2.19 *Sea $\psi^*(B) = \{\tau_1, \ldots, \tau_s\}$ la familia de testores típicos de una matriz básica B, entonces la matriz conformada por estos $[\psi^*[\psi^*(B)]] = B$ es también una matriz básica.*

2.4. Algoritmo LEX

La teoría de testores ha evolucionado a través del diseño de algoritmos para determinar familias de testores típicos, como por ejemplo BT [31], CT_EXT [18] , YYC [4] y BR [27]. Asimismo, se han desarrollado algoritmos para encontrar transversales mínimos en un hipergrafo como BMR [6], DL [12] y KS [23].

LEX es un algoritmo de escala exterior para el cálculo de testores típicos introducido en [5], consiste en la construcción de listas de rasgos que posean la propiedad de tipicidad y comprobar si este conjunto constituye un testor. En esta búsqueda se definen ciertos saltos que permiten obviar algunos conjuntos usando las propiedades del algoritmo, estas se detallarán más adelante.

El orden que se usa sobre el conjunto potencia de los rasgos es similar al orden lexicográfico con el que se comparan las cadenas de caracteres, de allí el algoritmo recibe su nombre.

2.4.1. Propiedades del algoritmo LEX

La notación $[\mathbf{X}|\mathbf{Y}]$ representa una lista ordenada de rasgos donde X es el primero rasgo de la lista y Y es una lista ordenada de los elementos restantes.

DEFINICIÓN 2.20 *El símbolo $\ll$ denota una relación de orden sobre el conjunto potencia de todos los rasgos, que cumple:*

1. $[X_i|Y] \ll [X_j|Y]$ *con* $i < j$
2. $[X_i|Y] \ll [X_i|Z]$ *con* $Y \ll Z$
3. *Para toda* $Y = [X|Z]$ *se cumple* $[\,] \ll Y$ *donde* $[\,]$ *es una lista vacía.*

Cuando se hable de listas será refiriéndose a una lista ordenada de rasgos.

El operador "+"denota la concatenación de listas, por ejemplo:

$$[X_1, X_2] + [X_3, X_4] = [X_1, X_2, X_3, X_4]$$

DEFINICIÓN 2.21 *Sea una lista de rasgos* $l = [X_{j_1}, ..., X_{j_s}]$ *de una MB y un rasgo* X_t. *Llamaremos conjunto de filas típicas de* X_t *con respecto a* l *al conjunto de índices de filas de MB que tienen 1 en la columna* X_t *y ceros en las de* X_{j_k}, *con* $1 \leq k \leq s$, $j_k \neq t$. *Lo denotaremos* $F^l_{X_t}$.

EJEMPLO 2.22 *Consideremos la siguiente MB:*

$$\begin{matrix} X_1 & X_2 & X_3 & X_4 & X_5 \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Si $l = [X_1, X_3]$, *entonces* $F^l_{X_2} = \{3\}$.

DEFINICIÓN 2.23 *Sea* $l = \left[X_{j_1}, ..., X_{j_s}\right]$ *una lista de rasgos de MB, denotaremos como* cr^l_i *a la cantidad de rasgos de* l *que tienen valores unitarios en la fila* i.

EJEMPLO 2.24 *En el ejemplo 2.22 tenemos que:* $cr^l_1 = 1$, $cr^l_2 = 1$, $cr^l_3 = 0$, $cr^l_4 = 2$.

DEFINICIÓN 2.25 *Dada una lista de rasgos de MB,* $l = \left[X_{j_1}, ..., X_{j_s}\right]$. *El hueco de* l *es el rasgo* X_p *de* l *tal que*

$$p = max\{j_q | X_{j_q}, X_{j_{q+1}} \in l \wedge j_{q+1} \neq j_q + 1\}$$

con $j_1 \leq p \leq j_s$.

PROPOSICIÓN 2.26 *Sean* $l = \left[X_{j_1}, ..., X_{j_s}\right]$ *y* $X_t \notin l$. *Sea* U *el conjunto de filas de MB donde* X_t *tiene rasgos unitarios. Si existe* $X_{j_k} \in l$ *tal que* $F^l_{X_{j_k}} \subset U$, *entonces* X_t *no formará parte de ningún TT con los rasgos de* l.

PROPOSICIÓN 2.27 *Sean* $l = \left[X_{j_1}, ..., X_{j_s}\right]$ *y* $X_t \notin l$. *Si* $\left|F^l_{X_t}\right| = 0$ *entonces* X_t *no formará parte de ningún TT con los rasgos de* l.

Si se cumplen cualquiera de las dos proposiciones anteriores se dice que X_t es un rasgo **excluyente**.

PROPOSICIÓN 2.28 *Sean* $l = \left[X_{j_1}, \ldots, X_{j_s}\right]$ *una lista asociada a un TT, con* $X_{j_s} = X_n$ *y* X_p *hueco de* l. *Sea* $l' = \left[X_{j_1}, \ldots, X_{j_k}, X_{p+1}\right]$ *con* $j_1 \leq \ldots \leq j_k = p - 1$. *No existe* λ *tal que este asociada a un TT y* $l \ll \lambda \ll l'$.

COROLARIO 2.29 *Si* $l = \left[X_{j_1}, \ldots, X_{j_s}\right]$ *es una lista asociada a un TT, con* $X_{j_s} = X_n$ *y no existe hueco de* l, *entonces no existe* $l \ll \lambda$ *tal que este asociada a un TT.*

COROLARIO 2.30 *Sea* $l = \left[X_{j_1}, \ldots, X_{j_s}\right]$ *tal que* $X = X_n$ *y* $l + [X]$ *es una lista asociada a un no testor. Si* X_p *hueco de* l, *sea* $l' = \left[X_{j_1}, \ldots, X_{j_k}, X_{p+1}\right]$ *con* $j_1 \leq \ldots \leq j_k = p - 1$, *entonces no existe una lista* λ *tal que este asociada a un testor y* $l \ll \lambda \ll l'$.

2.4.2. Código LEX

Algoritmo 1 LEX

Entrada: *Matriz básica MB*

Salida: *El conjunto de todos los testores típicos de MB*

1: *Ordenación de MB*

a: *Encontrar la fila con cantidad mínima de 1's; de existir más de una, escoger cualquiera de ellas. Ponerla como primera fila en MB.*

b: *Ordenar las columnas poniendo indistintamente como primeras las que tengan un 1 en la primera fila.*

2: *Inicialización*

a: $l = [\,]$, $X = X_1$ (X *es el primer candidato a elemento de* l).

3: *Evaluación del candidato* X

a: **Si** $l = [\,]$ *y la columna correspondiente a* X *tiene cero en la primera fila de MB* **entonces** *FIN.*

b: **Si** X *es excluyente con* l **entonces** *ir al paso 4 (proposiciones 2.26 y 2.27)*, no se acepta el candidato

c: **Si** *para toda fila i de MB* $cr_i^{l+[X]} > 0$ **entonces** *guardar* $l + [X]$ *es un TT, ir al paso 4*

d: **Si** $X = X_n$ **entonces** *ir al paso 4 (Corolario 2.30)*

e: *Hacer* $l = l + [X]$*, se acepta el candidato. Actualizar los valores de* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para todos los elementos* X_t *en* l.

4: *Selección del nuevo candidato*

a: **Si** $X \neq X_n$ **entonces** *sea j el índice de X en MB, hacer* $X = X_{j+1}$ *e ir al paso 3*

b: **Si** $l = [\,]$ **entonces** *FIN.*

c: **Si** $l + [X]$ *fue un TT o X no fue excluyente con* l **entonces**

Si *existe el hueco* X_p *de* $l + [X]$ **entonces**

I. *Hacer* $X = X_{p+1}$

II. *Eliminar de* l *todos los elementos desde* X_p *hasta* X_{j_s} *(proposición 2.28)*

III. *Actualizar* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para cada* X_t *de* l*. Ir al paso 3.*

Caso contrario *no existe* X_p*, FIN (corolario 2.29)*

d: *Hacer* $X = X_{j_s}$*, donde* X_{j_s} *es el último rasgo de* l *y* $l = l \setminus [X_{j_s}]$*. Actualizar* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para cada* X_t *de* l.

e: *Retornar al paso 4.*

EJEMPLO 2.31 *Consideremos la matriz del ejemplo 2.22*

$$A = \begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 & X_4 & X_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

1. *Ordenación de MB: La primera fila tiene igual o menor número de valores unitarios que las demás, la dejaremos como primera. Las columnas que tienen valores unitarios en la primera fila están colocadas como primeras, de modo que el orden se mantiene.*

2. *Inicialización:*

   a) $l = [\,]$, $X = X_1$

3. *Evaluación del candidato: Como no se cumplen los requisitos de los literales a hasta d, el candidato es aceptado y se añade a* l*. Además, se actualizan los valores de* cr_i^l *para todas las filas y* $F_{X_t}^l$ *para todos los elementos de* l*.*

   e) $l = [X_1]$
   $cr_1^l = cr_4^l = 1$
   $cr_2^l = cr_3^l = 0$
   $F_{X_1}^l = \{1, 4\}$

4. *Selección del nuevo candidato*
   Como $X \neq X_n = X_5$*, el siguiente candidato es* X_2*.*

   a) $X = X_2$ *y regresamos al paso 3.*

Continuando con los pasos establecidos en el algoritmo 1, se obtiene que la familia de testores típicos de la matriz A *es* $\psi^*(A) = \{[X_1, X_3, X_4], [X_1, X_5], [X_2, X_3]\}$,

2.5. Aprendizaje Simbólico

La teoría de hipergrafos permite modelar problemas en diversos campos científicos. Existen diferentes propuestas de algoritmos que permiten encontrar la familia de transversales mínimos en un hipergrafo, o lo que se conoce como "The transversal generation problem"[15]. En general, se exploran conjuntos de vértices que satisfacen condiciones específicas, el mecanismo de búsqueda depende de la representación del hipergrafo que maneja el algoritmo. Usualmente se opta por una matriz de incidencia.

La estrategia conocida como aprendizaje simbólico [16] ha sido desarrollada con el objetivo de aprovechar por completo este conocimiento, reduciendo el espacio de búsqueda de transversales mínimos. Es aplicable a una amplia variedad de algoritmos, incluso cuando estos tengan un método de exploración diferente. Adicionalmente, las implicaciones de los teoremas 2.16 y 2.17 posibilitan su uso en algoritmos destinados al cálculo de testores típicos. De hecho, pese a que el aprendizaje simbólico está explicado en términos de transversales, los algoritmos para encontrar testores irreducibles fueron la inspiración para su creación.

En esta sección se explicará en qué consiste la estrategia y de esta forma cumplir con el objetivo de este trabajo, incorporarla en LEX.

2.5.1. Partición del espacio de búsqueda

Las definiciones descritas a continuación son útiles para caracterizar un conjunto de rasgos:
Sea A una matriz básica, y τ un subconjunto de rasgos de A.

DEFINICIÓN 2.32 (Máscara de aceptación de τ) *La máscara de aceptación am_τ corresponde a la tupla binaria en la cual el i-ésimo elemento tiene 1 si la fila i de $A_{|\tau}$ tiene al menos un 1 en las columnas de τ y 0 caso contrario.*

DEFINICIÓN 2.33 (Máscara de compatibilidad de τ) *La máscara de compatibilidad cm_τ corresponde a la tupla binaria en la cual el i-ésimo elemento tiene 1 si la fila i de $A_{|\tau}$ tiene solo un 1 en las columnas de τ y 0 caso contrario.*

DEFINICIÓN 2.34 (Máscara de tipicidad de τ) *La máscara de tipicidad tm_τ corresponde a una tupla integral en la cual el i-ésimo elemento es el número de filas típicas del rasgo x_i de $A_{|\tau}$.*

Las máscaras de aceptación, compatibilidad y tipicidad permiten definir los siguientes índices:

DEFINICIÓN 2.35 (Índice de error de testor de τ) *Notado $e_T(\tau)$ es el número de columnas integradas únicamente por ceros en $A_{|\tau}$.*

DEFINICIÓN 2.36 (Índice de error típico de τ) *Notado $e_{Ty}(\tau)$, es el número de rasgos en τ que no tienen al menos una fila típica en $A_{|\tau}$, por tanto, corresponde al número de entradas cero en tm_τ.*

De la misma manera, los siguientes conceptos permiten clasificar conjuntos de vértices en una matriz de incidencia.

Dado un hipergrafo $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ y su matriz de incidencia A.

DEFINICIÓN 2.37 *Sea $\tau \subseteq \mathcal{V}$. Notaremos como $A_{|\tau}$ a la submatriz que se obtiene retirando todas las columnas que no están en τ.*

DEFINICIÓN 2.38 *Un subconjunto $\tau \subseteq \mathcal{V}$ es un transversal si y solo si $A_{|\tau}$ no contiene ninguna fila compuesta exclusivamente de ceros.*

DEFINICIÓN 2.39 *Un subconjunto $\tau \subseteq \mathcal{V}$ es mínimo si y solo si $A_{|\tau}$ contiene todas las filas de una matriz identidad de tamaño n, donde $n = |\tau|$.*

DEFINICIÓN 2.40 (Dominación) *El vértice v_1 domina al vértice v_2 si y solo si $\forall \mathcal{E}_i \in \mathcal{E}$ si $v_2 \in \mathcal{E}_i \Rightarrow v_1 \in \mathcal{E}_i$. La dominación se refiere a que en cualquier fila de A donde v_2 tiene valor uno v_1 también. La dominación mútiple es cuando un subconjunto de vértices domina un vértice.*

Cuando existe la propiedad de dominación entre dos vértices o un subconjunto de vértices y un vértice, diremos que el par es **incompatible**. Todas estas definiciones son intercambiables entre testores y transversales por la equivalencia publicada en [2]. Por tanto, el espacio de búsqueda para cualquier problema de transversales mínimos y testores típicos está particionado en cuatro clases:

- Testores (no típicos) /Transversales (no mínimos): los conjuntos que satisfacen la definición 2.38 pero no la definición 2.39 ($e_T(\tau) = 0$ y $e_{Ty}(\tau) \neq 0$).
- Conjuntos típicos (no testores) /mínimos (no transversales): los conjuntos que satisfacen la definición 2.39 pero no la definición 2.38 ($e_T(\tau) \neq 0$ y $e_{Ty}(\tau) = 0$).
- Testores típicos/Transversales mínimos: los conjuntos que satisfacen la definición 2.38 y la definición 2.39 ($e_T(\tau) = 0$ y $e_{Ty}(\tau) = 0$).

- Indeterminados: no satisfacen la definición 2.38 ni la definición 2.39 ($e_T(\tau) \neq 0$ y $e_{Ty}(\tau) \neq 0$).

EJEMPLO 2.41 *Consideremos la siguiente MB*

$$\begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Cuadro 2.1: Clasificación de subconjuntos de rasgos en una matriz básica

τ	$e_T(\tau)$	$e_{Ty}(\tau)$	Clasificación
X_1, X_2, X_3, X_4	0	1	Testor
X_5, X_6	0	0	Testor Típico
X_1, X_4, X_6	4	1	Indeterminado
X_1, X_3	3	0	Típico

Estas propiedades pueden ser obvias, pero facilitan identificar aquellos conjuntos que no necesitan ser sometidos a prueba. Así, se han construido las siguientes reglas:

1. Encontrar un conjunto transversal/testor, entonces se descartan todos sus superconjuntos ya que no pueden ser mínimos.
2. Encontrar un conjunto mínimo/típico, entonces se descartan todos sus subconjuntos ya que no pueden ser mínimos.
3. Encontrar un conjunto transversal mínimo/testor típico, entonces se descartan todos sus subconjuntos y superconjuntos ya que no pueden ser mínimos.
4. Cualquier otro conjunto encontrado (indeterminado) contiene algún par de vértices incompatibles y también debe ser descartado.

EJEMPLO 2.42 *Para los conjuntos de rasgos del ejemplo 2.41, los conjuntos descartados son los siguientes:*

Cuadro 2.2: Conjuntos descartados después de analizar τ

τ	Conjuntos descartados a partir de τ
X_1, X_2, X_3, X_4	$\{X_1, X_2, X_3, X_4, X_5\}, \{X_1, X_2, X_3, X_4, X_6\},$ $\{X_1, X_2, X_3, X_4, X_5, X_6\}$
X_5, X_6	$\{X_1, X_2, X_3, X_4, X_5, X_6\}, \{X_2, X_3, X_4, X_5, X_6\},$ $\{X_1, X_3, X_4, X_5, X_6\}, \{X_1, X_2, X_4, X_5, X_6\},$ $\{X_1, X_2, X_3, X_5, X_6\}, \{X_1, X_4, X_5, X_6\}, \{X_2, X_4, X_5, X_6\},$ $\{X_3, X_4, X_5, X_6\}, \{X_1, X_2, X_5, X_6\},$ $\{X_1, X_3, X_5, X_6\}, \{X_2, X_3, X_5, X_6\}, \{X_1, X_5, X_6\},$ $\{X_2, X_5, X_6\}, \{X_3, X_5, X_6\}, \{X_4, X_5, X_6\},$ $\{X_5\}, \{X_6\}$
X_1, X_4, X_6	$\{X_1, X_4\}, \{X_1, X_2, X_4\}, \{X_1, X_3, X_4\}, \{X_1, X_4, X_5\}, \{X_1, X_4, X_6\}$ $\{X_1, X_2, X_3, X_4\}, \{X_1, X_2, X_4, X_5\}, \{X_1, X_2, X_4, X_6\},$ $\{X_1, X_3, X_4, X_5\}, \{X_1, X_3, X_4, X_6\}, \{X_1, X_4, X_5, X_6\}, \{X_1, X_2, X_3, X_4, X_5\},$ $\{X_1, X_2, X_3, X_4, X_6\}, \{X_1, X_2, X_4, X_5, X_6\}, \{X_1, X_3, X_4, X_5, X_6\},$ $\{X_1, X_2, X_3, X_4, X_5, X_6\}$
X_1, X_3	$\{X_1\}, \{X_3\}$

Observemos que el conjunto $\{X_1, X_4, X_6\}$ posee el par incompatible $\{X_1, X_4\}$, pues X_4 domina X_1. Por consiguiente, todos los conjuntos que contienen este par son descartables.

Este análisis no suele ser implementado por completo. Sin embargo, mediante este aprendizaje se preserva la naturaleza del algoritmo proporcionándole un uso completo de su conocimiento.

2.5.2. Implementación de la estrategia

La regla 4 es comúnmente la menos utilizada. Sin embargo, si se añade no ofrece un riesgo para el rendimiento del algoritmo e impide arribar a información ya conocida.

La interacción entre el algoritmo anfitrión y la estrategia es la siguiente:

- Dependiendo de la taxonomía del algoritmo, este usa su orden de búsqueda predeterminado.
- Si el conjunto examinado no es un transversal mínimo/ testor típico, se busca incompatibilidades (simples o múltiples) para almacenarlas en una tabla.

- En cualquier momento el algoritmo puede consultar la tabla de incompatibilidades para seleccionar futuros subconjuntos de vértices a probar.

Las secciones 2.5.1 y 2.5.2 son una descripción breve de lo expuesto en [16] y [17], con la finalidad de desarrollar el propósito de este proyecto.

2.6. Matrices sintéticas de prueba para algoritmos de búsqueda de testores típicos

Para poner a prueba un algoritmo de cálculo de testores típicos se debe tener en cuenta qué parámetros serán considerados, ya que un estudio sin punto de referencia carece de validación. De este modo, es importante la selección de las matrices que se emplearán para que sea viable elaborar conclusiones sensatas.

Al tener como entrada una matriz, estos algoritmos son susceptibles a la cantidad de filas, columnas y transversales mínimos o testores irreducibles que estas poseen. Afortunadamente, [1] propicia la generación de matrices de prueba de las cuales se puede conocer de antemano el resultado final. Empleando estos procedimientos se puede determinar un comportamiento computacional específico variando ciertos parámetros.

2.6.1. Operadores de matrices

Sean $A = [a_{ij}]_{m \times n}$ y $B = [b_{ij}]_{m' \times n'}$ dos MB

DEFINICIÓN 2.43 (Operador θ) *La operación $\theta(A, B)$ produce una nueva matriz donde cada fila de A es concatenada a la derecha con cada fila de B como se muestra a continuación:*

$$\begin{bmatrix} a_{11} & \dots & a_{1n} & b_{11} & \dots & b_{n'} \\ & \vdots & & & \vdots & \\ a_{11} & \dots & a_{1n} & b_{m'1} & \dots & b_{m'n'} \\ & \vdots & & & \vdots & \\ a_{m1} & \dots & a_{mn} & b_{11} & \dots & b_{1n'} \\ & \vdots & & & \vdots & \\ a_{m1} & \dots & a_{mn} & b_{m'1} & \dots & b_{m'n'} \end{bmatrix}$$

DEFINICIÓN 2.44 (Operador γ) *La operación $\gamma(A, B)$ produce una nueva matriz donde A se encuentra en la esquina superior derecha, seguida de ceros en todas las columnas de B. La matriz B se encuentra en la esquina inferior izquierda, y está precedida de ceros en todas las columnas de A. Es decir:*

$$\begin{bmatrix} a_{11} & \dots & a_{1n} & 0 & \dots & 0 \\ & \vdots & & & \vdots & \\ a_{11} & \dots & a_{1n} & 0 & \dots & 0 \\ & \vdots & & & \vdots & \\ 0 & \dots & 0 & b_{11} & \dots & b_{n'} \\ & \vdots & & & \vdots & \\ 0 & \dots & 0 & b_{m'1} & \dots & b_{m'n'} \end{bmatrix}$$

DEFINICIÓN 2.45 (Operador φ) *La operación $\varphi(A, B)$ es el resultado de concatenar A y B, suponiendo que estas tienen el mismo número de filas ($m = m'$).*

$$\begin{bmatrix} a_{11} & \dots & a_{1n} & b_{11} & \dots & b_{1n'} \\ & \vdots & & & \vdots & \\ a_{11} & \dots & a_{1n} & b_{m'1} & \dots & b_{1mn'} \end{bmatrix}$$

Se puede verificar que los operadores φ, θ, γ aplicados a una matriz básica darán como resultado una nueva matriz básica. Si todos sus argumentos son matrices básicas, los tres operadores son asociativos. Así, si aplicamos N veces el operador θ a una matriz básica A escribiremos la matriz resultante como $\theta^N(A)$ (con $\theta^1(A) = A$). Si aplicamos N veces el operador φ, escribiremos la matriz resultante como $\varphi^N(A)$ (con $\varphi^1(A) = A$). Y de la misma manera, si aplicamos N veces el operador γ la matriz obtenida se denotará $\gamma^N(A)$ (con $\gamma^1(A) = A$).

2.6.2. Cálculo de testores típicos en una matriz sintética

Supongamos que A y B son dos matrices básicas, tal que los conjuntos $\psi^*(A)$ y $\psi^*(B)$ de todos sus testores típicos son conocidos. Entonces:

Sea $\mathcal{C}_A = \{x_1, \dots, x_n\}$ el conjunto de columnas de la matriz básica A, y $x_j \in \mathcal{C}_A$, llamaremos $[x_j]_N$ a las columnas de $\varphi^N(A)$ que son exactamente igual a x_j ($[x_j]_N = \{x_j, x_{j+N}, \dots, x_{j+(N-1)n}\}$). Sea $S \subseteq \mathcal{C}_A$ con $S = \{x_{j_1}, \dots, x_{j_s}\}$, $[S]_N$ denotará la familia de todos los subconjuntos de columnas de $\varphi^N(A)$ que se obtienen reemplazando una o varias columnas de S con otra columna de la misma clase, es decir, $[S]_N = [x_{j_1}]_N \times \dots \times [x_{j_s}]_N$. Se puede verificar que $|[S]_N| = N^{|S|}$.

PROPOSICIÓN 2.46 *El conjunto de testores típicos de una matriz concatenada con-*

sigo misma N veces, es el conjunto de todas las clases de testores típicos en A.

$$\psi^*(\varphi^N(A)) = \{[T]_N \mid T \in \psi^*(A)\}$$

Demostración. Observemos que los elementos de $[T]_N$ son idénticos a $A_{|\tau}$, aunque no tengan el mismo orden, debido a que cada columna en T solo puede ser reemplazada por una de la misma clase. Por definición, estos son testores típicos.
Ahora supongamos que existe un testor típico $S = \{x_{j_1}, \ldots, x_{j_s}\}$ fuera de $\{[T]_N \mid T \in \psi^*(A)\}$. Por tanto, existe al menos una columna de S que no es parte de A. Si remplazamos todas las columnas en S con una de la misma clase, entonces tendríamos una contradicción porque la submatriz resultante debe determinar un testor típico, el cual está en A, por lo que S debe estar en $[T]_N$ □

PROPOSICIÓN 2.47 *El conjunto de testores típicos en la matriz $\theta^N(A, B)$ es exactamente la unión de todos los testores típicos en A y todos los testores típicos en B.*

$$\psi^*(\theta^N(A, B)) = \psi^*(A) \cup \psi^*(B)$$

Demostración. Observemos que si identificamos las columnas de A y B y eliminamos las filas repetidas en cada conjunto obtenemos las columnas de las matrices A yB originales, por tanto el conjunto de testores típicos en A y B se conserva. No existe ningún testor en $\theta(A, B)$ con columnas en ambas matrices, porque si las columnas seleccionadas son testores en A o en B estos serían superconjuntos de testores, y por tanto no son típicos (irreducibles).
Si las columnas seleccionadas no fueran testores entonces:
Sean S_A y S_B el conjunto de columnas que no son testores en A y B respectivamente. Sea a la fila de A con ceros en las columnas de S_A y b la fila de B con ceros en las columnas de S_B. Como la fila formada por a y b ($[ab]$) pertenece a $\theta^N(A, B)$, la submatriz formada por $S_A \cup S_B$ tiene una fila de ceros y no puede ser un testor.

□

PROPOSICIÓN 2.48 *El conjunto de testores típicos en $\gamma^N(A, B)$ es el conjunto de todos los testores típicos formados como la unión de un testor en A y uno en B.*

$$\psi^*(\gamma^N(A, B)) = \{T_A \cup T_B \mid T_A \in \psi^*(A) \wedge T_B \in \psi^*(B)\}$$

Demostración. Notemos que, dado que a ambas matrices se les ha añadido varias filas de ceros, es imposible formar un testor formado únicamente por columnas del la-

do de la matriz A o del lado de la matriz B. Es decir, cualquier testor en $\gamma^N(A,B)$ debe incluir columnas de ambos lados de la matriz. Sea T un testor típico en $\gamma^N(A,B)$, entonces las columnas del lado de A deberían haber sido un testor típico en A y las columnas del lado de B deberían haber sido un testor típico en B. □

Gracias a estos resultados se obtienen los siguientes corolarios:

COROLARIO 2.49

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

COROLARIO 2.50 $|\psi^*(\theta(A,B))| = |\psi^*(A)| + |\psi^*(B)|$

COROLARIO 2.51 $|\psi^*(\gamma(A,B))| = |\psi^*(A)||\psi^*(B)|$

EJEMPLO 2.52 *Sean*

$$A = \begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 & X_4 & X_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \qquad B = \begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Puede verificarse que para la matriz A que

$$\psi^*(A) = \{[X_1, X_3, X_4], [X_1, X_5], [X_2, X_3]\}$$

y para la matriz B se tiene

$$\psi^*(B)) = \{[X_1, X_3, X_4], [X_1, X_3, X_5], [X_2, X_6], [X_3, X_4, X_6], [X_3, X_5, X_6]\}$$

Ya que A tiene un testor de tamaño 3 y dos de tamaño 2, entonces usando el corolario 2.51

$$|\psi^*(\varphi^3(A))| = (3)^3 + 2(2)^2 = 35$$

Para la matriz B con tres testores de tamaño 3 y uno de tamaño 2 se tiene

$$|\psi^*(\varphi^3(A))| = 3(3)^3 + (2)^2 = 85$$

Si realizamos la operación $\theta(A,B)$ entonces del corolario 2.50 se obtiene que

$$|\psi^*(\theta(A,B))| = 8$$

y para la matriz $\gamma(A,B)$ por el corolario 2.49 tenemos que

$$|\psi^*(\gamma(A,B))| = 15$$

En el cuadro 2.3 podemos observar como aumenta con respecto a N el número de columnas, filas y testores al aplicar ciertas combinaciones de operadores en estas matrices.

Cuadro 2.3: Matrices $\varphi^N(A), \theta^N(\theta(A,B)), \gamma^N(B)$

	$\varphi^N(A)$			$\theta^N(\theta(A,B))$			$\gamma^N(B)$		
N	**Filas**	**Columnas**	**TT**	**Filas**	**Columnas**	**TT**	**Filas**	**Columnas**	**TT**
1	4	5	3	20	11	8	5	6	5
2	4	10	4	400	22	16	10	36	25
3	4	15	5	8000	33	24	15	216	125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	4	$5N$	$3N^3+N^2$	20^N	$11N$	$8N$	5N	$6N$	5^N

Capítulo 3

Incorporación del aprendizaje simbólico en LEX

El aprendizaje simbólico es una estrategia general para guiar y optimizar el proceso de búsqueda dentro de los algoritmos de transversales mínimos y testores típicos, utilizando el conocimiento adquirido previamente sobre su espacio de búsqueda. Inicialmente fue implementado en los algoritmos BR [27] y YYC [4] en [16]. Nuestro propósito es integrarlo en LEX [5] y evaluar su desempeño con respecto al algoritmo original, para ello se probará el algoritmo modificado usando la metodología de matrices sintéticas descrita en la sección 2.6. Las etapas cruciales para incorporar la estrategia son la partición del espacio de búsqueda, identificar el orden que sigue el algoritmo e identificar la aportación que puede realizar el aprendizaje.

3.1. Partición del espacio de búsqueda en LEX

En LEX la evaluación del candidato (paso 3 del algoritmo 1) es la clave para la clasificación de conjuntos de rasgos de una matriz básica a través de las proposiciones 2.26 y 2.27. El conocimiento adquirido de éstas será guardado como se indica en la regla 4 de la sección 2.5. Luego de identificar el conjunto dominante y el rasgo o vértice dominado, el par será guardado en una tabla de conocimiento K para el uso posterior del algoritmo.

Cuando en LEX existe un rasgo excluyente (proposiciones 2.26 y 2.27) significa que la combinación que está siendo revisada es un testor no típico o un conjunto incompatible. Para rellenar K es de interés el segundo caso.

Suponiendo que el índice de error de testor (definición 2.35) y el índice de error

típico (definición 2.36) son distintos de cero pueden suceder dos cosas:

1. Cuando un conjunto de rasgos cumple la proposición 2.26 significa que el candidato actual retira la tipicidad de un rasgo de la lista de rasgos analizada, es decir, sea el rasgo i y L el candidato y la lista actual revisados en LEX respectivamente, donde $L + i$ cumplen la proposición 2.26, existe un subconjunto $S \subseteq L$ que domina i (definición 2.40).

2. Aplicando un razonamiento similar a la proposición 2.27 se puede encontrar otro par incompatible. Cuando el candidato a evaluar encaja en este caso significa que este posee 1's en donde uno o varios rasgos de L ya tienen valores unitarios. Dicho de otra forma, existe un subconjunto $S \subseteq L + i$ que domina algún rasgo de $j \in L$, con $j \notin S$.

Brevemente, la evaluación para cualquier rasgo i con respecto a una lista L usando la regla 4 descrita en la sección 2.5 se puede resumir en la figura 3.1.

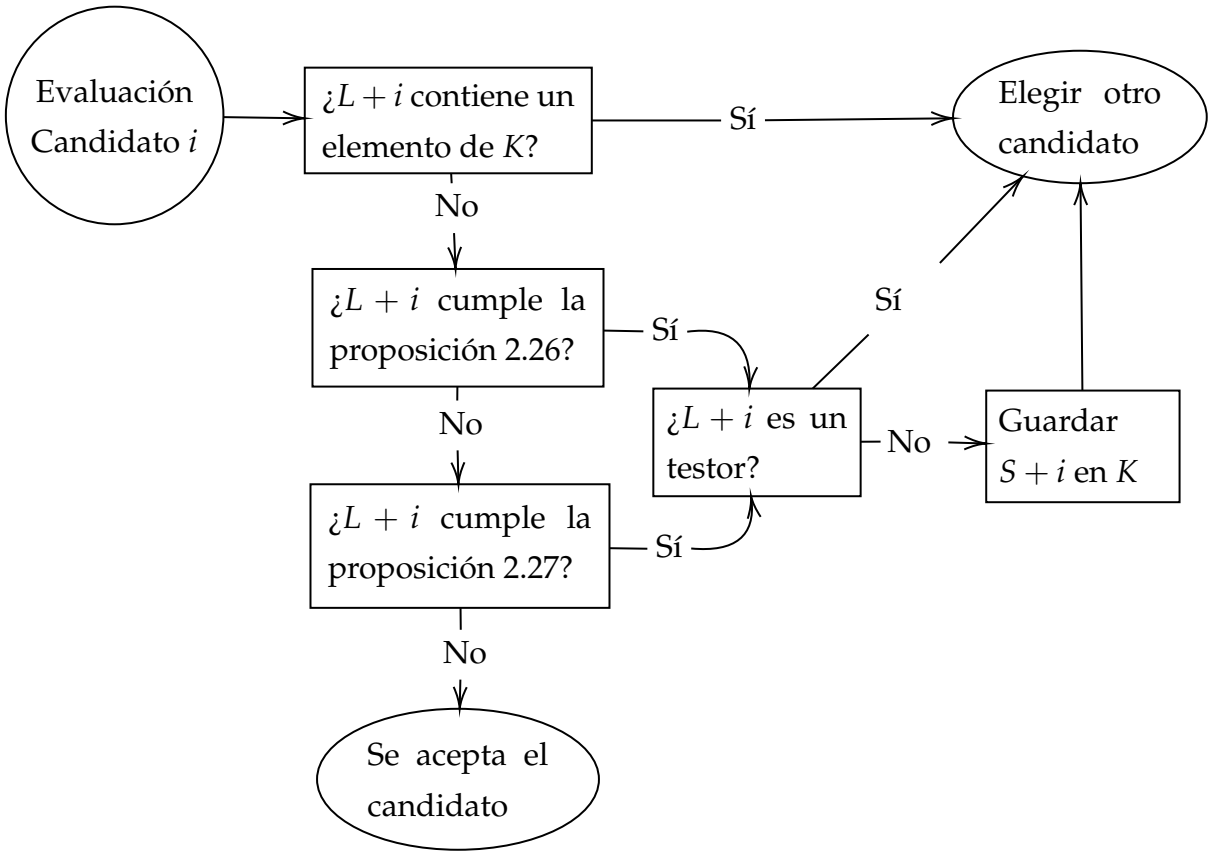


Figura 3.1: Evaluación del candidato i en LEX usando la tabla de conocimientos K

3.2. Algoritmo modificado

A partir de aquí, llamaremos al algoritmo LEX con aprendizaje simbólico LEX*, su pseudocódigo se presenta a continuación.

Algoritmo 2 LEX*

Entrada: *Matriz básica MB*

Salida: *El conjunto de todos los testores típicos de MB*

1: *Ordenación de MB*

a: *Encontrar la fila con cantidad mínima de 1's; de existir más de una, escoger cualquiera de ellas. Ponerla como primera fila en MB.*

b: *Ordenar las columnas poniendo indistintamente como primeras las que tengan un 1 en la primera fila.*

2: *Inicialización*

a: $l = [\,]$, $K = [\,]$ *(registro de incompatibilidades),* $X = X_1$ *(X es el primer candidato a elemento de l).*

3: *Evaluación del candidato X*

a: **Si** $l = [\,]$ *y la columna correspondiente a X tiene cero en la primera fila de MB* **entonces** *FIN.*

b: **Si** $l + [X] \supseteq k$*, donde k pertenece a la lista de incompatibles K* **entonces** *ir a 4*

c: **Si** *X es excluyente con l (proposiciones 2.26 y 2.27)* **entonces**

Si *para toda fila i,* $cr_i^{l+[X]} > 0$ **entonces** *ir a 4*

Caso contrario

I. **Si** *se cumple la proposición 2.26* **entonces** *sea s la sublista más pequeña de l tal que* $X_{j_k} \in s$ *y* $|F_X^s| = 1$*, añadir* $s + [X]$ *a K*

II. **Si** *se cumple la proposición 2.27* **entonces** *sea s la sublista más pequeña de l tal que* $|F_X^s| = 0$*, añadir* $s + [X]$ *a K.*

III. *Ir a 4.*

d: **Si** *para toda fila i de MB* $cr_i^{l+[X]} > 0$ **entonces** *guardar* $l + [X]$ *es un TT, ir al paso 4*

e: **Si** $X = X_n$ **entonces** *ir al paso 4 (Corolario 2.30)*

f: *Hacer* $l = l + [X]$*, se acepta el candidato. Actualizar los valores de* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para todos los elementos* X_t *en l.*

4: *Selección del nuevo candidato*

a: **Si** $X \neq X_n$ **entonces** *sea j el índice de X en MB, hacer* $X = X_{j+1}$ *e ir al paso 3* ;

b: **Si** $l = [\,]$ **entonces** FIN.;

c: **Si** $l + [X]$ *fue un TT o X no fue excluyente con l* **entonces**

Si *existe el hueco* X_p *de* $l + [X]$ **entonces**

I. *Hacer* $X = X_{p+1}$

II. *Eliminar de l todos los elementos desde* X_p *hasta* X_{j_s}*(proposición 2.28)*

III. *Actualizar* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para cada* X_t *de l. Ir al paso 3.*

Caso contrario *no existe* X_p*, FIN (corolario 2.29)*;

d: *Hacer* $X = X_{j_s}$*, donde* X_{j_s} *es el último rasgo de l y* $l = l \setminus [X_{j_s}]$*. Actualizar* cr_i^l *para todas las filas de MB y* $F_{X_t}^l$ *para cada* X_t *de l.*

e: *Retornar al paso 4.*

Ahora que la estrategia ha sido añadida, nos resta evaluar el desempeño del nuevo algoritmo usando un conjunto de matrices diseñadas. Pero antes de someterlo a pruebas más complejas, veamos un ejemplo sencillo de su funcionamiento.

EJEMPLO 3.1 *Consideremos la matriz del ejemplo 2.22. En el siguiente cuadro se muestra las combinaciones revisadas por LEX, y en azul aquella que LEX* ha omitido debido a la estrategia.*

Combinaciones Revisadas		
1	134	2
12	135	23
123	1	**24**
124	14	25
125	145	
1	1	
13	15	

El conjunto x_2, x_4 *se encuentra en la tabla de incompatibilidades K. Este conocimiento ha sido obtenido al revisar la combinación* x_1, x_2, x_4*, la cual cumple la proposición 2.27.*

Es importante entender que tanto el aprendizaje obtenido como su uso dependen

de la taxonomía del algoritmo y de la matriz básica que recibe. La estrategia no es responsable de cuánto conocimiento se consiga o si este es aprovechado en su totalidad.

3.3. Pruebas e interpretación de resultados

Finalmente podemos someter a prueba al nuevo algoritmo. La atención se centrará en el total de conjuntos que evalúa el algoritmo original frente al total de conjuntos que evalúa el nuevo algoritmo o visto de otra forma, qué proporción de conjuntos se logran omitir; consideramos este criterio pertinente porque a través de él se puede estimar la ventaja del aprendizaje simbólico independientemente de las características del equipo y lenguaje de programación que se utilice, y además porque es el motivo principal del uso de la estrategia es reducir el número de conjuntos evaluados.

Se han definido los siguientes términos con la finalidad de comprender y examinar los resultados más fácilmente:

1. Combinaciones Analizadas: el número total de conjuntos que genera el algoritmo.

2. Combinaciones Revisadas: el número total de conjuntos que el algoritmo ha considerado como candidatos, es decir aquellas que no se han descartado por el aprendizaje simbólico.

3. Combinaciones Descartadas: el número total de conjuntos que se ha omitido revisar debido al aprendizaje simbólico.

4. Conocimiento Adquirido: el número total de conjuntos que se registran como incompatibilidades en la tabla de conocimientos K.

Notemos que las combinaciones analizadas son los conjuntos que LEX original debe revisar, de esta forma tenemos una medida de referencia para el beneficio que la estrategia aporta al algoritmo. Si bien un agente influyente en el total de combinaciones descartadas es la taxonomía del algoritmo, la información que se obtiene y cuán útil resulte depende también de la matriz básica que se recibe, como ya se ha mencionado. De hecho, puede ser interesante plantearse un estudio más profundo desde este punto de vista y conocer qué tipo de matrices resultan más favorecidas al utilizar el algoritmo LEX*.

3.3.1. Diseño de matrices de prueba

Cuando hablamos de diseñar matrices de prueba nos referimos a crear matrices con ciertas características apoyándonos en los operadores descritos en la sección 2.6. Nuestro objetivo es incluir matrices con gran densidad de testores típicos y matrices de grandes dimensiones, pero escasa cantidad de testores típicos, de tal manera que se tenga un criterio base que proporcione conclusiones adecuadas y sirva de guía sobre cómo maneja diferentes tipos de matrices el nuevo algoritmo. Para el diseño utilizaremos la matriz $\mathbf{H_{2c}}$ que aparece en [7] y contiene un testor típico de cardinalidad 1 y tres de cardinalidad 2, las matrices $\mathbf{M_1}$ y $\mathbf{M_2}$ que aparecen en [3]; $\mathbf{M_1}$ contiene un testor irreducible de cardinalidad 1, dos de cardinalidad 2 y uno de cardinalidad 3, y $\mathbf{M_2}$ contiene cuatro testores irreducibles de cardinalidad 3. Además, incluiremos la matriz **Mínima** que aparece en [30] y contiene un testor típico de cardinalidad 1. Estas matrices se encuentran en el apéndice A.

A partir de ellas obtendremos tres matrices con diferentes características de la siguiente manera:

- Una matriz cuyo número de filas sea similar al número de columnas, es decir que se asemeje a una matriz cuadrada. Simplemente se tomará la matriz M_2 de dimensión 4×5. Renombraremos a esta matriz A.

- Una matriz tal que el número de filas es mayor al número de columnas, para obtenerla se usará la matriz M_1 y H_{2c} y se les aplicará el operador θ, es decir que se efectuará la operación $\theta(H_{2c}, M_1)$ y se obtendrá una matriz de dimensión 12×9. La nombraremos matriz B.

- Finalmente, se creará una matriz cuyo número de columnas sea mayor al número de filas. Hemos decidido utilizar la matriz Mínima y efectuar la operación $\varphi(\text{Minima}, 2)$ que produce una matriz de dimensión 2×10. Esta matriz se denominará matriz C

Ahora aplicaremos los tres operadores θ, φ y γ para diferentes valores de N.

Cuadro 3.1: Diseño de matrices de prueba

Matriz	Operador	N	Columnas	Filas	Testores Típicos
A	θ	$1,2,3,4,5$	$4N$	5^N	$4N$
	φ	$1,2,3,4,5$	$4N$	5	$4N^3$
	γ	$1,2,3,4,5,6,7,8,9,10$	$4N$	$5N$	4^N
B	θ	$1,2,3,4,5,6,7,8,9,10$	$10N$	2^N	$4N$
	φ	$1,2,3,4,5,6,7,8,9,10$	$10N$	2	$4N^2$
	γ	$1,2,3,4,5,6,7,8,9,10$	$10N$	$2N$	4^N
C	θ	$1,2,3,4$	$9N$	12^N	$8N$
	φ	$1,2,3,4,5$	$9N$	12	$5N^2+2N+N^3$
	γ	$1,2,3,4,5$	$9N$	$12N$	8^N

En el cuadro 3.1 se observa que el operador θ generará matrices de grandes dimensiones y poca cantidad de testores típicos mientras que el operador γ nos ofrece matrices con una gran cantidad de testores típicos, cumpliendo con la intención planteada inicialmente. Se ha decidido considerar este enfoque porque en la práctica matrices que poseen alguna de estas propiedades son muy comunes. Por otro lado, el operador θ permite crear matrices con un gran número de filas y el operador φ matrices con gran cantidad de columnas; a base de este criterio existe la posibilidad de que se alcance alguna otra conclusión relevante.

3.3.2. Presentación de resultados

Para analizar los resultados presentaremos cuadros que contienen el número de filas y columnas, el número de testores típicos, la cantidad de conocimiento adquirido (referente al punto 4 de los términos definidos al inicio de esta sección), y la proporción de conjuntos descartados para cada operador y matriz. Además, se incluirán dos gráficos, el de la izquierda corresponderá al tiempo de ejecución del algoritmo LEX versus el tiempo de LEX*, a la derecha se encontrarán las combinaciones analizadas frente a las combinaciones revisadas, o visto de otra forma las combinaciones que LEX y las combinaciones que LEX* han revisado en su totalidad respectivamente. Se ha incluido el tiempo de ejecución para ilustrar el beneficio de la estrategia incorporada en el algoritmo, pero no se obtendrán conclusiones a partir de este, puesto que varía en dependencia del lenguaje de programación y las características del equipo que se utilice. En este proyecto se ha utilizado una Deep Lear-

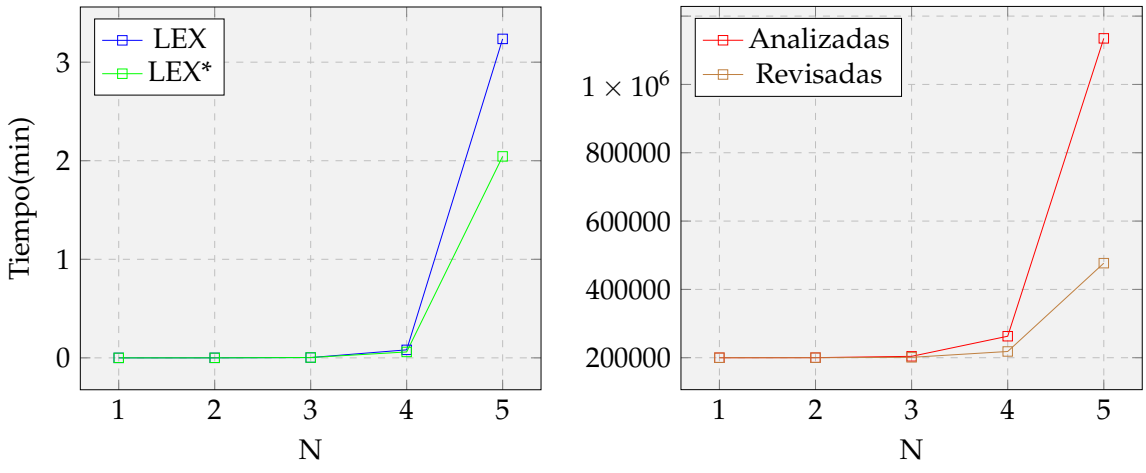
ning Work Station, específicamente una Nvidia DGX, impulsada por cuatro GPU NVIDIA Tesla V100 que proporcionan un rendimiento equivalente a cientos de servidores tradicionales. El algoritmo está programado en MATLAB versión R2019b, el código se encuentra en el apéndice B.

El conocimiento adquirido y el porcentaje de combinaciones descartadas son de especial interés para valorar el impacto del aprendizaje simbólico dentro de cada matriz, y de ese modo obtener un indicio de en cuáles matrices la estrategia de aprendizaje simbólico en LEX funciona mejor.

Comenzaremos con la matriz A. En el cuadro 3.2 vemos que para el operador θ a partir de $N = 2$ entre un 20 % y 30 % de conjuntos generados fueron descartados.

Cuadro 3.2: Resultados $\theta^N(A)$

N	**1**	**2**	**3**	**4**	**5**
Filas	4	16	64	256	1024
Columnas	5	10	15	20	25
TT	4	8	12	16	20
Conocimiento Adquirido	2	6	9	12	15
Combinaciones Descartadas (%)	0.00 %	23.24 %	27.83 %	29.05 %	29.61 %



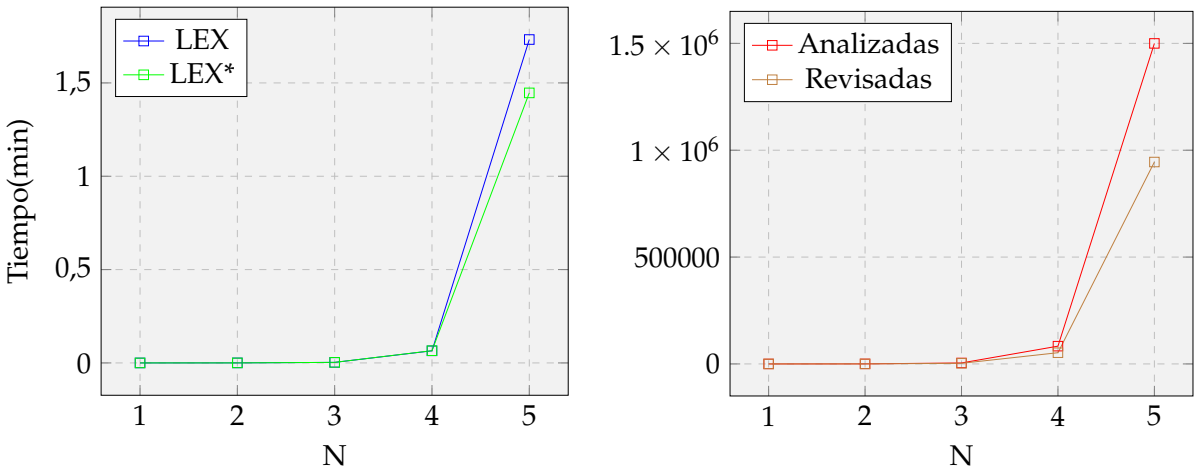
(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.2: Resultados para operador θ matriz A

En el operador γ aplicado a la matriz A se han descartado hasta un 37 % de combinaciones. En el cuadro 3.3 también notamos que se ha guardado mayor cantidad de conjuntos en conocimiento adquirido que en 3.2.

Cuadro 3.3: Resultados $\gamma^N(A)$

N	1	2	3	4	5
Filas	4	8	12	16	20
Columnas	5	10	15	20	25
TT	4	16	64	256	1024
Conocimiento Adquirido	2	7	11	15	19
Combinaciones Descartadas (%)	0.00 %	31.02 %	36.13 %	36.87 %	37.00 %



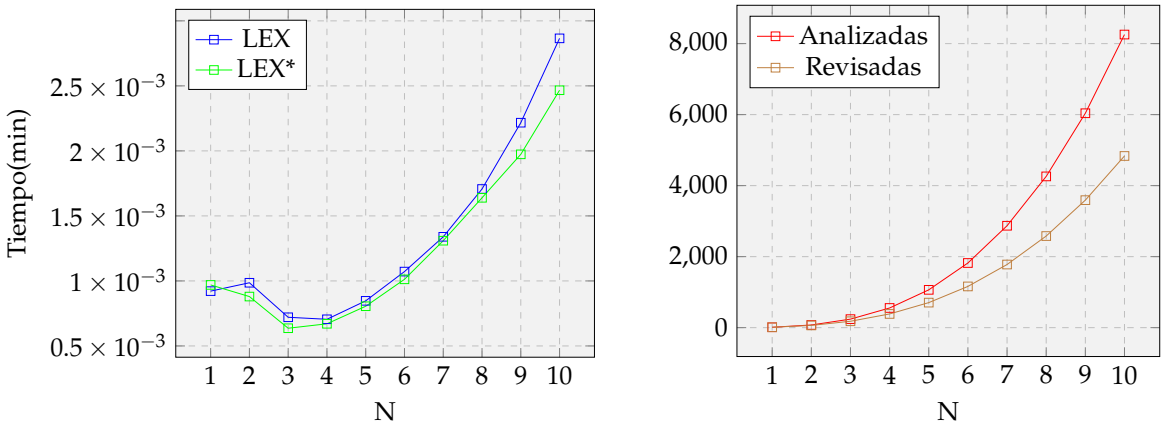
(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.3: Resultados para operador γ matriz A

Para el operador φ se incrementa el número de expresiones de conocimiento adquirido. Hasta $N = 10$ se ha logrado descartar más de un 40 % de conjuntos. Si comparamos con los operadores anteriores hasta $N = 5$ vemos que se descartan más combinaciones que en θ pero menos que γ.

Cuadro 3.4: Resultados $\varphi^N(A)$

N	1	2	3	4	5	6	7	8	9	10
Filas	4	4	4	4	4	4	4	4	4	4
Columnas	5	10	15	20	25	30	35	40	45	50
TT	4	32	108	256	500	864	1372	2048	2916	4000
Conocimiento Adquirido	2	13	33	62	100	147	203	268	342	425
Combinaciones Descartadas (%)	0.0 %	16.0 %	25.0 %	30.3 %	33.8 %	36.2 %	38.0 %	39.4 %	40.5 %	41.4 %



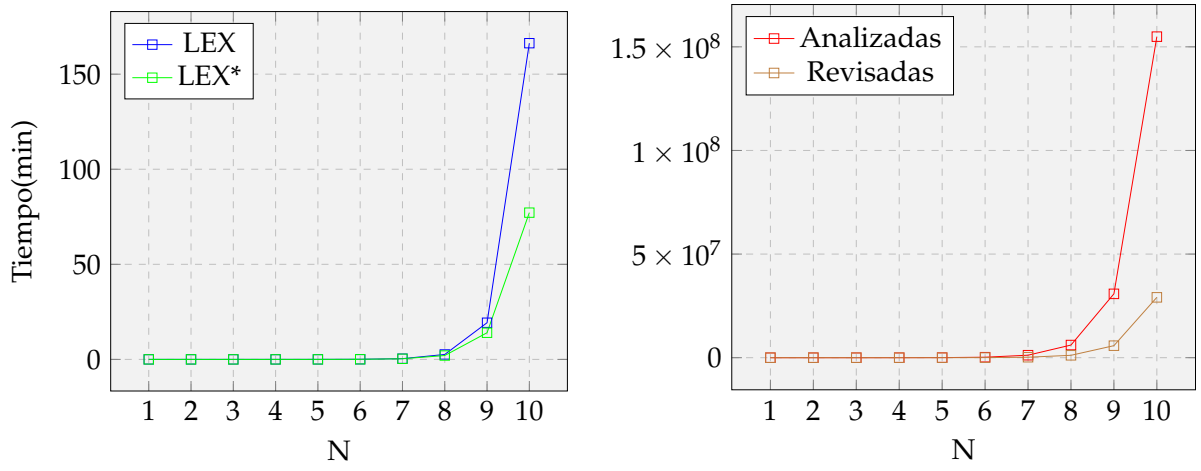
(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.4: Resultados para operador φ matriz A

A continuación se muestran los resultados de la matriz B. En los cuadros 3.5 y 3.6 se observa que el porcentaje de combinaciones omitidas, gracias a la estrategia llega a sobrepasar el 80 %. Pese a que en θ y γ se ha adquirido la misma cantidad de conocimiento, la proporción de conjuntos descartados en γ es un poco mayor.

Cuadro 3.5: Resultados $\theta^N(B)$

N	1	2	3	4	5	6	7	8	9	10
Filas	2	4	8	16	32	64	128	256	512	1024
Columnas	10	20	30	40	50	60	70	80	90	100
TT	4	16	36	64	100	144	196	256	324	400
Conocimiento Adquirido	7	16	24	32	40	48	56	64	72	80
Combinaciones Descartadas (%)	31.58 %	68.63 %	77.27 %	79.67 %	80.50 %	80.68 %	80.98 %	81.11 %	81.17 %	81.20 %

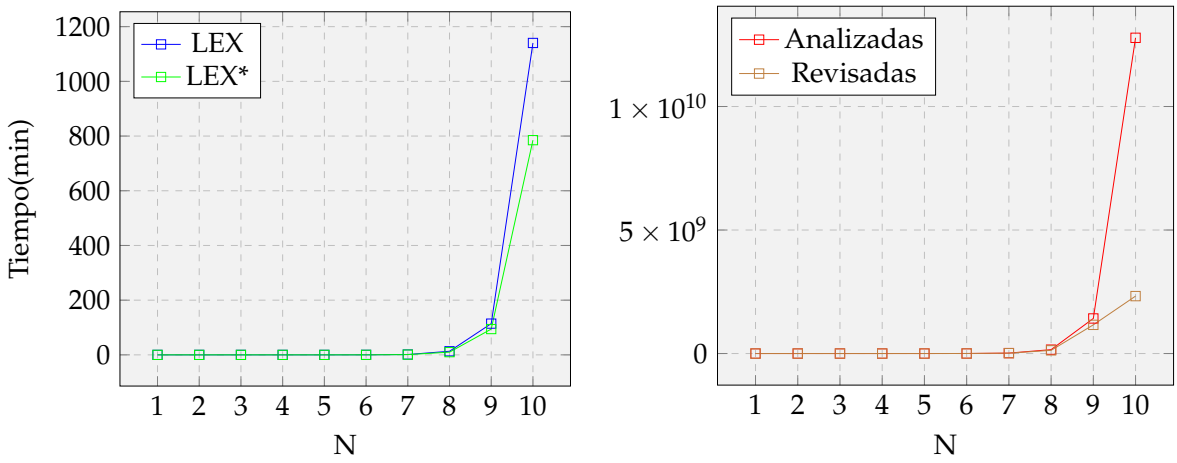


(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.5: Resultados para operador θ matriz B

Cuadro 3.6: Resultados $\gamma^N(B)$

N	1	2	3	4	5	6	7	8	9	10
Filas	2	4	6	8	10	12	14	16	18	20
Columnas	10	20	30	40	50	60	70	80	90	100
Testores típicos	4	16	64	256	1024	4096	16384	65536	262144	1048576
Conocimiento Adquirido	7	16	24	32	40	48	56	64	72	80
Combinaciones Descartadas (%)	31.58 %	71.19 %	79.19 %	80.98 %	81.49 %	81.68 %	81.76 %	81.79 %	81.81 %	81.81 %



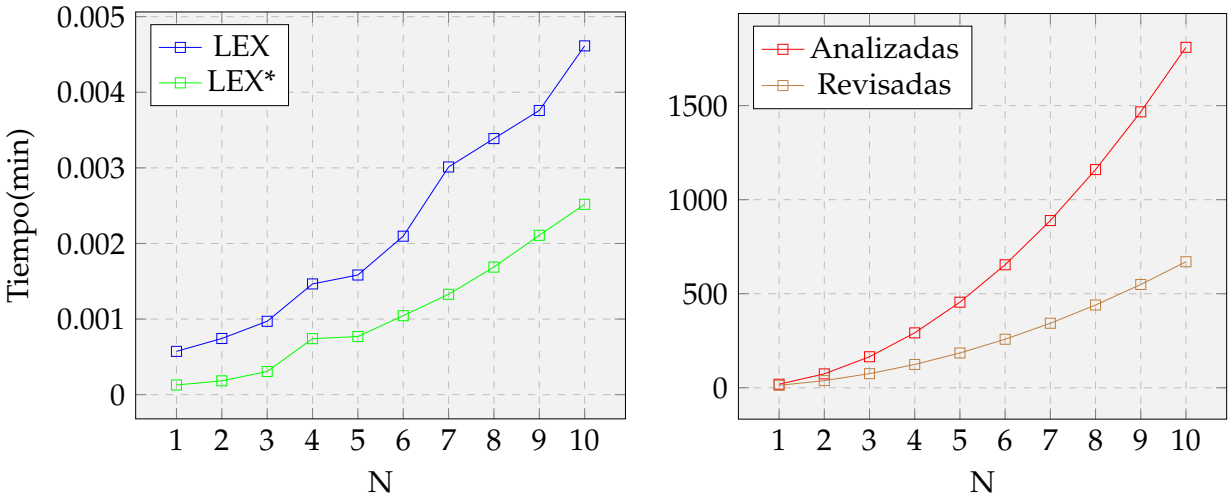
(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.6: Resultados para operador γ matriz B

En el cuadro 3.6 se evidencia que nuevamente para φ existe mayor número de conjuntos en conocimiento adquirido, se excluyen menos combinaciones que en los casos anteriores, pero seguimos obteniendo un buen porcentaje de conjuntos descartados.

Cuadro 3.7: Resultados $\varphi^N(B)$

N	1	2	3	4	5	6	7	8	9	10
Filas	2	2	2	2	2	2	2	2	2	2
Columnas	10	20	30	40	50	60	70	80	90	100
TT	4	16	36	64	100	144	196	256	324	400
Conocimiento Adquirido	7	18	33	52	75	102	133	168	207	250
Combinaciones Descartadas (%)	31.58 %	48.65 %	54.55 %	57.53 %	59.34 %	60.55 %	61.42 %	62.07 %	62.58 %	62.98 %



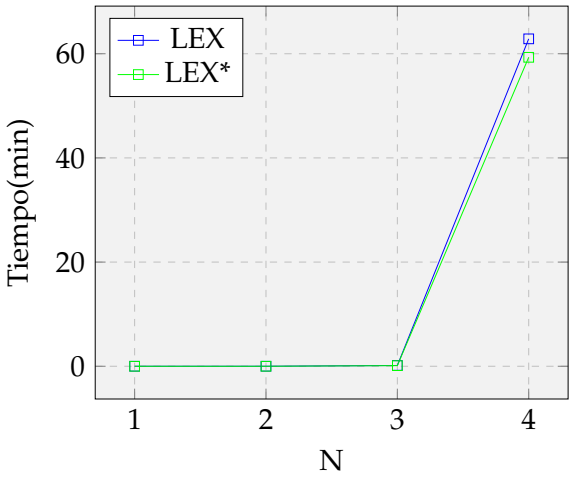
(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.7: Resultados para operador φ matriz B

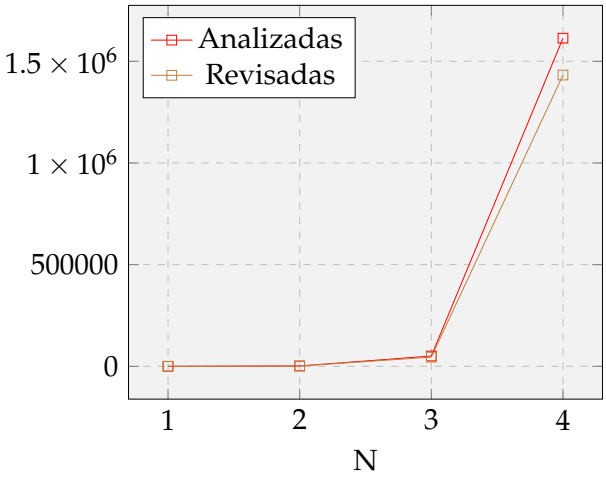
En los cuadros 3.8, 3.9 y 3.10 observamos que usando cualquiera de los tres operadores si incorporamos la estrategia de aprendizaje simbólico se consigue reducir el número de candidatos a TT revisados. Al igual que en las matrices A y B, el mayor porcentaje de conjuntos descartados se obtiene para el operador γ.

Cuadro 3.8: Resultados $\theta^N(C)$

N	**1**	**2**	**3**	**4**
Filas	12	144	1728	20736
Columnas	9	18	27	36
TT	8	16	24	32
Conocimiento Adquirido	1	2	3	4
Combinaciones Descartadas (%)	6.12 %	10.09 %	10.99 %	11.23 %



(a) Tiempo para LEX y LEX*

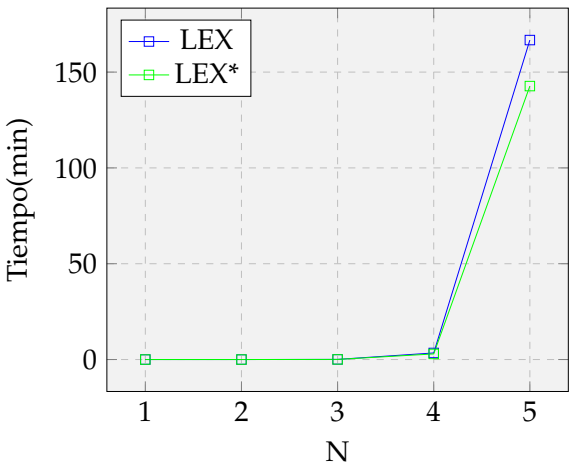


(b) Combinaciones Analizadas y Revisadas

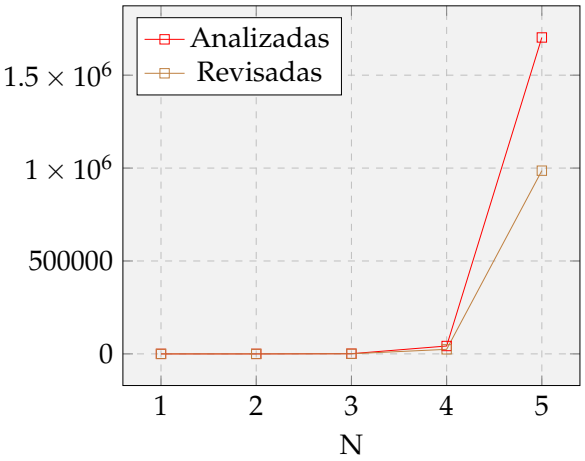
Figura 3.8: Resultados para operador θ matriz C

Cuadro 3.9: Resultados $\gamma^N(C)$

N	**1**	**2**	**3**	**4**	**5**
Filas	12	24	36	48	60
Columnas	9	18	27	36	45
TT	8	64	512	4096	32768
Conocimiento Adquirido	1	78	117	156	195
Combinaciones Descartadas (%)	6.12 %	50.27 %	57.05 %	57.78 %	57.91 %



(a) Tiempo para LEX y LEX*

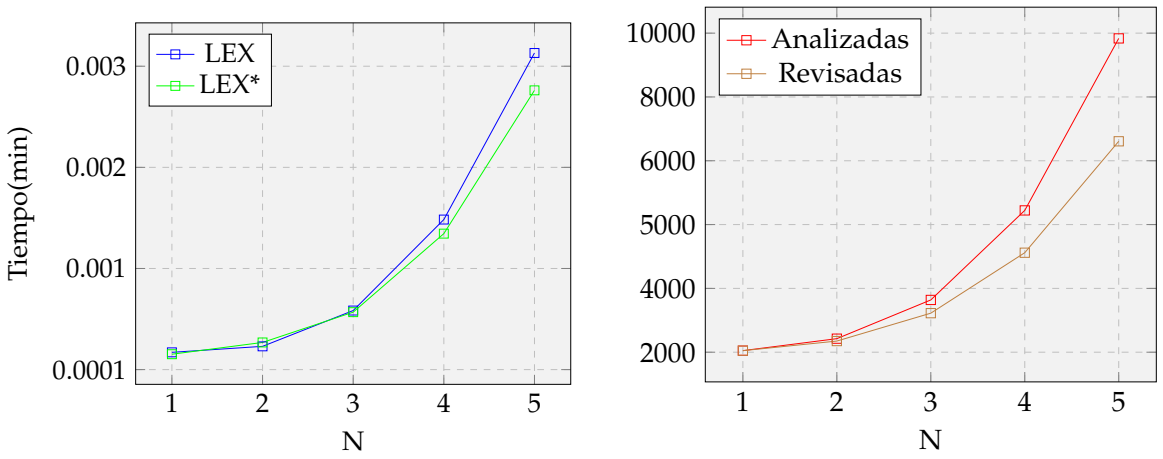


(b) Combinaciones Analizadas y Revisadas

Figura 3.9: Resultados para operador γ matriz B

Cuadro 3.10: Resultados $\varphi^N(C)$

N	1	2	3	4	5
Filas	12	12	12	12	12
Columnas	9	18	27	36	45
TT	8	32	78	152	260
Conocimiento Adquirido	1	11	30	58	95
Combinaciones Descartadas (%)	6.12 %	20.52 %	27.53 %	31.64 %	34.32 %



(a) Tiempo para LEX y LEX* **(b)** Combinaciones Analizadas y Revisadas

Figura 3.10: Resultados para operador φ matriz C

En la operación θ se guarda menos conjuntos en la tabla de conocimiento en comparación con las matrices resultantes de φ y γ.

Capítulo 4

Conclusiones

Cuando se habla de eficiencia, se suele abordar un estudio del comportamiento de los algoritmos a través de la teoría de complejidad computacional. Sin embargo, el efecto *No Free Lunch* para los algoritmos de cálculo de transversales mínimos, y a su vez el cálculo de testores típicos por la equivalencia presentada en los teoremas 2.16 y 2.17, nos sitúa en una posición más complicada.

El teorema de No Free Lunch (NFLT) [33, 22] establece que una estrategia de optimización universal de propósito general es imposible.

Para nuestro caso en particular, las funciones de coste de los algoritmos LEX y LEX* resultarán diferentes debido a las modificaciones, pero el efecto No Free Lunch afirma que no hay garantías de que una función de coste desarrolle mejor que otra para todas las matrices básicas. Por lo que abordar la evaluación del nuevo algoritmo desde este punto de vista no sería recomendable. En otras palabras, no existe un algoritmo bueno y un algoritmo malo en términos generales. Esta es la razón por la cual nuestros experimentos consideran diferentes tipos de matrices y se obtienen conclusiones para cada grupo.

Una implicación importante del NFLT es que, a falta de hipótesis previas sobre la función de coste analizada, todos los algoritmos de búsqueda tienen el mismo rendimiento esperado. Es decir, que todos son igualmente válidos y por ello siguen vigentes. Un estudio más detallado de la complejidad del tiempo y espacio de una variedad de algoritmos para la determinación de transversales mínimos es desarrollado en [20], donde se muestra que estos tienen una complejidad no polinomial. Además, se explica que no ha sido posible encontrar un algoritmo para esta tarea en tiempo polinomial. Dado que el problema del cálculo de transversales mínimos y testores típicos es equivalente [2, 3], sucede lo mismo con estos algoritmos, incluidos

LEX y LEX*.

Sin embargo, cuando la estrategia de aprendizaje simbólico fue presentada, se habla de la posibilidad de mejorar la eficiencia de un algoritmo basándose en la capacidad de evitar la mayor cantidad de pruebas innecesarias posible. Cabe mencionar que existe una relación costo beneficio, y como se explica en el artículo [16], esta ganancia merece la pena para hipergrafos grandes. Si la instancia del problema es pequeña, la estrategia sí optimizará el proceso de búsqueda, pero existe un costo computacional adicional.

El beneficio real es evidente cuando la instancia del problema es grande, pues la cantidad de combinaciones que se omiten permite excluir las operaciones siguientes que son las que originalmente evalúan cada una de estas combinaciones, y el tiempo de ejecución adicional que requiere la estrategia en comparación es menor. Como no podemos analizar explícitamente la función de coste por el NFLT, al igual que en [16] se muestra la cantidad de combinaciones revisadas de cada uno de los algoritmos y este es el criterio de evaluación para los distintos grupos de matrices. Al fin y al cabo, el hecho de que el aprendizaje simbólico sea relevante solo en instancias grandes es suficiente, pues estas son nuestro objetivo.

Como ya se ha mencionado, el tiempo de ejecución obtenido en los experimentos es reportado, pero este es afectado por factores como el lenguaje de programación y el equipo utilizado. De modo que es presentado porque son resultado de los experimentos, pero no se toma como referencia para este estudio. No obstante, podrían realizarse varias pruebas y utilizar un soporte estadístico para el tiempo de ejecución si se desea reducir su sesgo y evaluarlo.

En las pruebas de la sección 3.3 se observa que se registran incompatibilidades que permiten excluir conjuntos innecesarios, mejorando así el proceso de búsqueda de LEX. Sin embargo, no nos hemos limitado a presentar evidencia de ello, sino también a estudiar cómo actúa la estrategia dentro de LEX a través de estos resultados. Si bien como ya se ha explicado, no existe un procedimiento que nos permita establecer qué algoritmo es mejor de forma global, se puede plantear ciertos parámetros que se consideren interesantes de estudiar y nos lleven a determinar ciertos comportamientos.

En esta ocasión nos hemos enfocado en el diseño de matrices con gran cantidad de testores y matrices con grandes dimensiones y una cantidad escasa de testores. Adicionalmente, hemos variado la estructura de las matrices base con respecto al número de filas y columnas que estas poseen.

La tabla de conocimientos es llenada y consultada según el método de búsqueda empleado por el huésped, en este caso el orden lexicográfico que maneja el algoritmo ha permitido que la cantidad de conjuntos aprendidos sea pequeña en comparación a la cantidad de revisiones evitadas producidas, indicando un buen aprovechamiento de la tabla de conocimientos.

Una observación interesante es que todas las matrices a las que se les ha aplicado el operador γ reportan un mayor porcentaje de conjuntos descartados. No sería prudente concluir que las matrices que provengan de alguna operación en específico son manejadas de manera más eficiente por LEX*, ya que en la práctica no todas las matrices provienen de alguna de las operaciones matriciales usadas en los experimentos, pero notemos que el operador γ produce matrices con gran cantidad de testores y por tanto podemos intuir que, según los resultados obtenidos en este trabajo, el aprendizaje simbólico aplicado al algoritmo LEX reduce en mayor proporción el número de combinaciones revisadas en matrices con grandes cantidades de testores en comparación con las matrices de grandes densidades y pocos testores, las cuales se han obtenido al utilizar el operador θ.

En cambio, en las matrices operadas con θ se aprecia que la cantidad de conjuntos registrados en la tabla de conocimientos es menor o igual a la cantidad de conjuntos registrados en el resto de los operadores, esto puede deberse a su gran dimensión o más específicamente al extenso número de filas por el que se caracterizan estas matrices. En las matrices φ cuya particularidad es un número superior de columnas sobre filas, LEX* ha aprendido más que en las demás matrices.

Por otra parte, las matrices provenientes de B, la cual contiene poca cantidad de filas y gran cantidad de columnas, ha mostrado ser la más beneficiada con respecto a las matrices provenientes de otras matrices base, pues la cantidad de revisiones ha bajado mucho más; las matrices provenientes de A, cuya estructura es similar a una matriz cuadrada, las presiden con un porcentaje de conjuntos descartados más pequeño, mientras que aquellas creadas a partir de la matriz C que contiene más cantidad de filas que columnas, son las que presentaron el menor porcentaje. Es probable que a LEX* le sea posible descartar mayor número de conjuntos en matrices cuya cantidad de columnas es superior en relación con sus filas, pero asimismo en estas parece almacenar más conocimiento.

Los resultados obtenidos en este experimento pueden servir de guía para futuros trabajos que aporten al desarrollo teórico en el campo de testores típicos y, gracias a su convergencia teórica entre conceptos, en el campo de transversales mínimos.

Algunas opciones que se pueden plantear son las siguientes:

- Contrastar el comportamiento de otros algoritmos con aprendizaje simbólico integrado y de ser factible encontrar relaciones entre la estructura de la matriz y la habilidad de la estrategia para disminuir revisiones.
- De igual manera, se puede verificar si existen ciertos métodos de búsqueda dentro de los algoritmos que resultan más eficientes que otros cuando se les proporciona el apoyo del aprendizaje simbólico.
- Además de variar las dimensiones de la matriz, se puede estudiar el desempeño de LEX* frente a otros parámetros como la proporción de entradas con valores unitarios o nulos de las matrices, o analizar su comportamiento en matrices en escalera descendente de las cuales se conoce que son difíciles de manejar por su cantidad exorbitante de testores típicos.
- Comparar la cantidad de incompatibilidades aprendidas que se almacena en diferentes tipos de algoritmos, así como el aprovechamiento de este también puede ser un punto interesante.

Finalmente, además de proveer una descripción más minuciosa del funcionamiento del aprendizaje simbólico, así como una demostración de su impacto, este proyecto impulsado por la motivación de contribuir con el desarrollo teórico de los campos involucrados, pone a disposición el algoritmo LEX* para aplicaciones en varios problemas capaces de ser modelados a través de testores típicos o transversales mínimos cuando pueda ser oportuno.

Apéndice A

Matrices utilizadas para el diseño de matrices de prueba

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$H_{2c} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \text{Mínima} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Apéndice B

Código

B.1. Funciones complementarias

```
function  y = FXt(A,L,t,n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%Autora: Ingrid Guevara %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FXt
%Input:matriz basica A,lista de rasgos L,rasgo t
%Output:calcula el conjunto de filas tipicas de el rasgo t
%respecto a L y la cantidad n de elementos de L
    T=1:n;
    B=sum(A(:,L(L~=t)),2);
    y=T(B<A(:,t));
end
```

```
function I= Incompatibles11(A,L,i,r,l)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%Autora: Ingrid Guevara %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Incompatibles11 encuentra la sublista de L incompatible
%con i en el sentido de la 1era prop
%Input:matriz basica A,lista de rasgos L,rasgo i,
%rasgo r perteneciente a L,cantidad  l de rasgos de L
%Output:I sublista mas corta de L que es incompatible con i
if l>1
```

```
        M=A(logical(A(:,r)),:);
        T=L(L~=r);
        S=M(~logical(M(:,i)),T);
        I=zeros(1,l+1);
        I([l,l+1])=[r,i];
            if size(S,1)>1
                [~,idx] = sort(sum(S),'descend');
            else
                [~,idx] = sort(S,'descend');
            end
        T=T(idx);
        n=1;
        while any(sum(M(:,I(I~=0)),2)<2)
            I(n)=T(n);
            n=n+1;
        end
          I=I(I~=0);
          I=sort(I);
else
            I=[L,i];
end

end
```

```
function I = Incompatibles22(A,L,i,S1,l)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%Autora: Ingrid Guevara %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Incompatibles22 encuentra la sublista de L que es
%incompatible con i en el sentido de la 2da prop
%Input:matriz basica A,lista de rasgos L,i rasgo que quita
%la tipicidad de L,S1 conjunto de filas de A que tienen
%valor unitario en la columna i
%Output:I sublista mas corta de L que es incompatible con i
I=zeros(1,l+1);
I(l+1)=i;
if any(A(:,i))
```

```
14      M=A(S1,:);
15      if numel(S1)>1
16          [~,idx] = sort(sum(M(:,L)),'descend');
17      else
18          [~,idx] = sort(M(:,L),'descend');
19      end
20      n=1;
21      T=L(idx);
22    while any(sum(M(:,I(I~=0)),2)<2)
23          I(n)=T(n);
24          n=n+1;
25    end
26    I=sort(I);
27 end
28 I=I(I~=0);
29 end
```

B.2. LEX*

```
1  function [TT,tt,in,conj,cd,ca,ce,tmpej] = lexas( MB )
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%%%Autora: Ingrid Guevara %%%%%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  % lexas LEX utilizando Aprendizaje Simbolico
7  %    Input:Matriz Basica MB
8  %    Output:
9  %    TT: Testores Tipicos
10 %    tt: Cantidad de Testores Tipicos
11 %    in: Cantidad de Incompatibilidades
12 %    cd: Cantidad de Conjuntos Descartados
13 %    ca: Cantidad de Conjuntos Analizados
14 %    tmpej: Tiempo de Ejecucion
15
16 %% 1. Ordenar
17     tstart = now();
```

```
18      A=ordenar(MB);
19  %% 2. Inicializacion
20      stop=0; %criterio de parada
21      stop1=0; %criterio de parada
22      [y,n]=size(A);
23      TToE=0; %1 si L+i es TT o no excluyente,0 cc
24      L=[]; %lista de rasgos
25      i=1; %candidato actual
26      TT={}; %lista de testores tipicos
27      I=cell(n,n,n); %lista de incompatibilidades
28      M=zeros(n,n);
29      N=zeros(n,n,n);
30      F={}; %conjuntos de filas tipicas
31      l=0;  %cantidad de elementos lista revisada
32      cr=zeros(y,1);%vector conteo de rasgos con valor unitario
            por fila
33      %------ variables de apoyo
34      c=zeros(y,1);
35      indi=zeros(n,n);
36      a=inf;
37      %-------------------------
38      tt=0;
39      in=0;
40      conj=0;
41      cd=0;
42      ca=0;
43      for s=1:n
44          for j=1:n
45              for k=1:n
46                  I{s,j,k} = zeros(n,1);
47              end
48          end
49      end
50      tic;
51    while(stop==0)
52      %% 3. Evaluacion del candidato X
```

```
53          if  l==0 && A(1,i)==0
54              break
55          end
56      while(stop1==0)
57       ca=ca+1;
58       per=0;
59       if  l>=a-1
60          for  j=L(M(i,L)>0)
61           for  k=1:indi(i,j)
62               if  N(i,j,k)<=l+1
63                   if  sum(I{i,j,k}([L,i]))==N(i,j,k)
64                       per=1;
65                       break
66                   end
67               end
68          end
69          if  per
70              break
71          end
72        end
73
74       if  per
75           cd=cd+1;
76           TToE=0;
77           break
78       elseif  M(i,i)>0
79           per=1;
80           cd=cd+1;
81           TToE=0;
82           break
83       end
84
85       conj=conj+1;
86       Fs=FXt(A,L,i,y);
87       S1=find(A(:,i));
88       c=sum([cr,A(:,i)],2);
```

```
89      prop1=0;
90      for j=1:l
91          if ismembc(F{j},S1)
92              prop1=1;
93              if sum(c==0)>0
94                  in=in+1;
95                  ConjIn=Incompatibles11(A,L,i,L(j),l);
96                  indi(i,ConjIn(1))=indi(i,ConjIn(1))+1;
97                  N(i,ConjIn(1),indi(i,ConjIn(1)))=numel(ConjIn
                        );
98                  M(i,ConjIn(1))=1;
99                  I{i,ConjIn(1),indi(i,ConjIn(1))}(ConjIn)=1;
100                 if a>N(i,ConjIn(1),indi(i,ConjIn(1)))
101                         a=N(i,ConjIn(1),indi(i,ConjIn(1)));
102                 end
103                 break
104             end
105         end
106     end
107     if prop1
108         TToE=0;
109         break
110     end
111     if  isempty(Fs)   %Prop 2
112         if any(c==0)
113             ConjIn=Incompatibles22(A,L,i,S1,l);
114             in=in+1;
115             indi(i,ConjIn(1))=indi(i,ConjIn(1))+1;
116             N(i,ConjIn(1),indi(i,ConjIn(1)))=numel(ConjIn);
117             M(i,ConjIn(1))=1;
118             I{i,ConjIn(1),indi(i,ConjIn(1))}(ConjIn)=1;
119             if a>N(i,ConjIn(1),indi(i,ConjIn(1)))
120                 a=N(i,ConjIn(1),indi(i,ConjIn(1)));
121             end
122         end
123         TToE=0;
```

```
124             break
125         end
126         if c>0 %Es testor tipico
127             tt=tt+1;
128             TT{tt}=[L,i];
129             stop1=1;
130             TToE=1;
131             continue
132         end
133         if i==n
134             stop1=1;
135             continue
136         end
137         L=[L,i];
138         TToE=1;
139         cr=c;
140         l=numel(L);
141         F=cell(1,l);
142         for j=1:l-1
143             F{j}=FXt(A,L,L(j),y);
144         end
145         F{l}=Fs;
146         stop1=1;
147 
148     end
149 
150     %% 4. Seleccion del nuevo candidato
151     if i~=n
152         i=i+1;
153         stop1=0;
154         continue
155     end
156     if l==0
157         break
158     end
159     if TToE
```

```
160             [p,U]=gap_L([L,i]);
161             if p~=0
162                 L=U;
163                 i=p;
164             else
165                 break
166             end
167         else
168             i=L(1);
169             L(1)=[];
170             stop1=1;
171         end
172         %%%Actualizacion
173         if isempty(L)
174             cr=zeros(y,1);
175         else
176             cr=sum(A(:,L),2);
177         end
178         l=numel(L);
179         F=cell(1,l);
180         if ~per
181             for j=1:l
182                 F{j}=FXt(A,L,L(j),y);
183             end
184         end
185     end
186     tmpej=toc;
187
188     end
```

Bibliografía

[1] Alba-Cabrera, Eduardo, Salvador Godoy-Calderon y Julio Ibarra-Fiallo: *Generating synthetic test matrices as a benchmark for the computational behavior of typical testor-finding algorithms*. Pattern Recognition Letters, 80:46–51, 2016.

[2] Alba-Cabrera, Eduardo, Salvador Godoy-Calderon, Manuel S Lazo-Cortés, J Fco Martínez-Trinidad y Jesús A Carrasco-Ochoa: *On the Relation Between the Concepts of Irreducible Testor and Minimal Transversal*. IEEE Access, 7:82809–82816, 2019.

[3] Alba-Cabrera, Eduardo, Julio Ibarra-Fiallo y Salvador Godoy-Calderon: *A theoretical and practical framework for assessing the computational behavior of typical testor-finding algorithms*. En *Iberoamerican Congress on Pattern Recognition*. Springer, 2013.

[4] Alba-Cabrera, Eduardo, Julio Ibarra-Fiallo, Salvador Godoy-Calderon y Fernando Cervantes-Alonso: *YYC: A fast performance incremental algorithm for finding typical testors*. En *Iberoamerican Congress on Pattern Recognition*, páginas 416–423. Springer, 2014.

[5] Algaza, Y Santiesteban y A Pons Porrata: *LEX: A New Algorithm for the Calculus of all Typical Testors*. vol, 1:85–95.

[6] Bailey, James, Thomas Manoukian y Kotagiri Ramamohanarao: *A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns*. En *Third IEEE International Conference on Data Mining*, páginas 485–488. IEEE, 2003.

[7] Berge, Claude: *Hypergraphs: combinatorics of finite sets*, volumen 45. Elsevier, 1984.

[8] Cheguis, IA y SV Yablonskii: *About testors for electrical outlines*. Uspieji Matematicheskij Nauk, 4(66):182–184, 1955.

[9] Cheguis, SV YABLONSKII IA y SV Yablonskii: *Logical Methods for controlling electrical systems*. Trudy Matematicheskava Institutaimeni VA Steklova, 1958.

[10] Diaz-Sanchez, German, Ivan Piza-Davila, Guillermo Sanchez-Diaz, Miguel Mora-Gonzalez, Oscar Reyes-Cardenas, Abraham Cardenas-Tristan y Carlos Aguirre-Salado: *Typical testors generation based on an evolutionary algorithm*. En *International Conference on Intelligent Data Engineering and Automated Learning*, páginas 58–65. Springer, 2011.

[11] Dmitriev, AN, Yu I Zhuravlev y FP Krendeliev: *About mathematical principles of objects and phenomena classification*. Diskretni Analiz, 7:3–15, 1966.

[12] Dong, Guozhu y Jinyan Li: *Mining border descriptions of emerging patterns from dataset pairs*. Knowledge and Information Systems, 8(2):178–202, 2005.

[13] Eiter, Thomas y Georg Gottlob: *Hypergraph transversal computation and related problems in logic and AI*. En *European Workshop on Logics in Artificial Intelligence*, páginas 549–564. Springer, 2002.

[14] Elbassioni, Khaled, Matthias Hagen y Imran Rauf: *A lower bound for the HBC transversal hypergraph generation*. Fundamenta Informaticae, 130(4):409–414, 2014.

[15] Gainer-Dewar, Andrew y Paola Vera-Licona: *The minimal hitting set generation problem: algorithms and computation*. SIAM Journal on Discrete Mathematics, 31(1):63–100, 2017.

[16] González-Guevara, Víctor Iván, Salvador Godoy-Calderon, Eduardo Alba-Cabrera y Hiram Calvo: *Symbolic learning for improving the performance of transversal-computation algorithms*. IEEE Access, 7:19752–19761, 2019.

[17] González-Guevara, Víctor Iván, Salvador Godoy-Calderon, Eduardo Alba-Cabrera y Julio Ibarra-Fiallo: *A mixed learning strategy for finding typical testors in large datasets*. En *Iberoamerican Congress on Pattern Recognition*, páginas 716–723. Springer, 2015.

[18] Guillermo, Sanchez Diaz, Ivan Piza-Davila, Manuel Lazo-Cortes, Miguel Mora-Gonzalez y Javier Salinas-Luna: *A fast implementation of the CT_EXT algorithm for the testor property identification*. En *Mexican International Conference on Artificial Intelligence*, páginas 92–103. Springer, 2010.

[19] Gunopulos, Dimitrios, Heikki Mannila, Roni Khardon y Hannu Toivonen: *Data mining, hypergraph transversals, and machine learning*. En *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, páginas 209–216, 1997.

[20] Hagen, Matthias: *Algorithmic and Computational Complexity Issues of MONET*. Cuvillier Verlag, 2008.

[21] Hagen, Matthias: *Lower bounds for three algorithms for transversal hypergraph generation*. Discrete Applied Mathematics, 157(7):1460–1469, 2009.

[22] Ho, Yu Chi y David L Pepyne: *Simple explanation of the no-free-lunch theorem and its implications*. Journal of optimization theory and applications, 115(3):549–570, 2002.

[23] Kavvadias, Dimitris J y Elias C Stavropoulos: *An efficient algorithm for the transversal hypergraph generation.* J. Graph Algorithms Appl., 9(2):239–264, 2005.

[24] Klamt, Steffen, Utz Uwe Haus y Fabian Theis: *Hypergraphs and cellular networks*. PLoS Comput Biol, 5(5):e1000385, 2009.

[25] Li, Dong, Zhiming Xu, Sheng Li y Xin Sun: *Link prediction in social networks based on hypergraph*. En *Proceedings of the 22nd International Conference on World Wide Web*, páginas 41–42, 2013.

[26] Li, Fang y Qunxiong Zhu: *Document Clustering in Research Literature Based on NMF and Testor Theory.* JSW, 6(1):78–82, 2011.

[27] Lias-Rodríguez, Alexsey y Aurora Pons-Porrata: *BR: A new method for computing all typical testors*. En *Iberoamerican Congress on Pattern Recognition*, páginas 433–440. Springer, 2009.

[28] Pons-Porrata, Aurora, Reynaldo Gil-García y Rafael Berlanga-Llavori: *Using typical testors for feature selection in text categorization*. En *Iberoamerican Congress on Pattern Recognition*, páginas 643–652. Springer, 2007.

[29] Pons-Porrata, Aurora, José Ruiz-Shulcloper y Rafael Berlanga-Llavori: *A method for the automatic summarization of topic-based clusters of documents*. En *Iberoamerican Congress on Pattern Recognition*, páginas 596–603. Springer, 2003.

[30] Ruiz, J, E Alba y M Lazo: *Introducción al Reconocimiento de Patrones*. Ediciones CINVESTAV, México, 1994.

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

[32] Sanchez-Diaz, Guillermo, Manuel Lazo-Cortes y Ivan Piza-Davila: *A fast implementation for the typical testor property identification based on an accumulative binary tuple*. International Journal of Computational Intelligence Systems, 5(6):1025–1039, 2012.

[33] Wolpert, David H, William G Macready *y cols.*: *No free lunch theorems for search*. Informe técnico, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[34] Zhang, Byoung Tak: *Random hypergraph models of learning and memory in biomolecular networks: shorter-term adaptability vs. longer-term persistency*. En *2007 IEEE Symposium on Foundations of Computational Intelligence*, páginas 344–349. IEEE, 2007.