INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

LABORATORIO DE INTELIGENCIA ARTIFICIAL

Técnicas de aprendizaje simbólico para algoritmos de búsqueda de testores típicos

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

Ing. Víctor Iván González Guevara

DIRECTOR DE TESIS:

Dr. Salvador Godoy Calderón

Ciudad de México

Julio de 2016

SIP-14

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, siendo las 12:00 horas del día 09 del mes de junio de 2016 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Técnicas de aprendizaje simbólico para algoritmos de búsqueda de testores típicos"

Presentada por el alumno:

GONZÁLEZ	**GUEVARA**	**VÍCTOR IVÁN**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	4	0	0	8	8

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de Tesis

Dr. Salvador Godoy Calderón

Dr. Edgardo Manuel Felipe Riverón

Dra. Nareli Cruz Cortés

Dr. Ricardo Barrón Fernández

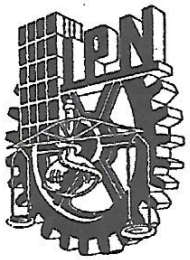
Dr. Jesús Guillermo Figueroa Nazuno

Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

ESTADOS UNIDOS MEXICANOS
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 7 del mes Julio del año 2016, el que suscribe Víctor Iván González Guevara estudiante del Programa de Maestría en Ciencias de la Computación con número de registro A14008, adscrito al Laboratorio de Inteligencia Artificial, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Salvador Godoy Calderón y cede los derechos del trabajo titulado "*Técnicas de aprendizaje simbólico para algoritmos de búsqueda de testores típicos*", al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección vgonzalez.general@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Víctor Iván González Guevara

Nombre y firma

Resumen

Este trabajo presenta una estrategia general de aprendizaje simbólico para mejorar el desempeño de los algoritmos de búsqueda de testores típicos. El objetivo de la estrategia propuesta es permitir que cualquier algoritmo de búsqueda de testores típicos logre la reducción más significativa posible del espacio de búsqueda. La estrategia propuesta se basa en la caracterización de cada uno de los subconjuntos en el espacio de búsqueda, particionando el espacio en cuatro clases distintas. Cada una de estas clases identifica y almacena diferente información de aprendizaje. Cualquier algoritmo de búsqueda de testores típicos, ya sea interno o externo, puede ser adaptado para incorporar la estrategia propuesta y tomar ventaja de la información obtenida de diversas maneras.

Abstract

This paper presents a general symbolic learning strategy for improving the performance of typical testor-finding algorithms. The goal of the proposed strategy is to allow any typical testor-finding algorithm to achieve the most significant reduction possible in the search space of any problem. It is based on the characterization of each subset in the search space, partitioning it into four different classes. Each one of these classes allows the identification and storage of different learning information. Any typical testor-finding algorithm, either internal or external, can be adapted to incorporate the proposed strategy and to take advantage of the learned information in different ways.

A mi mamá

María Antonieta Guevara Melo

A quien dedico especialmente esta parte de mis sueños, ya que sin ella no sería quien soy actualmente

A mi hermano

César Daniel González Guevara

Por ser un gran amigo que ha estado para mí en todo momento, escuchándome y apoyándome

A Alejandro Gómez Soto

Por estar incondicionalmente en cada momento, apoyándonos siempre como parte de la familia

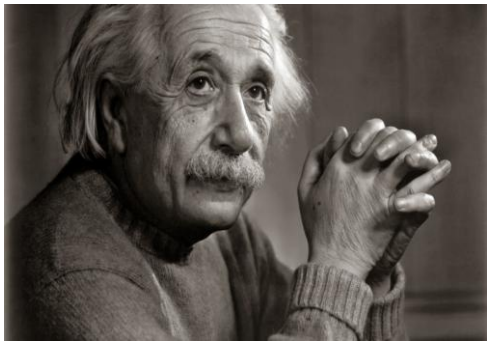
Agradecimientos

El Instituto Politécnico Nacional (IPN), mi segunda Patria, ha sido mi orgullo y emblema desde hace 10 años. Más que una institución, es el corazón de mi vida en la que me formé con ética en el ámbito cultural, deportivo y académico-científico. El IPN ha sido testigo y parte de innumerables escenarios durante mi estancia en el mismo, los cuales serán inolvidables. En sus cimientos, mis fracasos se volvieron el coraje para emprender un nuevo camino hacia el éxito, formándome como un ser íntegro. Por ello, en lo que reste de mi existencia seguiré honrando con placer y honor los colores guinda y blanco, así como su escudo y su noble mascota; cantando con amor y decoro su himno, y siempre llevaré muy en alto las siglas IPN, ya que soy politécnico por convicción y no por circunstancia.

Agradezco también al Centro de Investigación en Computación por brindarme la oportunidad de desenvolverme como investigador, contando con el apoyo económico de la SIP y del CONACYT. Asimismo, agradezco a todos los profesores y amigos que durante mi formación académica me brindaron su apoyo incondicional y dejaron en mí grandes enseñanzas, muchas gracias.

Agradezco particularmente al Dr. Salvador Godoy Calderón, el que me haya abierto las puertas para seguir adelante con ésta investigación. Gracias a su apoyo, confianza, enseñanzas e interés no perdí el entusiasmo de seguir adelante ya que me mostró que es un gran investigador y un gran ser humano. También agradezco a todos los investigadores que integraron la comisión evaluadora, por sus consejos e interés mostrados durante el desarrollo de esta investigación.

Finalmente quiero agradecer a mi familia: mi mamá y a mi hermano por siempre por brindarme su apoyo, sus consejos, por compartir mis tristezas y alegrías. Gracias por acompañarme en este viaje, ya que sin ustedes yo no sería lo que soy ahora.



No pretendamos que las cosas cambien si siempre hacemos lo mismo. La crisis es la mejor bendición que puede sucederle a las personas y países, porque la crisis trae progresos. La creatividad nace de la angustia como el día nace de la noche oscura. Es en la crisis que nace la inventiva, los descubrimientos y las grandes estrategias. Quien supera la crisis se supera a sí mismo, sin quedar "superado".

Quien atribuye a las crisis sus fracasos y penurias, violenta su propio talento y respeta más a los problemas que a las soluciones. La verdadera crisis es la crisis de la incompetencia. El inconveniente de las personas y los países es la pereza para encontrar las salidas y soluciones. Sin crisis no hay desafíos, sin desafíos la vida es una rutina, una lenta agonía. Sin crisis no hay méritos. Es en la crisis donde aflora lo mejor de cada uno, porque si crisis todo viento es caricia. Hablar de crisis es promoverla, y callar en la crisis es exaltar el conformismo. En vez de esto, trabajemos duro. Acabemos pronto de una vez con la única crisis amenazadora, que es la tragedia de no querer luchar por superarla.

Albert Einstein

Índice general

1. Introducción **1**
- 1.1. Problemática . . . 2
- 1.2. Motivación . . . 2
- 1.3. Hipótesis . . . 2
- 1.4. Objetivos . . . 3
  - 1.4.1. Objetivo general . . . 3
  - 1.4.2. Objetivos específicos . . . 3
- 1.5. Aportaciones . . . 3
- 1.6. Estructura de la tesis . . . 3

2. Revisión del estado del arte **5**
- 2.1. Enfoque determinístico . . . 5
- 2.2. Enfoque heurístico . . . 6

3. Marco teórico **7**
- 3.1. Conceptos básicos en Teoría de Testores . . . 7
- 3.2. Conceptos básicos en Aprendizaje Automático . . . 9

4. Propuesta de solución **11**
- 4.1. Caracterización del espacio de búsqueda . . . 11
- 4.2. Estrategia de aprendizaje . . . 13
- 4.3. Adaptaciones sobre algoritmos de búsqueda de testores típicos . . . 15
  - 4.3.1. Algoritmo Binario Recursivo (BR) . . . 15
  - 4.3.2. Algoritmo Yablonsky y Compatibles (YYC) . . . 18

5. Resultados experimentales **23**
- 5.1. Escenario 1: Matrices con crecimiento lineal en columnas . . . 24
- 5.2. Escenario 2: Matrices con crecimiento exponencial en renglones . . . 28
- 5.3. Escenario 3: Matrices con crecimiento proporcional . . . 32
- 5.4. Escenario 4: Matrices identidad . . . 35
- 5.5. Resumen general de resultados . . . 39

6. Conclusiones y trabajo futuro **41**
- 6.1. Conclusiones generales . . . 41
- 6.2. Trabajo futuro . . . 42

Referencias **43**

Apéndices **45**

A. Operadores para la generación de matrices sintéticas de prueba **46**
A.1. Operador concatenación (φ) 46
A.2. Operador combinación (θ) 46
A.3. Operador desplazamiento (γ) 47

Índice de figuras

4.1. Ejemplo de caracterización de elementos en el espacio 13

5.1. Gráfica comparativa del número *hits* entre ambas versiones de BR ante matrices con crecimiento lineal en columnas 25
5.2. Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento lineal en columnas 25
5.3. Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento lineal en columnas 26
5.4. Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento lineal en columnas 27
5.5. Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento lineal en columnas 27
5.6. Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento lineal en columnas 27
5.7. Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento exponencial en renglones 28
5.8. Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento exponencial en renglones 29
5.9. Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices con crecimiento exponencial en renglones 30
5.10. Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento exponencial en renglones 30
5.11. Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento exponencial en renglones 31
5.12. Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento exponencial en renglones 31
5.13. Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento proporcional 33
5.14. Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento proporcional 33
5.15. Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices con crecimiento proporcional 34
5.16. Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento proporcional 34

5.17. Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento proporcional 35
5.18. Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento proporcional 35
5.19. Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices identidad . . . 36
5.20. Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices identidad 37
5.21. Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices identidad . . 38
5.22. Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices identidad 38
5.23. Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices identidad 38
5.24. Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices identidad 39

Índice de tablas

5.1. Resultados de las dos versiones de BR ante matrices con crecimiento lineal en columnas . . . 24
5.2. Resultados de las dos versiones de YYC ante matrices con crecimiento lineal en columnas . . . 26
5.3. Resultados de las dos versiones de BR ante matrices con crecimiento exponencial en renglones 28
5.4. Resultados de las dos versiones de YYC ante matrices con crecimiento exponencial en renglones 29
5.5. Resultados de las dos versiones de BR ante matrices con crecimiento proporcional 32
5.6. Resultados de las dos versiones de YYC ante matrices con crecimiento proporcional 34
5.7. Resultados de las dos versiones de BR ante matrices identidad 36
5.8. Resultados de las dos versiones de YYC ante matrices identidad 37
5.9. Resumen general de resultados 40

Índice de algoritmos

1. Algoritmo BR [Estructura básica] 17
2. Analizar($Lista$) [BR versión original] 18
3. Analizar($Lista$) [BR versión adaptada] 19
4. Algoritmo YYC [Versión original] 20
5. ConjuntoComplatible(τ_k, c_p) 20
6. ActualizarDominaciones(R) 21
7. Algoritmo YYC [Versión adaptada] 22

Capítulo 1

Introducción

En algunos momentos selectos en la historia de la ciencia, se han manifestado algunas convergencias entre diferentes enfoques de un mismo fenómeno, resultando ser equivalentes. Tal fue el caso, por ejemplo, en física cuántica, con la dualidad onda-partícula, que postula la equivalencia del comportamiento entre ondas y las partículas [1]. En otras ocasiones, sin embargo, la equivalencia emerge de modelos aparentemente tan diferentes, que inicialmente no es claro que modelen el mismo fenómeno. Tal es el caso del problema conocido por el acrónimo MONET (por su nombre en inglés *Monotone Equivalence Test*). Se trata de un problema en lógica simbólica que, en su forma original, como problema de decisión, consiste en decidir si dos formas normales monótonas, una disyuntiva (*FND: Forma Normal Disyuntiva*) y otra conjuntiva (*FNC: Forma Normal Conjuntiva*), son equivalentes. La relevancia de este problema radica justamente en las equivalencias conocidas y en la gran cantidad de aplicaciones del concepto que se abordan desde la perspectiva de diferentes disciplinas. El problema MONET es equivalente al problema de decidir si dos hipergrafos simples son transversales (problema TRANSHYP). También, es equivalente al problema de decidir si un subconjunto de variables en un conjunto rugoso, es un reducto del mismo. Por último, el problema MONET támbien es equivalente a de decidir si una familia de subconjuntos de rasgos es la familia de testores típicos en una muestra de supervisión. En conjunto, todas estas instancias equivalentes conforman la base teórica para numerosas aplicaciones en inteligencia artificial, reconocimiento de patrones, biología computacional, bases de datos, minería de datos, sistemas distribuidos, teoría de grafos, aprendizaje automático, lógica y sistemas de comunicación móviles.

En consecuencia, MONET termina por ser un problema de cobertura, que puede ser interpretado como la enumeración (en algún sentido) de todas las soluciones mínimas de algún sistema. Parece natrural entonces, que los trabajos de investigación que planteen nuevos algoritmos para solucionar problemas de tipo MONET o que mejoren las técnicas ya conocidas, sean relevantes desde el punto de vista de las aplicaciones prácticas.[2]

La selección de rasgos es la rama del reconocimiento de patrones responsable de identificar aquellos rasgos descriptivos que proporcionan información relevante para efectos de clasificación. La teoría de testores es una de las herramientas más comunes utilizada para dicha tarea. Durante la última década varios algoritmos han sido diseñados para encontrar la familia completa de testores típicos en un conjunto de datos [3] [4] [5]. Desafortunadamente, la complejidad en el tiempo para encontrar todos los testores típicos tiene un crecimiento exponencial con respecto al número de rasgos que describen a los objetos. Además, investigaciones recientes han dado a conocer los diferentes elementos

que tienen efecto sobre la complejidad de este problema, tales como, el número de renglones en la matriz básica inicial, la densidad de esa matriz, el número de testores típicos contenidos en ella y su estructura subyacente. Todos estos factores complican seriamente la búsqueda de testores típicos en grandes conjuntos de datos. Además, algunos de los resultados empíricos publicados en *benchmarks* cuidadosamente diseñados como en [6], se suman a la intuición de que no hay un único algoritmo de búsqueda de testores típicos que tenga el mejor comportamiento posible para cualquier problema. Este tipo de intuición hacia el teorema *No-Free-Lunch*, tomado del campo de los algoritmos evolutivos y bioinspirados, fomenta la investigación de técnicas que permitan un mejor desempeño de los algoritmos utilizados para encontrar testores típicos.

Con ese objetivo en mente, este trabajo propone una estrategia de aprendizaje, diseñada para dirigir de manera eficiente la búsqueda de testores típicos en el espacio de búsqueda. La estrategia propuesta será responsable de identificar y almacenar la información pertinente para la finalidad indicada, mientras que el algoritmo de búsqueda de testores típicos subyacente decide cuándo y cómo se hace uso de la información aprendida.

1.1. Problemática

Bajo la perspectiva clásica de la teoría de testores, se han diseñado diferentes algoritmos para encontrar la familia completa de testores típicos en un conjunto de datos. Sin embargo, a medida que el conjunto de datos crece, menor es el desempeño de los algoritmos. No obstante, al considerar la tendencia actual de trabajar con grandes volúmenes de datos, la perspectiva clásica de la teoría de testores parece ya no aportar más conocimiento que pueda ayudar a generar nuevos algoritmos o mejorar el desempeño de los ya existentes.

1.2. Motivación

Dadas las conocidas equivalencias entre problemas de la familia MONET y la gran cantidad de aplicaciones prácticas que se abordan a partir del estudio de dichos problemas, la motivación principal de este trabajo es aportar elementos, desde una perspectiva no estudiada previamente (aprendizaje simbólico) para ayudar a resolver la problemática anteriormente planteada, en el entendido de que cualquier solución no solo tendrá impacto en teoría de testores, sino también, en todas aquellas disciplinas que tengan algún problema equivalente a MONET.

1.3. Hipótesis

El punto de partida para este trabajo es la hipótesis de que al adaptando técnicas de aprendizaje simbólico sobre los algoritmos de búsqueda de testores típicos es posible mejorar su desempeño.

1.4. Objetivos

A continuación se enuncia tanto el objetivo general como los objetivos específicos de esta tesis.

1.4.1. Objetivo general

Proponer una técnica general de aprendizaje simbólico para mejorar el desempeño de los algoritmos de búsqueda de testores típicos.

1.4.2. Objetivos específicos

1. Revisar e implementar algunos de los algoritmos de búsqueda de testores típicos existentes en la literatura.
2. Caracterizar el espacio asociado a la búsqueda de testores típicos.
3. Diseñar una estrategia de aprendizaje a partir de la caracterización del espacio que permita mejorar el desempeño de los algoritmos de búsqueda de testores típicos.
4. Seleccionar y adaptar dos algoritmos de búsqueda de testores típicos incorporándoles la estrategia de aprendizaje.
5. Comparar experimentalmente el desempeño de los algoritmos seleccionados respecto a sus versiones adaptadas y evaluar los resultados.

1.5. Aportaciones

Hasta donde se tiene noticia, esta tesis es el primer trabajo en teoría de testores que se aborda desde la perspectiva del aprendizaje simbólico, al plantear un nuevo esquema de caracterización para el espacio asociado a la búsqueda de testores típicos y una nueva técnica general de aprendizaje simbólico, que mejora el desempeño de los algoritmos de búsqueda de testores típicos.

1.6. Estructura de la tesis

Este trabajo de tesis está organizado de la siguiente manera:

En el Capítulo 2 se describen algunos trabajos sobre teoría de testores y sus diferentes perspectivas al afrontar el problema de búsqueda de testores típicos.

En el Capítulo 3 se dan los conceptos clásicos, tanto en teoría de testores como en aprendizaje automático, con el fin de que haya una mayor comprensión del problema y de la propuesta de solución que se propone.

En el Capítulo 4 se expresa detalladamente la propuesta de solución de este trabajo; comenzando por explicar, que bajo la perspectiva de la inteligencia artificial se puede concebir el proceso de solución a un problema como una búsqueda sobre un espacio de estados. Posteriormente, se identifican algunas propiedades del espacio asociado al problema de búsqueda de testores típicos, las cuales permiten

estructurar el espacio y, a partir de dicha estructura, plantear una estrategia de aprendizaje que será incorporada en dos de los algoritmos más representativos de la Teoría de Testores (BR y YYC).

En el Capítulo 5 se muestran los resultados experimentales de la comparación del desempeño entre las versiones originales de los algoritmos BR y YYC, respecto de sus versiones adaptadas que incorporan la estrategia de aprendizaje.

Finalmente, en el Capítulo 6 se dan las conclusiones acerca del conocimiento adquirido en el transcurso de la investigación y el trabajo futuro que se deriva de ésta.

Capítulo 2

Revisión del estado del arte

La Teoría de Testores [7] ha demostrado en repetidas ocasiones ser una herramienta útil para los problemas de selección de rasgos en reconocimiento de patrones. Los testores típicos han sido utilizados en una amplia variedad de problemas prácticos, tales como el diagnóstico de enfermedades [8], la categorización de texto [9], el resumen de documentos [10] y el agrupamiento de documentos [11]. Además, la teoría de testores ha tenido un crecimiento importante en la última década, sobre todo con relación al desarrollo de nuevos algoritmos [3][4][5] [12]. Generalmente a todos estos algoritmos se les conoce como algoritmos de búsqueda de testores típicos (ABTT).

Por otro lado, uno de los objetivos principales de los ABTT es obtener el mejor desempeño posible, y así afrontar problemas con conjuntos de datos cada vez mayores como los que se presentan en aplicaciones reales. Con base en lo anterior, se han establecido dos enfoques importantes en los que se clasifican los ABTT: el enfoque determinístico y el enfoque heurístico. A continuación se enuncian los trabajos que tienen la mayor relación con esta investigación según el enfoque.

2.1. Enfoque determinístico

Bajo el enfoque determinístico existen principalmente dos trabajos que tienen estrecha relación con esta investigación y que sirvieron de inspiración para la propuesta de solución que se presenta en este trabajo.

El primer trabajo presenta el algoritmo BR (acrónimo en inglés de *Binary Recursive*) [3], el cual fue diseñado como una alternativa eficiente ante los algoritmos LEX [5] y CT-EXT [13]. BR evita la revisión de un gran número de subconjuntos de rasgos que son irrelevantes en la búsqueda de testores típicos, esto lo hace mediante el concepto de conjunto excluyente y el aprovechamiento de la naturaleza binaria de la entrada, utilizando operaciones binarias que detecten las propiedades de cada subconjunto a revisar.

El segundo trabajo, descrito en [14], define una implementación rápida del algoritmo CT-EXT [13] (que es uno de los algoritmos más rápidos reportads hasta el momento) basada en una tupla binaria acumulativa, desarrollada para encontrar todos los testores típicos. La tupla binaria acumulativa de este algoritmo, es una manera útil de simplificar la búsqueda de subconjuntos de rasgos que cumplan

con la propiedad de testor, debido a que su aplicación disminuye el número de operaciones implicadas en el proceso de búsqueda de todos los testores típicos.

2.2. Enfoque heurístico

Bajo el enfoque heurístico existen principalmente dos trabajos que tienen estrecha relación con esta investigación y que sirvieron de inspiración para la propuesta de solución que se presenta en este trabajo.

En el primer trabajo presenta una modificación del algoritmo *Hill-Climbing* para resolver el problema de búsqueda de testores típicos [15]. Este trabajo establece que hay problemas para los cuales no es necesario encontrar el conjunto completo de testores típicos y que solo basta con encontrar un subconjunto de ellos. Por esta razón, se define un operador de aceleración el cual es incorporado en el paso de mutación del algoritmo *Hill-Climbing*. Este operador mejora la capacidad de exploración del algoritmo, siendo capaz de encontrar un subconjunto de rasgos que es testor típico mediante un número mínimo de operaciones.

El operador de aceleración depende principalmente de un índice para la evaluación de los subconjuntos de rasgos y, de acuerdo al valor que entregue el índice serán las acciones que se llevarán a cabo sobre el subconjunto de rasgos. Si el índice toma el valor cero, entonces el subconjunto es un testor típico y se almacena como parte de la solución. Mientras que si el valor del índice es positivo, entonces el subconjunto es un testor y remueve cierta cantidad de rasgos del mismo para mantener la diversidad en la población del algoritmo. Por último, si el valor del índice es negativo, entonces el subconjunto se considera como un "no testorz únicamente se le agregan cierta cantidad de nuevos rasgos para preservar la diversidad de la población del algoritmo.

Además, la adaptación resultante después de incluir el operador de aceleración contiene dos parámetros con los cuales se configura el algoritmo para determinar cual es el subconjunto de testores típicos que debe hallar. Existen tres configuraciones: encontrar el conjunto de testores típicos de menor cardinalidad, encontrar el conjunto de testores típicos de una cardinalidad determinada o ignorar las dos configuraciones anteriores, lo que representa buscar la familia completa de testores típicos.

El segundo trabajo, definido en [16], establece un nuevo enfoque para resolver el problema de búsqueda de testores típicos, al plantear dicho problema como un problema de optimización. Además, en este trabajo se conjetura que el concepto de testor típico puede ser caracterizado mediante dos propiedades, siendo esta la aportación más significativa de este trabajo. No obstante, a partir de estas dos propiedades se define una función ponderada para la evaluación de subconjuntos de rasgos, que posteriormente es utilizada como función objetivo para un algoritmo genético, con el fin de encontrar la mayor cantidad de testores típicos en matrices de gran tamaño.

Capítulo 3

Marco teórico

En este capítulo se enuncian los conceptos clásicos tanto en teoría de testores como en aprendizaje automático, con el propósito de brindar un mayor contexto del problema y de la propuesta de solución que se plantea.

3.1. Conceptos básicos en Teoría de Testores

El concepto de testor es la base de la llamada Teoría de Testores que surge en la década de los cincuenta [17] y se desarrolla como una rama de la Lógica Matemática.

Aunque en un inicio se formula el concepto de testor a partir de la aplicación de métodos lógicos para la localización de desperfectos en ciertos circuitos eléctricos, poco tiempo después se vincula a los problemas de reconocimiento de patrones [18], y desde entonces, a la par de la Teoría de Testores para esquemas lógicos, se desarrolla también la Teoría de Testores para problemas de clasificación.

La primera formulación del concepto de testor en el marco del reconocimiento de patrones es dada por Yu. I. Zhuravlev en 1966 [18], a partir de la idea introducida por I. A. Cheguis y S. V. Yablonsky [17] en la década anterior, al abordar el problema de la búsqueda de desperfectos en circuitos eléctricos.

En [18] la búsqueda de testores se considera como un problema de reconocimiento de patrones con aprendizaje, pero de forma muy simplificada. Las variables en términos de las cuales se describen los objetos sólo toman valores en $\{0, 1\}$ y la muestra de supervisión (denominada como tabla) se considera dividida en dos subtablas T_0 y T_1, que corresponden a dos clases disjuntas. A partir de estas consideraciones se da la siguiente definición.

Definición 3.1 (Definición bivalente)**.** *El conjunto $R' = \{x_1, x_2, \ldots, x_s\}$ de columnas de una tabla $T = \{T_0, T_1\}$ se denomina testor si después de eliminar de T todas las columnas excepto las de R', no existe renglón alguno de T_0 igual a uno de T_1. R' es un testor típico si no existe $R' \subset R''$ tal que R'' sea testor. [18]*

Posteriormente y de forma muy natural el concepto se extiende a rasgos no necesariamente bivalentes y con cualquier número de clases, pero manteniéndose dentro de la teoría clásica de conjuntos. A esta versión extendida del concepto se le conoce como testor clásico de Zhuravlev [18].

Definición 3.2 (Muestra de supervisión)**.** *Se denomina muestra de supervisión al conjunto* $TM = \{O_1, O_2, \ldots, O_m\}$ *de* m *objetos, en el que cada objeto* O_i *pertenece a una clase* $K_i \in \{K_1, K_2, \ldots K_c\}$ *y está descrito en términos de* r *rasgos* $R = \{x_1, x_2, \ldots, x_r\}$*. Cada rasgo* $x_i \in R$ *toma valores en un conjunto* $M_i, i = 1, \ldots n$ *(dominio) [15].*

En términos de la muestra de supervisión se pueden definir los testores de Zhuravlev como sigue:

Definición 3.3 (Testor de Zhuravlev)**.** *Un subconjunto de rasgos* $\tau \subseteq R$ *es un testor, si y solo si, al reducir la muestra de supervisión a* $TM\,|_\tau$ *no se confunden objetos en clases distintas. Además,* τ *es testor típico si no existe* $\sigma \subset \tau$ *tal que* σ *sea testor [18].*

Por conveniencia, los algoritmos en teoría de testores trabajan sobre una matriz binaria, conocida como **matriz básica** (MB), de menor tamaño que la muestra de supervisión. Para generar la matriz MB se construye previamente la matriz de diferencias (MD) utilizando un criterio de comparación booleano y de diferencia $D_i : M_i \times M_i \to \{0, 1\}$.

Definición 3.4 (Matriz de diferencias)**.** *La matriz de diferencias MD para los objetos* $O_i \in TM$*, es una matriz binaria, donde sus filas se obtienen mediante la comparación entre cada par de objetos de* TM *que pertenecen a clases distintas, utilizando un criterio de comparación booleano y de diferencia [15].*

Computacionalmente resulta más eficiente trabajar con la matriz MD en lugar de la muestra de supervisión TM. Debido a que para la creación de MD, la comparación entre dos objetos arbitrarios de TM se realiza sólo una vez. Además, MD es una matriz binaria.

A partir de la matriz MD se construye la matriz MB mediante las siguientes definiciones.

Definición 3.5 (Subrenglón)**.** *Decimos que* p *es un subrenglón de* q *si:* $\forall_j[q_j = 0 \Rightarrow p_j = 0]$ *y* $\exists_i[p_i = 0 \Rightarrow q_i = 1]$ *[15].*

Definición 3.6 (Renglón básico)**.** *Un renglón* p *de* MD *es básico si no hay ningún otro renglón en* MD *que sea subrenglón de* p *[15].*

Definición 3.7 (Matriz básica)**.** *La submatriz obtenida al reducir* MD *a todos sus renglones básicos se le conoce como matriz básica (denotada por* MB*) [15]*

Comúnmente, los algoritmos utilizados para encontrar el conjunto completo de testores típicos hacen uso de MB en lugar de MD, debido a la sustancial reducción de renglones entre MD y MB. Ahora, en términos de MB, podemos definir un testor como sigue:

Definición 3.8. *Un subconjunto de rasgos $\tau \subseteq R$ es un testor, si y solo si, al reducir la matriz básica a $MB\,|_\tau$ no aparece un renglón compuesto únicamente por ceros. Además, τ es testor típico si no existe $\sigma \subset \tau$ tal que σ sea testor.*

Los algoritmos empleados para encontrar el conjunto completo de testores típicos en matrices básicas se agrupan en dos clases de acuerdo al espacio sobre el cual buscan [19]. Estas clases son:

- **Algoritmos externos:** Buscan los testores típicos induciendo un orden y un recorrido sobre el espacio de búsqueda determinado por el conjunto potencia de los rasgos que describen a los objetos de estudio.
- **Algoritmos internos:** Buscan los testores típicos a partir de los subconjuntos que pueden formarse con los unos de la matriz básica. Esto sugiere que el espacio de los algoritmos internos es un subconjunto del espacio de los externos.

3.2. Conceptos básicos en Aprendizaje Automático

El término aprendizaje tiene diferentes significados en distintos contextos; sin embargo, en forma general, tanto para seres biológicos, como para agentes artificiales, se debe entender como todo cambio o modificación que un sistema realiza sobre sí mismo, a partir de sus experiencias anteriores, y que le permite lograr un mejor resultado la siguiente vez que enfrenta situaciones semejantes a las anteriormente experimentadas.

Entonces, aprender es el proceso por el cual se identifica y se almacena (memoriza) la información que resultará útil la siguiente vez que se enfrente una determinada situación (llamada información de aprendizaje, o simplemente aprendizaje). Evidentemente, al concebir de esa manera el proceso, la actividad de identificar la información útil es la actividad crítica; retenerla o memorizarla no representa reto alguno para una persona sana, y menos aún para una computadora.

De los anteriores conceptos se deriva una de las ramas más relevantes de la Inteligencia Artificial, la disciplina de Aprendizaje Automático (*Machine Learning*), cuya definición refleja claramente el objetivo de aprender:

Definición 3.9. *"Se dice que un programa de computadora aprende de la experiencia E, con respecto a alguna clase de tarea o actividad T y con respecto a alguna medida de desempeño M, cuando su desempeño en las tareas T mejora, según la medida M, después de experimentar la experiencia E" [20].*

Esta disciplina es el resultado de combinar técnicas del área de Reconocimiento de Patrones (el estudio de los procesos de representación y clasificación de datos) con conceptos y objetivos de otras disciplinas como Psicología, Ciencia Cognitiva y, por supuesto, Inteligencia Artificial.

En particular, la disciplina Aprendizaje Automático persigue la construcción de programas de computadora capaces de analizar datos y realizar predicciones al respecto. Una de las características más relevantes de los métodos seguidos por esta disciplina, el razonamiento inductivo, es decir, la

generalización a partir de ejemplos. Este tipo de razonamiento resulta fundamental cuando se requiere analizar un conjunto de datos y aprender de él.

Dos paradigmas principales se han desarrollado en el ámbito del aprendizaje automático [21]. Estos son:

- **Apendizaje Simbólico Empírico (ASE):** Este paradigma de aprendizaje se enfoca en la adquisición de nuevo conocimiento mediante técnicas inductivas haciendo uso intensivo de funciones de semejanza/diferencia.
- **Apendizaje Basado en Explicación (ABE):** Este paradigma de aprendizaje se enfoca en el refinamiento del conocimiento actual, para incrementar el desempeño de un sistema mediante la explotación eficiente de su conocimiento, ya sea aprendiendo nuevas reglas compuestas, o bien, aprendiendo a encontrar reglas de forma más eficiente.

Capítulo 4

Propuesta de solución

En el área de Inteligencia Artificial, una forma clásica de concebir el proceso de solución de problemas, es modelar el problema mismo como una máquina de estados y el proceso de solución como una búsqueda en el espacio de estados de dicha máquina (también conocido como *espacio de búsqueda*). Según ese modelo, a partir de un estado inicial y mediante un conjunto de reglas de transformación, se busca hallar uno o más elementos del espacio que satisfagan una condición u objetivo planteado. Por ello, al plantear la solución de un problema en esos términos, resulta conveniente conocer el espacio de búsqueda y, de ser posible, inducir algún tipo de orden o estructura sobre él, de manera que el conocimiento disponible sobre el espacio facilite el proceso de búsqueda, sin importar la técnica específica empleada para buscar.

Bajo la perspectiva anterior, el presente trabajo comienza por identificar algunas propiedades del espacio asociado al problema de encontrar testores típicos. Las propiedades identificadas permitirán estructurar el espacio y, a partir de dicha estructura, inferir una estrategia de aprendizaje que permita incrementar el desempeño de algunos algoritmos existentes para la tarea de encontrar testores típicos.

4.1. Caracterización del espacio de búsqueda

El planteamiento clásico de un problema consistente en encontrar testores típicos, comienza siempre con una muestra de supervisión en la que se describe a ciertos objetos en estudio a partir de un conjunto $R = \{x_1, x_2, \ldots, x_n\}$ de atributos, técnicamente llamados *rasgos descriptivos*. Esa muestra de supervisión organiza a los objetos en clases disjuntas y no vacías, formando con ellos una partición. Los testores típicos, como se presentó en el capítulo anterior, son subconjuntos de rasgos que cumplen dos propiedades importantes:

1. Discriminar objetos pertenecientes a clases distintas (testores),
2. No poder eliminar de ellos algún elemento (algún rasgo) sin que el conjunto restante pierda la propiedad de ser testor (son *minimales* según el criterio de cardinalidad).

Tradicionalmente, los algoritmos que buscan testores típicos reciben como entrada una matriz básica que contiene, de forma compactada, toda la información acerca de la comparación entre objetos que pertenecen a clases distintas en la muestra de supervisión del problema. También de manera tradicional, la matriz básica se asume construida con funciones binarias de diferencia.

Las dos propiedades anteriores pueden ser caracterizadas en términos de la matriz básica que constituye la entrada de cualquier problema:

Sea M una muestra de supervisión (particional) que describe a los objetos en estudio a partir del conjunto $R = \{x_1, x_2, \ldots, x_n\}$ de rasgos descriptivos y sea MB la matriz básica de M. Entonces, es posible realizar la siguiente definición:

Definición 4.1 (Propiedad de testor)**.** *Un subconjunto $\tau \subseteq R$ de rasgos descriptivos es un* **Testor** *en MB, si y sólo si, la submatriz restringida $MB|_\tau$ no contiene subrenglón alguno que esté compuesto únicamente por ceros.*

Para el caso de la segunda propiedad (ser minimal con respecto al criterio de cardinalidad) también se puede realizar una caracterización en términos de la matriz básica inicial. Sin embargo, en ese caso, la caracterización resulta independiente de la anterior:

Definición 4.2 (Propiedad de típico)**.** *Un subconjunto $\tau \subseteq R$ de rasgos descriptivos es* **Típico** *en MB, si y sólo si, la submatriz restringida $MB|_\tau$ contiene todos los renglones de una matriz identidad de dimensión $|\tau|$.*

Caracterizar de esta forma, resulta en una independencia de esta propiedad con respecto a la anterior. En la definición original de *Zhuravlev* (Definición 3.3) la etiqueta de típico se asigna exclusivamente a subconjuntos de rasgos que previamente cumplen la condición de ser testores. Con la caracterización anterior ese supuesto ya no se cumple y resulta posible que un subconjunto de rasgos sea típico, sin ser testor. A pesar de ese cambio conceptual resulta claro que, cuando un subconjunto de rasgos cumple las dos propiedades anteriores (Definiciones 4.1 y 4.2), entonces también satisface la definición original de *Zhuravlev*.

Esa independencia, que resulta de gran relevancia para la estrategia de aprendizaje propuesta en este trabajo, se puede verificar fácilmente en ejemplo de la Figura 4.1, en el cual el subconjunto τ_1 es un **Testor**, aunque no típico, mientras que el subconjunto τ_2 es **Típico** pero no testor. De la misma forma, el subconjunto τ_3 es un **Testor Típico**, es decir, satisface la definición original de *Zhuravlev* y satisface las dos propiedades independientes de las definiciones 4.1 y 4.2. Por último, el subconjunto τ_4 no satisface ninguna de las dos propiedades iniciales.

El ejemplo de la Figura 4.1 ilustra la primera pieza de conocimiento general que conforma la solución propuesta en este trabajo; el hecho de que en todo problema de encontrar testores típicos, el espacio de búsqueda se estructura como una partición en cuatro clases:

1. **Testores (no típicos).** Aquellos subconjuntos de rasgos que satisfacen la definición 4.1 pero no la definición 4.2.
2. **Típicos (no testores).** Aquellos subconjuntos que satisfacen la definición 4.2 pero no la definición 4.1.
3. **Testores Típicos.** Subconjuntos que satisfacen al mismo tiempo las definiciones 4.1 y 4.2.
4. **Indeterminados.** Todos los demás subconjuntos de rasgos que no satisfacen ninguna de las dos definiciones.

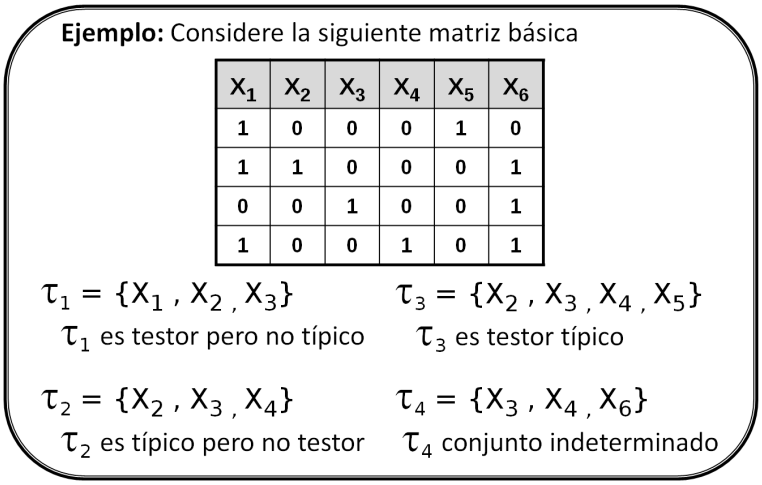


Figura 4.1: Ejemplo de caracterización de elementos en el espacio

4.2. Estrategia de aprendizaje

A partir de la estructura establecida sobre el espacio de búsqueda en términos de las cuatro clases definidas en la sección anterior, se propone una estrategia de aprendizaje que permita a los ABTT descartar la mayor cantidad de elementos dentro de ese espacio, durante el proceso de búsqueda.

Entenderemos por aprendizaje al proceso mediante el cual, cualquier pieza de información relevante para la solución de un problema, se identifica y almacena para su uso posterior. Así que, reforzando lo anteriormente dicho, el objetivo de la estrategia propuesta en este trabajo es proporcionar un conocimiento extra (adicional al conocimiento procedural que emplea cualquier ABTT) que permita realizar una búsqueda más eficiente sobre el espacio de búsqueda.

Cuando un ABTT es adaptado para utilizar la estrategia de aprendizaje, ésta es la responsable identificar y almacenar toda la información posible del subconjunto de rasgos que se está revisando, mientras que el algoritmo subyacente decide cuándo y cómo se hace uso de la información aprendida.

El análisis de un subconjunto de rasgos, en términos de la estrategia propuesta, comienza por determinar su pertenencia en alguna de las cuatro clases en las que se particiona el espacio de búsqueda (*Testor*, *Típico*, *Testor Típico* o *Indeterminado*). Dependiendo de la clase en la que dicho subconjunto sea clasificado, puede extraerse diferente información que ayude a mejorar el desempeño de un ABTT.

Las piezas de información más importantes que puede ser extraídas son las ***dominaciones entre rasgos***, que indican que los rasgos involucrados en la dominación (dominador y dominado) presentan una incompatibilidad entre ellos y por tanto, un conjunto que los incluya a todos ellos nunca es un testor típico. Las dominaciones entre rasgos se clasifican en dos tipos: **simples** y **compuestas**. En el caso de las dominaciones simples se establece una restricción uno a uno (esa pareja de rasgos no puede pertenecer a un testor típico), debido a que los unos en la columna de un rasgo (dominador) cubren a los unos en la columna del otro rasgo (dominado). De manera formal podemos definir una dominación simple de la siguiente manera:

> **Definición 4.3** (Dominación simple)**.** *Sean x_i y x_j dos rasgos descriptivos asociados a sus respectivas columnas en una matriz básica. Decimos que x_i* ***domina a*** *x_j, si y sólo si en todo renglón donde x_j tiene un valor 1, x_i también lo tiene.*

Las dominaciones compuestas establecen una restricción uno a muchos (el rasgo individual y el subconjunto de rasgos no pueden pertenecer, al mismo tiempo, a un testor típico), ya que los unos contenidos en las columnas asociadas al subconjunto (dominador compuesto) cubren a los unos en la columna del rasgo del rasgo individual (dominado). Sin embargo, es posible mapear el concepto de dominaciones simples a compuestas, al considerar el dominador simple como un dominador compuesto de cardinalidad uno. De esa manera, se puede establecer el siguiente concepto generalizado de dominación entre rasgos:

> **Definición 4.4** (Dominación generalizada)**.** *Sea σ un subconjunto de rasgos y x_i un rasgo individual, ambos asociados a sus respectivas columnas en una matriz básica. Decimos que el conjunto de rasgos σ* ***domina a*** *x_i, si y sólo si en todo renglón donde x_i tiene un valor 1, algún elemento de σ también lo tiene.*

Las técnicas de aprendizaje dividen su conocimiento en dos tipos: **conocimiento contextual** y **conocimiento adquirido**. En la estrategia de aprendizaje que se propone, se considera como conocimiento contextual a la información previamente conocida sobre cada una de las clases en las que se particiona el espacio de búsqueda y, como conocimiento adquirido, a las dominaciones entre rasgos previamente definidas.

Dependiendo de la clase en la que un subconjunto de rasgos sea clasificado ésta determina la información de aprendizaje que debe almacenarse y como debe usarse. Además, cada una de las clases aplica su propio conocimiento contextual para descartar subconjutos del espacio de búsqueda. A continuación, se describe como se gestiona tanto el conocimiento contextual como el conocimiento adquirido en cada clase.

1. Si el subconjunto a analizar pertenece a la clase **Testores (no típicos)**, se deberán almacenar todas las dominaciones contenidas en dicho subconjunto como parte de la información de aprendizaje. Además, todo subconjunto de rasgos que pertenece a esta clase es un testor no minimal y puede al menos ser reducido a un testor típico mediante la eliminación de ciertos rasgos que le permitan satisfacer la definición 4.2. Por lo tanto, resulta irrelevante revisar cualquiera de sus superconjuntos.

2. Si el subconjunto a analizar pertenece a la clase **Típicos (no testores)**, implica que dicho subconjunto no tiene información de aprendizaje que ayude a mejorar el proceso de búsqueda. Sin embargo, todo elemento en esta clase representa un candidato viable mediante el cual se puede construir un testor típico adicionando ciertos rasgos que le permitan satisfacer la definición 4.1.

3. Si el subconjunto a analizar pertenece a la clase **Testores Típicos**, se deberá almacenar dicho subconjunto como parte de la información de aprendizaje. No obstante, dado que todos los elementos de esta clase son el objetivo de la búsqueda, resulta irrelevante revisar cualquiera de sus subconjuntos o superconjuntos.

4. Si el subconjunto a analizar pertenece a la clase **Indeterminados**, se deberán almacenar todas las dominaciones contenidas en dicho subconjunto como parte de la información de aprendizaje. No obstante, dado que todos los elementos de esta clase son recíprocos a los elementos de la clase Testores Típicos, resulta irrelevante revisar cualquiera de sus superconjuntos.

4.3. Adaptaciones sobre algoritmos de búsqueda de testores típicos

Con el propósito de verificar la eficacia y eficiencia de la estrategia de aprendizaje propuesta sobre las dos clases de ABTT descritas (algoritmos externos e internos), se seleccionó un algoritmo de cada clase para adaptar sobre ellos la estrategia y poder verificar si existe una mejora en su desempeño como consecuencia de la misma. El criterio para la selección de estos algoritmos fue fecha de publicación (los más recientes). Como algoritmo externo se seleccionó el algoritmo Binario Recursivo (BR) y, como algoritmo interno se seleccionó Yablonsky y Compatibles (YYC).

4.3.1. Algoritmo Binario Recursivo (BR)

En esta sección se describe brevemente el funcionamiento del algoritmo BR para posteriormente explicar en forma detallada el proceso de adaptación de este algoritmo para incorporar la estrategia de aprendizaje.

Descripción del algoritmo original

Binario Recursivo (BR) [3], es un algoritmo externo de búsqueda de testores típicos diseñado esencialmente como una alternativa eficiente ante otros algoritmos externos, tales como LEX [5] y CT-EXT [13]. El algoritmo BR opera realizando un recorrido recursivo sobre el espacio de búsqueda y, aprovechando la naturaleza de la matriz básica, utiliza operaciones binarias para determinar si el subconjunto de rasgos a revisar es un testor típico; de no serlo, descarta todo subconjunto en el espacio que no conduzca a tal condición (aun testor no típico le sobran rasgos para ser un testor típico y, a un típico no testor le faltan rasgos).

A continuación, se revisan algunos conceptos básicos definidos en [3] para la descripción de algoritmo BR y su funcionamiento.

Definición 4.5 (Máscara de aceptación)**.** *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Denominamos **máscara de aceptación** de τ, denotada como ma_τ, a la n-tupla binaria en la cual su i-ésimo elemento toma el valor uno si el i-ésimo renglón en MB tiene al menos un 1 en las columnas correspondientes a los rasgos de τ. En otro caso toma el valor cero.*

Definición 4.6 (Máscara de compatibilidad)**.** *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Denominamos **máscara de compatibilidad** de τ, denotada como mc_τ, a la n-tupla binaria en la cual su i-ésimo elemento toma el valor uno, si el i-ésimo renglón en MB tiene exactamente un 1 en cualquiera de las columnas correspondientes a los rasgos de τ. En otro caso toma el valor cero.*

Proposición 4.1 (Cálculo de máscara de aceptación). *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos y $x \notin \tau$ un rasgo de MB. La máscara de aceptación del subconjunto $\tau + \{x\}$ se calcula como sigue:*

$$ma_{\tau+\{x\}} = ma_\tau \vee mc_x$$

Proposición 4.2 (Cálculo de máscara de compatibilidad). *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos y $x \notin \tau$ un rasgo de MB. La máscara de compatibilidad del subconjunto $\tau + \{x\}$ se calcula como sigue:*

$$mc_{\tau+\{x\}} = ((mc_\tau \otimes column_x) \wedge mc_\tau) \vee (\neg ma_\tau \wedge column_x)$$

En términos de BR, un testor es definido mediante la máscara de aceptación como sigue:

Definición 4.7 (Definición alternativa de testor en el contexto del algoritmo BR). *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Decimos que τ es un testor, si y solo si, $ma_\tau = (1, \ldots, 1)$.*

A partir de la definición anterior, se define testor típico en términos de BR como sigue:

Definición 4.8 (Definición alternativa de testor típico en el contexto del algoritmo BR). *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Decimos que τ es un testor típico, si y solo si, τ es un testor y para cada rasgo $x_i \in \tau$ existe un renglón r en MB, tal que el renglón r en la columna x_i tenga un 1 y 0 en todas las columnas correspondientes a los rasgos restantes en τ.*

Como se mencionó anteriormente, BR fue propuesto como una alternativa eficiente ante otros algoritmos externos de búsqueda de testores típicos; la principal razón de esto es que BR puede identificar mediante la propiedad denominada como *excluyente* a todos aquellos subconjuntos que no conducen a un testor típico, evitando revisar cualquier superconjunto de estos.

Definición 4.9. *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos y $x \notin \tau$ un rasgo de MB. Decimos que x_i es excluyente con respecto a τ si alguna de las siguientes dos condiciones se cumple:*

1. $ma_{\tau+\{x\}} = ma_\tau$
2. $\exists x_i \in \tau \mid mc_{\tau+\{x\}} \wedge column_{x_i} = (0, \ldots, 0)$

Antes de llevar a cabo una búsqueda sobre el espacio, se realiza un reordenamiento de los renglones y columnas de MB con el propósito de revisar la menor cantidad posible de subconjuntos y disminuir la probabilidad de hallar subconjuntos excluyentes. Este reordenamiento se logra colocando como primer renglón de MB a aquel renglón r con la menor cantidad de unos en la matriz y posteriormente colocando como primeras columnas de MB a aquellas donde el valor de r sea 1.

Como siguiente paso, se establece una estructura *Lista* que alberga el espacio de búsqueda actual que BR debe revisar durante su recorrido. Esta lista se inicializa con los conjuntos unitarios formados por los rasgos correspondientes a los unos del primer renglón en la matriz reordenada.

El algoritmo BR opera extrayendo recursivamente de *Lista* subconjuntos de rasgos, verificando tras cada extracción, las condiciones que estos cumplen. Dependiendo de las condiciones que dicho subconjunto exhiba se clasifica en alguno de los siguientes grupos:

a) **Testor Típico:** Si τ pertenece a esta categoría entonces es insertado al conjunto de testores típicos Ψ^*. El conjunto Ψ^* es la respuesta final del algoritmo.

b) **Excluyente:** Si τ pertenece a esta categoría implica que ninguno de sus descendientes será un testor típico. Por tal motivo, se descarta a τ y a todos sus superconjuntos que estén contenidos en *Lista*. Entiéndase por descendiente de τ, a todo conjunto $\tau \cup \{x\}$ donde x es el siguiente rasgo en el orden lexicográfico con respecto al orden de las columnas de MB.

c) **Candidato:** Si τ pertenece a esta categoría significa que alguno de sus descendientes tiene una alta probabilidad de ser testor típico.

La estructura básica del pseudocódigo para BR se muestra en el Algoritmo 1; sin embargo, el proceso de análisis del espacio de búsqueda se encuentra retratado en el Algoritmo 2. Este último requiere ciertas rutinas auxiliares para llevar a cabo el análisis del espacio de búsqueda; dichas funciones se describen a continuación:

- $TestorTipico(\tau)$: Es un predicado que devuelve VERDADERO si τ es un testor típico de la matriz MB y FALSO en caso contrario.
- $Excluyente(\tau)$: Es un predicado que devuelve VERDADERO si el último rasgo $x_i \in \tau$ es excluyente con respecto a $\tau \setminus x_i$ y FALSO en caso contrario.
- $Candidato(\tau)$: Es un predicado que devuelve VERDADERO si τ no es un testor típico ni un conjunto excluyente y FALSO en caso contrario.

Algoritmo 1: Algoritmo BR [Estructura básica]

Entrada: Matriz básica $MB_{(m \times n)}$.
Salida: Conjunto Ψ^* de todos los testores típicos en MB.

1 $\Psi^* \leftarrow \emptyset$
2 $Lista \leftarrow \emptyset$
3 Reordenamiento de renglones y columnas en MB.
4 Insertar como conjunto en *Lista* toda columna con valor 1 en el primer renglón de MB.
5 $\Psi^* \leftarrow Analizar(Lista)$

Algoritmo 2: Analizar($Lista$) [BR versión original]

Entrada: Una estructura $Lista = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$ con conjuntos de etiquetas de columnas de MB.

Salida: Conjunto Ψ^* de todos los testores típicos en MB.

```
1  si Lista = ∅ entonces
2      Regresar ∅
3  en otro caso
4      τ ← Pop(Lista)
5      si TestorTipico(τ) entonces
6          Remover todo x_i ∈ Lista talque τ ⊆ x_i.
7          Regresa τ ∪ Analizar(Lista).
8      si no, si Excluyente(τ) entonces
9          Remover todo x_i ∈ Lista talque τ ⊆ x_i.
10         Regresa Analizar(Lista).
11     en otro caso
12         Insertar en Lista todo descendiente de τ antes del primer elemento de cardinalidad uno.
           Regresa Analizar(Lista).
```

Adaptación de BR con la estrategia de aprendizaje

La idea original del algoritmo BR es discriminar la mayor cantidad posible de elementos del espacio de búsqueda, principalmente mediante el concepto de excluyente. No obstante, la adaptación parte de la misma idea, sólo que considerando las cuatro clases con las que se caracterizó el espacio de búsqueda y la información de aprendizaje que puede obtenerse de las mismas.

Fundamentalmente el Algoritmo 3 tiene la misma estructura que el Algoritmo 2, los cambios radican en las clases utilizadas en cada versión y las acciones a tomar de acuerdo a cada clase. En la versión original el algoritmo consideraba las clases testor típico, excluyente y candidato. Por el contrario, la versión adaptada considera las clases testor (no típico), típico (no testor), testor típico e indeterminados, recolectando la información de aprendizaje descrita en la estrategia, así como sus consideraciones.

4.3.2. Algoritmo Yablonsky y Compatibles (YYC)

En esta sección se describe brevemente el funcionamiento del algoritmo YYC para posteriormente explicar en forma detallada el proceso de adaptación de este algoritmo para incorporar la estrategia de aprendizaje.

Descripción del algoritmo original

Yablonsky & Compatibles (YYC) [19], es un algoritmo interno de búsqueda de testores típicos que analiza renglón por renglón la matriz MB, identificando de forma incremental el conjunto de testores típicos.

Algoritmo 3: Analizar($Lista$) [BR versión adaptada]

Entrada: Una lista $Lista = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$ con conjuntos de columnas de MB.
Salida: Conjunto Ψ^* de todos los testores típicos en MB.

```
1  Inicializar tabla de dominaciones D.
2  si Lista = ∅ entonces
3      Regresar ∅
4  en otro caso
5      τ ← Pop(Lista)
6      si Testor(τ) entonces
7          Agregar las dominaciones de τ en la tabla D.
8          Regresa Analizar(Lista).
9      si no, si Tipico(τ) entonces
10         Insertar en Lista todo descendiente de τ antes del primer elemento de cardinalidad uno.
11         Regresa Analizar(Lista).
12     si no, si TestorTipico(τ) entonces
13         Remover todo x_i ∈ Lista talque τ ⊆ x_i.
14         Regresa τ ∪ Analizar(Lista).
15     en otro caso
16         Agregar las dominaciones de τ en la tabla D.
17         Remover todo x_i ∈ Lista talque τ ⊆ x_i.
18         Regresa Analizar(Lista).
```

A continuación, se revisan algunos conceptos básicos definidos en [19] para la descripción del algoritmo YYC y su funcionamiento.

En primer lugar, se define el concepto de **conjunto compatible** que representa la irreducibilidad de un conjunto de rasgos.

Definición 4.10. *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Decimos que τ contiene un conjunto compatible, si y solo si después de aplicar algunas transformaciones elementales (reordenamiento de renglones o columnas) a $MB\,|_\tau$, ésta contiene una matriz identidad.*

Definición 4.11. *Sea $\tau = \{x_1, x_2, \ldots, x_s\}$ un subconjunto de rasgos. Decimos que τ es un testor típico, si y solo si $MB\,|_\tau$ contiene un conjunto compatible y además no incluye renglón alguno de con puros ceros.*

La idea fundamental del algoritmo YYC consiste en convertir la búsqueda de testores típicos en un proceso incremental, determinando los testores típicos mientras se analiza la matriz básica renglón por renglón.

Siguiendo la idea anterior, el algoritmo YYC comienza identificando el conjunto de testores típicos en el primer renglón de la matriz. A partir de ese momento, cada nuevo renglón que se analiza desencadena una actualización sobre el conjunto de testores típicos que se tiene hasta el renglón anterior. Si se conoce que un subconjunto de rasgos es testor típico hasta el renglón anterior y el nuevo renglón de la matriz tiene valor uno en la columna correspondiente de cualquiera de sus rasgos, entonces, el subconjunto sigue siendo testor típico. Por otro lado, si el nuevo renglón de la matriz tiene valor cero en todas las columnas correspondientes a todos los rasgos del subconjunto, entonces nuevos rasgos que tengan valor uno en el renglón actual deberán ser agregados al subconjunto con la finalidad de preservar la condición de testor. El Algoritmo 4 muestra el pseudocódigo del algoritmo YYC.

Algoritmo 4: Algoritmo YYC [Versión original]

Entrada: Matriz básica $MB_{(m \times n)}$.
Salida: Conjunto Ψ^* de todos los testores típicos en MB.

```
1  Ψ* ← ∅
2  para cada columna c_j, j = {1, 2, ..., n} talque mb_{1,c_j} = 1 hacer
3      Ψ* ← Ψ* ∪ {c_j}
4  para cada renglón r_i, i = {1, 2, ..., m} hacer
5      Ψ*_aux ← ∅
6      para cada testor τ_k ∈ Ψ* hacer
7          si ∃x_p ∈ τ_k talque mb_{r_i,x_p} = 1 entonces
8              Ψ*_aux ← Ψ*_aux ∪ τ_k
9          en otro caso
10             para cada columna c_q, q = {1, 2, ..., n} talque mb_{r_i,c_q} = 1 hacer
11                 si ConjuntoCompatible(τ_k, c_q) entonces
12                     Ψ*_aux ← Ψ*_aux ∪ {τ_k ∪ {c_p}}
13     Ψ* ← Ψ*_aux
```

Algoritmo 5: ConjuntoComplatible(τ_k, c_p)

Entrada: Subconjunto $\tau = \{x_1, x_2, \ldots, x_n\}$ de columnas de la matriz básica.
Entrada: Columna c_p de la matriz básica.
Salida: VERDADERO si existe un conjunto compatible en la submatriz definida por $\tau \cup \{c_p\}$.
FALSO en otro caso.

```
1  si |{r_s | r_s ∈ RefSM, Sum(r_s) = 1}| ≥ |τ ∪ {c_p}| entonces
2      Condición1 ← VERDADERO
3  RefSM ← |{r_s | r_s ∈ RefSM, Sum(r_s) = 1}|
4  si ∀c_k ∈ RefSM, Sum(c_k) = 1 entonces
5      Condición2 ← VERDADERO
6  Regresar (Condición1 AND Condición2)
```

Un paso crítico con respecto al desempeño del Algoritmo 4 es la búsqueda de los conjuntos compatibles. Existen varios algoritmos en la literatura que pueden ser utilizados para ese objetivo; sin embargo, con el propósito de hacer más eficiente al algoritmo YYC, sus autores proponen un algoritmo para determinar los conjuntos compatibles (véase Algoritmo 5). Este algoritmo utiliza una función auxiliar denominada $Sum(< vector >)$, donde vector puede ser un renglón o una columna de MB, devolviendo la suma de los elementos del vector proporcionado.

Adaptación de YYC con la estrategia de aprendizaje

La idea original del algoritmo YYC es establecer un procedimiento incremental para encontrar los testores típicos en una matriz básica. No obstante, el algoritmo termina estableciendo un procedimiento pseudoincremental, ya que para cada combinación de rasgos que el algoritmo genera resulta necesario volver a revisar renglones previamente analizados.

Con el propósito de generar un algoritmo completamente incremental, la idea fundamental de la adaptación es respetar en mayor media la estructura original del algoritmo, pero centrándose en utilizar la estrategia de aprendizaje como un medio para recopilar información útil de cada renglón analizado y posteriormente utilizarla para discriminar la mayor cantidad posible de elementos del espacio de búsqueda.

El Algoritmo 7 muestra el pseudocódigo de la adaptación de YYC. En esta versión para cada nuevo renglón de la matriz básica se actualiza la tabla de dominaciones (pasos 4 y 8). El proceso de actualización de la tabla de dominaciones se detalla en el Algoritmo 6. Otro de los cambios fundamentales que tiene la adaptación, es la sustitución del cálculo de conjuntos compatibles por una revisión a la tabla de dominaciones (paso 14), ya que es mucho más rápido y no requiere revisar renglones de la matriz previamente analizados. Estos cambios vuelven a esta adaptación un algoritmo totalmente incremental, pudiendo recibir la matriz renglón por renglón sin necesidad de conocerla completa.

Algoritmo 6: ActualizarDominaciones(R)

Entrada: Renglón $R = \{x_1, x_2, \ldots, x_n\}$ de la matriz básica.
Entrada: Tabla de dominaciones D, con entradas de la forma $x_i \rightarrow Dominadores$. Siendo $Dominadores$ la familia de conjuntos que cubren a x_i.
Salida: Tabla de dominaciones actualizada.

```
1 para cada x_i ∈ R hacer
2     si existe x_i como entrada en D entonces
3         para cada d ∈ Dominadores hacer
4             si d ∩ R entonces
5                 Mantener d en el conjunto Dominadores
6             en otro caso
7                 Combinar d con cada elemento en R \ {x_i} evitando generar superconjuntos de
                  otros dominadores
8     en otro caso
9         DomSimp ← Agregar cada elemento de R \ {x_i} como un conjunto individual
```

Algoritmo 7: Algoritmo YYC [Versión adaptada]

Entrada: Matriz básica $MB_{(m \times n)}$.
Salida: Conjunto Ψ^* de todos los testores típicos en MB.

```
1  Ψ* ← ∅
2  D ← NULL
3  para cada columna c_j, j = {1, 2, ..., n} talque mb_{1,c_j} = 1 hacer
4  |  D ← ActualizarDominaciones(r_1)
5  |  Ψ* ← Ψ* ∪ {c_j}
6  para cada renglón r_i, i = {1, 2, ..., m} hacer
7  |  Ψ*_aux ← ∅
8  |  D ← ActualizarDominaciones(r_i)
9  |  para cada testor τ_k ∈ Ψ* hacer
10 |  |  si ∃x_p ∈ τ_k talque mb_{r_i,x_p} = 1 entonces
11 |  |  |  Ψ*_aux ← Ψ*_aux ∪ τ_k
12 |  |  en otro caso
13 |  |  |  para cada columna c_q, q = {1, 2, ..., n} talque mb_{r_i,c_q} = 1 hacer
14 |  |  |  |  si ∄x ∈ τ | x no es dominado por algún otro z ∈ τ o algún σ ⊂ {τ \ x} entonces
15 |  |  |  |  |  Ψ*_aux ← Ψ*_aux ∪ {τ_k ∪ {c_p}}
16 |  |  Ψ* ← Ψ*_aux
```

Capítulo 5

Resultados experimentales

Con el propósito de discernir bajo qué familias de problemas la estrategia de aprendizaje beneficia en mayor medida a los ABTT, se diseñaron cuatro escenarios experimentales en los que se compara el desempeño de las versiones adaptadas de BR y YYC con respecto a sus versiones originales. Cabe mencionar, que el término **desempeño** utilizado en este trabajo esta compuesto por dos criterios: el número de subconjuntos de rasgos revisados por el ABTT (denominados como *Hits*) y el tiempo de ejecución empleado por la implementación del ABTT para encontrar el conjunto de testores típicos. Así, en términos de estos dos criterios, decimos que un ABTT tiene mejor desempeño que otro si tiene un menor conteo de *hits* o tiene un menor tiempo de ejecución.

Las familias de problemas que fueron seleccionadas para llevar a cabo la fase de experimentación de este trabajo fueron: matrices con crecimiento lineal en columnas, matrices con crecimiento exponencial en renglones, matrices con crecimiento proporcional y matrices identidad. Cada una de estas familias representa uno de los cuatro escenarios de experimentación.

Para cada escenario se diseñó un conjunto de matrices de prueba siguiendo la metodología descrita en [6]. Ésta metodología plantea un conjunto de operadores que permiten la generación de matrices básicas con dimensiones controladas en las que se conoce de antemano el número total de testores típicos; las matrices generadas mediante esta metodología permiten la validación y evaluación de los ABTT.

Los resultados obtenidos en cada escenario se concentran en dos tablas: la tabla de resultados para las versiones de BR y tabla de resultados para las versiones de YYC. Cada una de estas tablas está dividida en 4 secciones: información sobre la matriz de prueba (dimensiones, número de testores típicos, etc.), resultados de la versión original del algoritmo, resultados de la versión adaptada del algoritmo y mejoras en desempeño. En las secciones donde se reportan los resultados de los algoritmos se consideran principalmente dos rubros: el número de *hits* y el tiempo de ejecución en segundos. Sin embargo, en la sección que corresponde a la versión adaptada del algoritmo, existen dos columnas más, una para almacenar las dominaciones detectadas en cada experimento y otra para el número de conjuntos descartados por la información de aprendizaje (testores típicos y dominaciones). Finalmente, la sección de mejoras en desempeño está compuesta por dos columnas, una para almacenar la diferencia en *hits* que existe entre la versión adaptada y la versión original, y otra, para almacenar la diferencia en tiempo de ejecución que existe entre la versión adaptada y la versión original.

A continuación, se describe cada uno de los escenarios y los resultados obtenidos por los algoritmos en cada uno.

5.1. Escenario 1: Matrices con crecimiento lineal en columnas

En este escenario se muestra cuál es el beneficio que aporta la estrategia de aprendizaje a los ABTT ante matrices con crecimiento lineal en el número de columnas, mediante la comparación del desempeño de las versiones adaptadas de BR y YYC con respecto sus versiones originales.

Los experimentos en este escenario están compuestos por un conjunto de diez matrices sintéticas, generadas mediante el uso sucesivo del **operador concatenación (φ)** descrito en [6] (véase Apéndice A), dando como resultado matrices con un crecimiento lineal en el número de columnas. Este tipo de matrices tienen la peculiaridad contienen muchas columnas idénticas, por lo cual se intuye que estas matrices albergan una gran cantidad de dominaciones de cardinalidad uno.

En la Tabla 5.1 se reportan los resultados mediante los que se comparan ambas versiones del algoritmo BR. Estos resultados muestran que para cada experimento que se llevó a cabo, la versión adaptada de BR presenta una mejora sustancial, tanto en *hits* como en tiempo de ejecución, con respecto a la versión original. En las Figuras 5.1 y 5.2 puede visualizarse el contraste, en *hits* y tiempo de ejecución, que hay entre ambas versiones del algoritmo BR.

Tabla 5.1: Resultados de las dos versiones de BR ante matrices con crecimiento lineal en columnas

No.	Renglones	Columnas	TT	BR original		BR adaptado				Diferencias	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	4	96	66,304	289,296	44.5023	85,248	13.6041	15,688	204,048	204,048	30.8982
2	4	102	84,388	366,469	66.8144	106,930	19.9469	18,581	259,539	259,539	46.8675
3	4	108	105,948	458,154	98.0651	132,516	28.0614	21,807	325,638	325,638	70.0037
4	4	114	131,404	566,067	141.2624	162,450	39.8358	25,384	403,617	403,617	101.4266
5	4	120	161,200	692,020	199.5205	197,200	56.0630	29,330	494,820	494,820	143.4575
6	4	126	195,804	837,921	277.6620	237,258	76.1960	33,663	600,663	600,663	201.4660
7	4	132	235,708	1,005,774	381.1723	283,140	104.2205	38,401	722,634	722,634	276.9518
8	4	138	281,428	1,197,679	516.9500	335,386	140.2407	43,562	862,293	862,293	376.7092
9	4	144	333,504	1,415,832	690.9740	394,560	185.9480	49,164	1,021,272	1,021,272	505.0260
10	4	150	392,500	1,662,525	913.6904	461,250	244.2169	55,225	1,201,275	1,201,275	669.4735

Esta familia de problemas representa un caso de estudio poco favorable para los algoritmos externos, ya que a medida que aumenta el número de columnas en la matriz el espacio de búsqueda crece en forma desmedida. A pesar de ello, como se observa en los resultados, la estrategia de aprendizaje mejora de manera dramática el desempeño de BR, principalmente por dos razones. En primer lugar, por el número de dominaciones que existen en las matrices de prueba, ya que éstas permiten descartar combinaciones que no conducen a un testor típico. En segundo lugar, el gran número de testores típicos que contienen las matrices de prueba, ya que estos igualmente ayudan a discriminar todos los superconjuntos de si mismos que haya en el espacio de búsqueda.

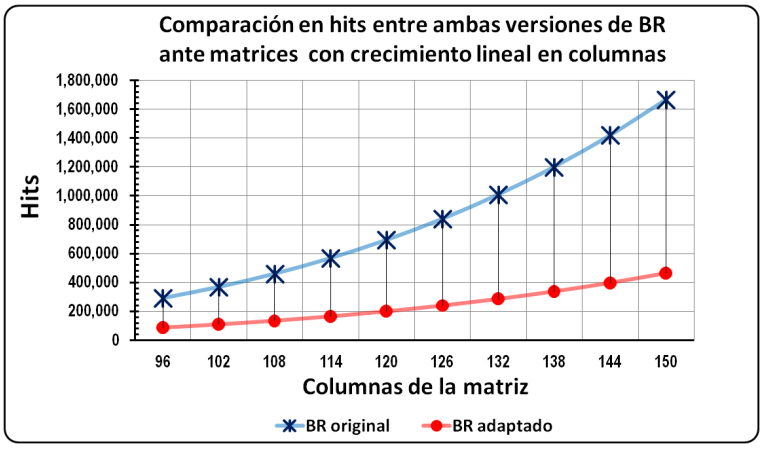


Figura 5.1: Gráfica comparativa del número *hits* entre ambas versiones de BR ante matrices con crecimiento lineal en columnas

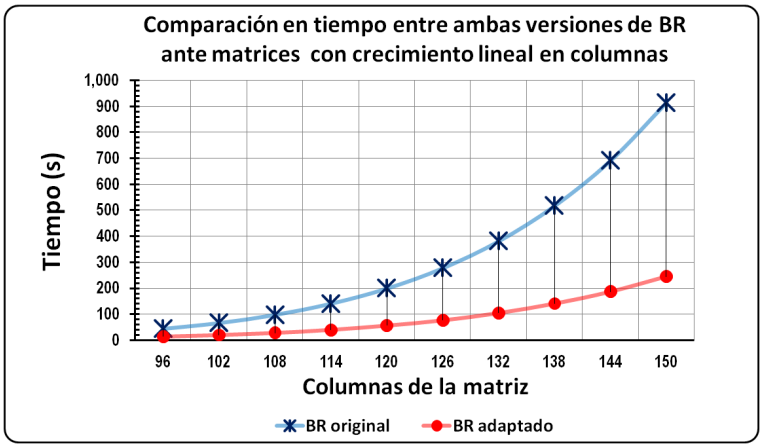


Figura 5.2: Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento lineal en columnas

En la Tabla 5.2 se reportan los resultados con los que se comparan ambas versiones del algoritmo YYC. Estos resultados muestran que para cada experimento que se llevó a cabo, la versión adaptada de YYC presenta una mejora significativa en *hits* con respecto a la versión original, como se ve en la Figura 5.3. Por el contrario, en cuestión del tiempo de ejecución la versión adaptada de YYC no presenta ningún tipo de mejora, sino todo lo contrario, presenta un déficit cómo puede verse en la Figura 5.4.

Es claro que la adaptación del algoritmo YYC presenta una importante deficiencia en el tiempo de ejecución. Dicha deficiencia es debido a que la estrategia de aprendizaje tiene un costo computacional alto ante matrices con muy pocos renglones, resultando innecesaria su aplicación en estos casos. Además, debemos recordar que YYC es un algoritmo que analiza la matriz de entrada renglón por renglón, por lo que las matrices que afectan su desempeño son aquellas que tienen un gran número de renglones y, es sobre éstas matrices, donde resulta conveniente aplicar la estrategia de aprendizaje. No obstante, el conjunto de matrices de prueba con el que se realizó los experimentos de este escenario,

Tabla 5.2: Resultados de las dos versiones de YYC ante matrices con crecimiento lineal en columnas

No.	Renglones	Columnas	TT	YYC original		YYC adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	4	96	66,304	206,112	0.1484	70,688	48.4823	151,200	135,424	135,424	-48.3339
2	4	102	84,388	261,868	0.2116	89,624	89.7392	190,927	172,244	172,244	-89.5276
3	4	108	105,948	328,248	0.2450	112,140	138.0451	238,032	216,108	216,108	-137.8002
4	4	114	131,404	406,524	0.3000	138,662	177.2293	293,379	267,862	267,862	-176.9293
5	4	120	161,200	498,040	0.3859	169,640	257.3424	357,880	328,400	328,400	-256.9564
6	4	126	195,804	604,212	0.4596	205,548	432.3378	432,495	398,664	398,664	-431.8782
7	4	132	235,708	726,528	0.5543	246,884	527.9854	518,232	479,644	479,644	-527.4311
8	4	138	281,428	866,548	0.6746	294,170	759.5387	616,147	572,378	572,378	-758.8641
9	4	144	333,504	1,025,904	0.7943	347,952	1,014.3444	727,344	677,952	677,952	-1,013.5501
10	4	150	392,500	1,206,300	0.9748	408,800	1,182.0754	852,975	797,500	797,500	-1,181.1006

contiene matrices con muy pocos renglones, haciendo que la estrategia de aprendizaje decrementara el desempeño de la adaptación de YYC en tiempo de ejecución con respecto al de su versión original; dicho decremento puede observarse en la columna **Tiempo (s)** de la sección **Mejoras** en la Tabla 5.2.

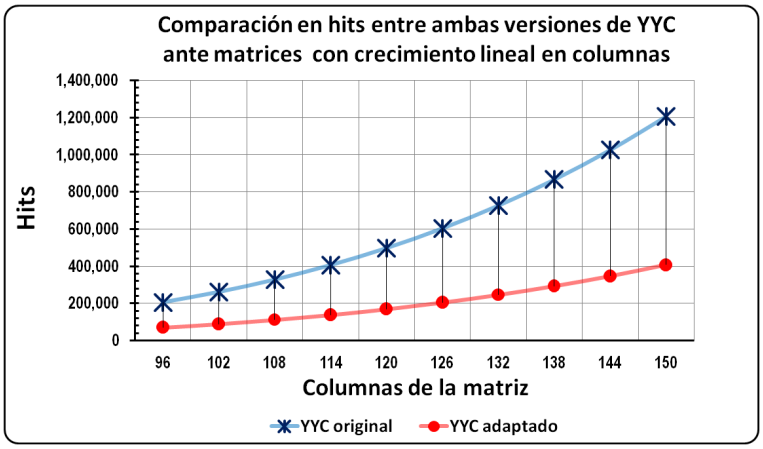


Figura 5.3: Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento lineal en columnas

En las Figuras 5.5 y 5.6 se muestran las gráficas comparativas de los resultados obtenidos tanto por las versiones originales como por las versiones adaptadas de los algoritmos BR y YYC. En estas gráficas puede observarse que tanto la adaptación de BR como la de YYC obtuvieron un mejor desempeño en *hits* que sus versiones originales. Sin embargo, en tiempo de ejecución únicamente la adaptación de BR obtuvo un mejor desempeño al incorporar la estrategia de aprendizaje.

Por otro lado, también es posible observar que la versión original del algoritmo YYC obtuvo el menor tiempo de ejecución para todas las matrices de prueba, incluso por debajo de las versiones adaptadas de ambos algoritmos.

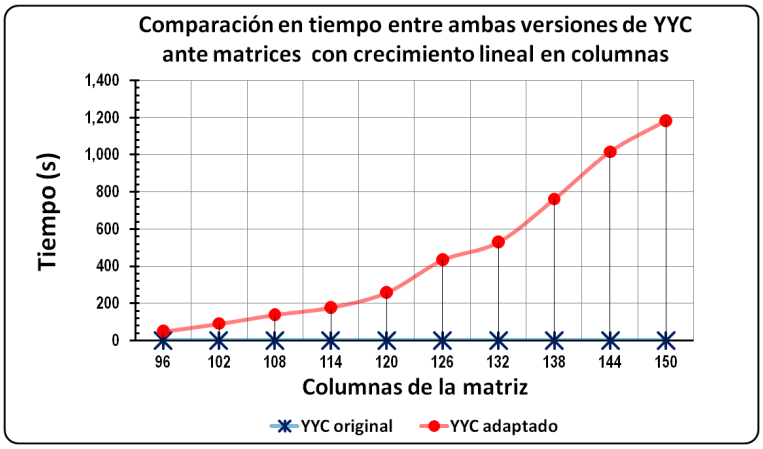


Figura 5.4: Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento lineal en columnas

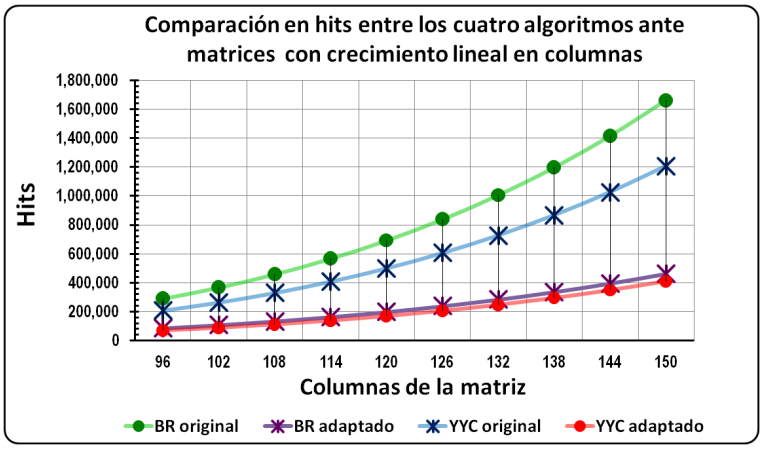


Figura 5.5: Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento lineal en columnas

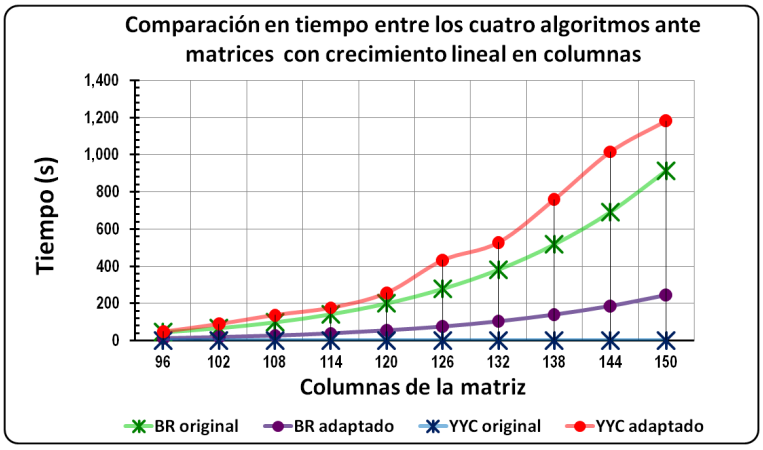


Figura 5.6: Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento lineal en columnas

5.2. Escenario 2: Matrices con crecimiento exponencial en renglones

En este escenario se muestra cuál es el beneficio que aporta la estrategia de aprendizaje a los ABTT ante matrices con crecimiento exponencial en el número de renglones, mediante la comparación del desempeño de las versiones adaptadas de BR y YYC con respecto sus versiones originales.

Los experimentos en este escenario están compuestos por un conjunto de diez matrices sintéticas, generadas mediante el uso sucesivo del **operador combinación (θ)** descrito en [6] (véase Apéndice A), dando como resultado matrices con un crecimiento exponencial en el número de renglones. En este tipo de matrices cuanto más se incrementa el número de renglones más combinables se vuelven sus columnas, por lo cual se intuye que estas matrices tienen una alta probabilidad de contener dominaciones de cardinalidad mayor a uno.

Tabla 5.3: Resultados de las dos versiones de BR ante matrices con crecimiento exponencial en renglones

No.	Renglones	Columnas	TT	BR original		BR adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	2	4	2	4	0.00006	4	0.00013	0	0	0	-0.00007
2	4	8	4	31	0.00010	19	0.00011	16	12	12	-0.00001
3	8	12	6	166	0.00041	97	0.00057	220	69	69	-0.00016
4	16	16	8	775	0.00427	391	0.00392	928	384	384	0.00035
5	32	20	10	3,394	0.04309	1,587	0.03275	6,514	1,807	1,807	0.01035
6	64	24	12	14,371	0.52801	6,339	0.30987	34,632	8,032	8,032	0.21814
7	128	28	14	59,686	6.52980	25,881	3.58238	235,640	33,805	33,805	2.94742
8	256	32	16	244,975	82.97990	105,455	40.60521	1,367,648	139,520	139,520	42.37469
9	512	36	18	997,834	1,153.36935	430,247	528.42907	8,355,902	567,587	567,587	624.94028
10	1,024	40	20	4,043,611	17,556.93848	1,747,211	6,742.74599	47,631,416	2,296,400	2,296,400	10,814.19249

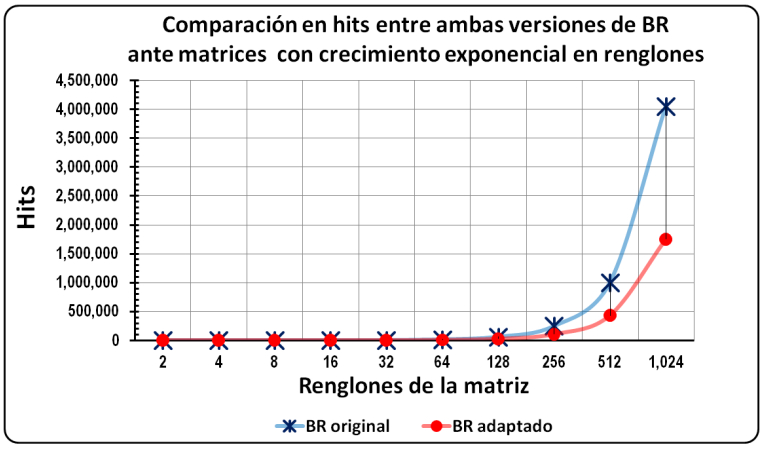


Figura 5.7: Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento exponencial en renglones

En la Tabla 5.3 se reportan los resultados mediante los cuales se comparan las dos versiones del algoritmo BR. Estos resultados muestran que en cada experimento realizado la versión adaptada de BR exhibe un mejor desempeño en *hits* que su versión original, tal y como puede verse en la Figura 5.7. Por otra parte, en los experimentos 1, 2 y 3, la adaptación de BR presenta un déficit en el tiempo

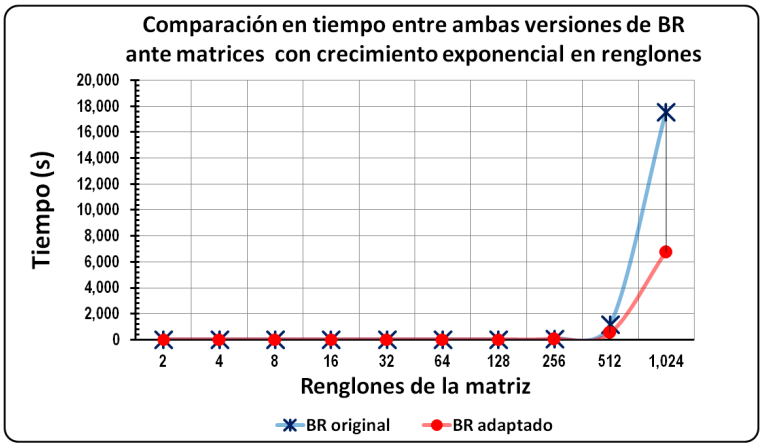


Figura 5.8: Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento exponencial en renglones

de ejecución con respecto a su versión original. No obstante, al igual que en el escenario anterior, estos casos se manifiestan principalmente en matrices pequeñas donde la aplicación de la estrategia de aprendizaje no resulta necesaria. Sin embargo, a medida que la matriz crece, comienza a hacerse notoria una mejora en el desempeño por parte de la versión adaptada de BR. Tal comportamiento puede verse en la Figura 5.8 y con mayor detalle en la columna **Tiempo (s)** de la sección **Mejoras** en la Tabla 5.3.

En la Tabla 5.4 se reportan los resultados mediante los cuales se comparan las dos versiones del algoritmo YYC. Examinando los resultados, es posible notar que, en términos de *hits*, la adaptación de YYC tiene un mejor desempeño en comparación con el de su versión original, como puede verse en la Figura 5.9. Por el contrario, en términos de del tiempo de ejecución, la adaptación de YYC logra un mejor desempeño respecto a su versión original a partir del experimento número 8, como se ve en la Figura 5.10.

Tabla 5.4: Resultados de las dos versiones de YYC ante matrices con crecimiento exponencial en renglones

No.	Renglones	Columnas	TT	YYC original		YYC adaptado				Diferencias	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	2	4	2	3	0.0001	3	0.0001	2	0	0	-0.00005
2	4	8	4	12	0.0001	7	0.0001	22	5	5	-0.00001
3	8	12	6	36	0.0001	13	0.0001	69	23	23	-0.00003
4	16	16	8	96	0.0002	23	0.0003	164	73	73	-0.00016
5	32	20	10	240	0.0006	41	0.0012	355	199	199	-0.00056
6	64	24	12	576	0.0027	75	0.0040	750	501	501	-0.00129
7	128	28	14	1,344	0.0119	141	0.0253	1,589	1,203	1,203	-0.01341
8	256	32	16	3,072	0.0599	271	0.0397	3,400	2,801	2,801	0.02019
9	512	36	18	6,912	0.2597	529	0.1155	7,335	6,383	6,383	0.14418
10	1,024	40	20	15,360	1.2005	1,043	0.3259	15,890	14,317	14,317	0.87462

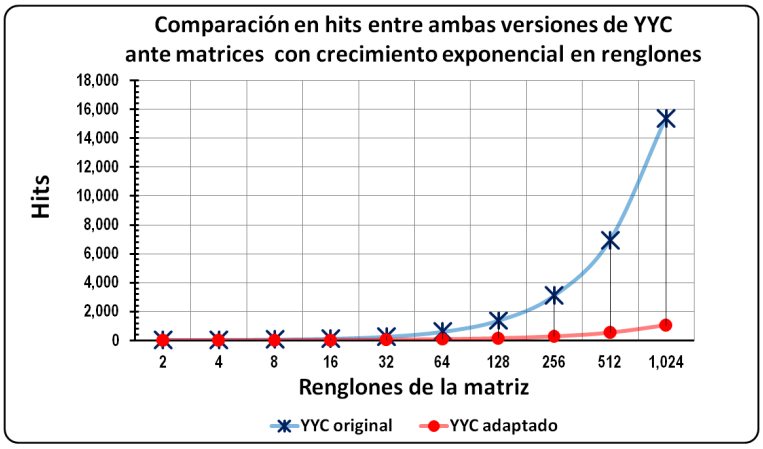


Figura 5.9: Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices con crecimiento exponencial en renglones

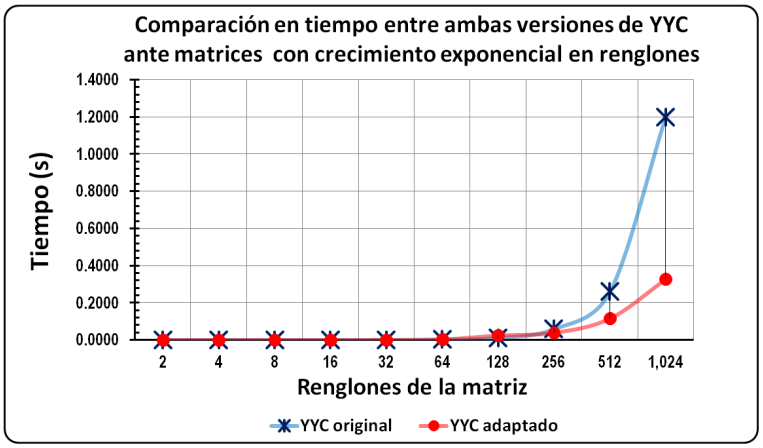


Figura 5.10: Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento exponencial en renglones

Esta familia de problemas representa un caso de estudio poco favorable para los algoritmos internos, especialmente para el algoritmo YYC, ya que como se ha mencionado anteriormente, este algoritmo analiza la matriz de entrada renglón por renglón y, a medida que la matriz tiene una mayor cantidad de renglones mayor es el costo computacional para analizarla. A pesar de ello, como se observa en los resultados, la estrategia de aprendizaje logra que la versión adaptada de YYC tenga un mejor desempeño que su versión original, lo cual ocurre a partir del experimento número 8 donde el desempeño de la versión original de YYC comienza a disminuir conforme la matriz crece y es donde precisamente la estrategia beneficia al algoritmo.

En las Figuras 5.11 y 5.12 se muestran las gráficas comparativas de los resultados obtenidos tanto por las versiones originales como por las versiones adaptadas de los algoritmos BR y YYC. En estas gráficas se puede observar que a pesar de ser un escenario poco favorable para YYC, el desempeño tanto de BR como el de su adaptación se ve afectado frente esta familia de problemas.

Por otra parte, también se puede observar que el beneficio que aporta la estrategia de aprendizaje es más notorio en los algoritmos externos que en los internos. Esto es producto del tamaño de las matrices que fueron utilizadas para llevar a cabo los experimentos de este escenario, ya que son instancias muy pequeñas para YYC y su adaptación, por lo cual, el beneficio que conlleva la estrategia para estos algoritmos comenzó a ser notorio hasta que las matrices eran lo suficientemente grandes, por lo que podemos intuir, que si se hicieran experimentos con matrices lo suficientemente grandes el beneficio que aporta la estrategia sería aún más notorio.

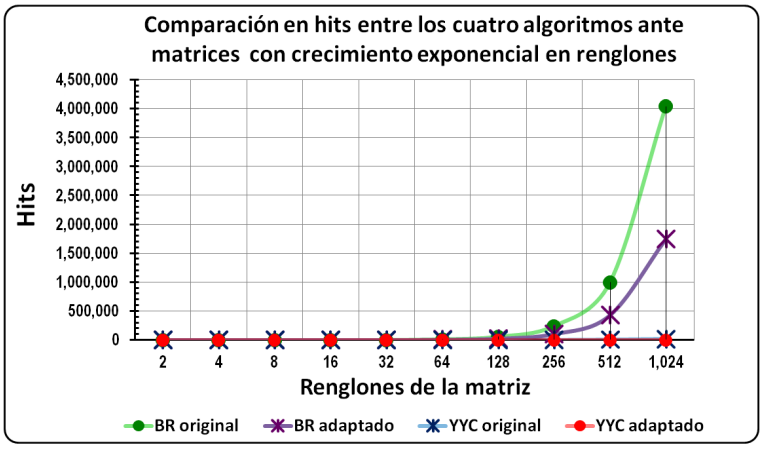


Figura 5.11: Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento exponencial en renglones

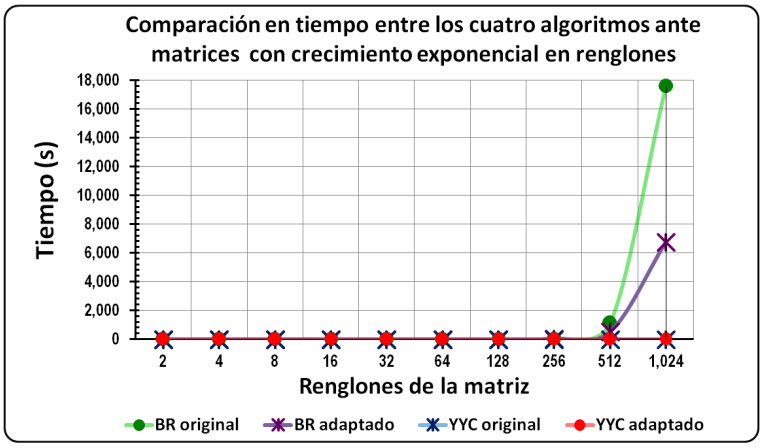


Figura 5.12: Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento exponencial en renglones

5.3. Escenario 3: Matrices con crecimiento proporcional

En este escenario se muestra cuál es el beneficio que aporta la estrategia de aprendizaje a los ABTT ante matrices con crecimiento proporcional, mediante la comparación del desempeño de las versiones adaptadas de BR y YYC con respecto sus versiones originales.

Los experimentos en este escenario están compuestos por un conjunto de diez matrices sintéticas, generadas mediante el uso sucesivo del **operador desplazamiento (γ)** descrito en [6] (véase Apéndice A), dando como resultado matrices con un incremento proporcional tanto en renglones como en columnas. Este tipo de matrices se caracteriza por tener una densidad de unos mínima, lo cual hace que sus columnas sean altamente combinables y reduzcan la probabilidad de contener dominaciones.

En la Tabla 5.5 se reportan los resultados obtenidos por las dos versiones del algoritmo BR. Estos resultados muestran que en cada experimento que pudo completarse, la versión adaptada de BR exhibe un mejor desempeño en *hits* que su versión original, tal como se ve en la Figura 5.13. Por otra parte, en los experimentos 1 y 2, la adaptación de BR presenta un déficit en el tiempo de ejecución con respecto a su versión original, debido a que son instancias muy pequeñas y la estrategia no representa ningún beneficio. Sin embargo, a medida que la matriz crece, comienza a hacerse notoria la mejora en el desempeño por parte de la adaptación de BR, tal como puede verse en la Figura 5.14 y con mayor detalle en la columna **Tiempo (s)** de la sección **Mejoras** en la Tabla 5.5.

Tabla 5.5: Resultados de las dos versiones de BR ante matrices con crecimiento proporcional

No.	Renglones	Columnas	TT	BR original		BR adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	2	4	2	4	0.00006	4	0.00013	0	0	0	-0.00007
2	4	8	4	30	0.00010	24	0.00015	4	6	6	-0.00005
3	6	12	8	174	0.00071	133	0.00044	6	41	41	0.00026
4	8	16	16	1,038	0.01089	782	0.00712	8	256	256	0.00377
5	10	20	32	6,222	0.30779	4,671	0.21321	10	1,551	1,551	0.09458
6	12	24	64	37,326	8.49921	28,000	5.94410	12	9,326	9,326	2.55511
7	14	28	128	223,950	291.71619	167,969	193.97666	14	55,981	55,981	97.73953
8	16	32	256	1,343,694	10,721.02206	1,007,778	6,796.02460	16	335,916	335,916	3,924.99746
9	18	36	512	-	-	-	-	-	-	-	-
10	20	40	1,024	-	-	-	-	-	-	-	-

Del conjunto de matrices que fue utilizado para llevar acabo los experimentos de este escenario, hubo algunas matrices para las que algoritmo BR y su adaptación no pudieron dar una solución al problema incluso después de haber transcurrido 15 horas de ejecución, por tal motivo, se tuvo que interrumpir la ejecución del algoritmo y prescindir de su solución. Esto deja claro dos cosas. En primer lugar, que esta familia de problemas representa un caso nada favorable para los algoritmos externos, ya que al tener matrices con columnas altamente combinables se genera la gran mayoría del espacio de búsqueda habiendo la necesidad de revisar todos los subconjuntos generados. En segundo lugar, que la estrategia de aprendizaje a pesar de mejorar el desempeño del algoritmo BR, tal como se ve las Figuras 5.13 y 5.14, no supone un beneficio importante para estos algoritmos frente a esta familia de problemas.

En la Tabla 5.6 se reportan los resultados obtenidos por las dos versiones del algoritmo YYC. Examinando los resultados podemos notar que, en términos de *hits*, la adaptación de YYC revisa

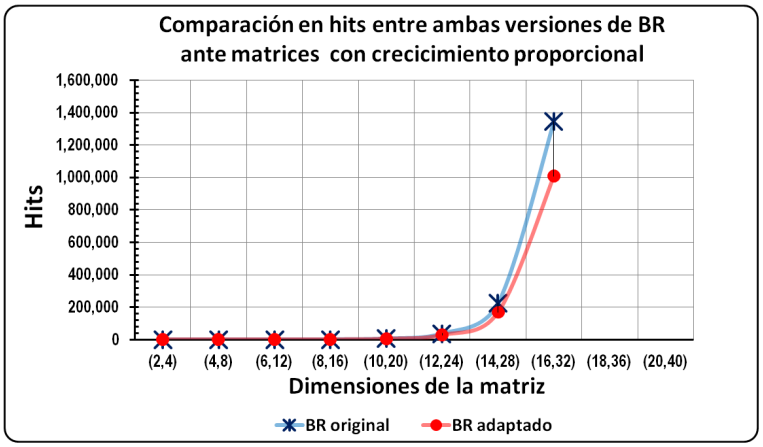


Figura 5.13: Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices con crecimiento proporcional

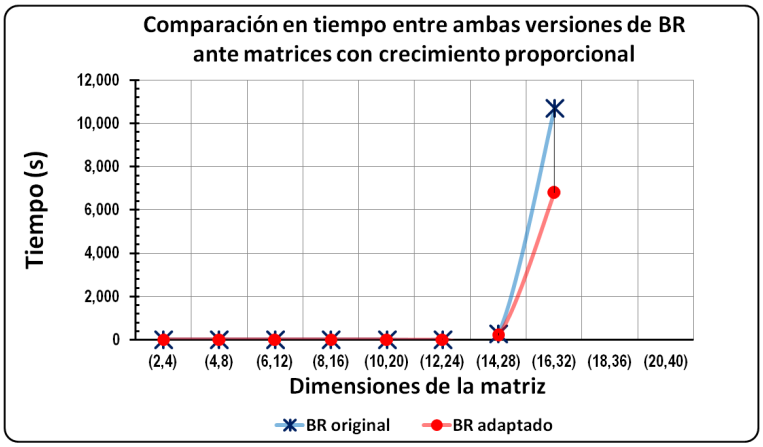


Figura 5.14: Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices con crecimiento proporcional

exactamente el mismo número de conjuntos que su versión original, tal como se ve en la Figura 5.15. Por el contrario, en términos del tiempo de ejecución, la adaptación de YYC obtiene un mejor desempeño respecto a su versión original, como se ve en la Figura 5.16. El hecho de que la versión adaptada del algoritmo YYC revise exactamente el mismo número de conjuntos, aunque con una ligera reducción en el tiempo de ejecución, implica que la estrategia de aprendizaje no representa un beneficio importante para este algoritmo ante esta familia de problemas.

En las Figuras 5.17 y 5.18 se muestran las gráficas comparativas de los resultados obtenidos tanto por las versiones originales como por las versiones adaptadas de los algoritmos BR y YYC. En estas gráficas lo primero que se puede observarse es, que a pesar de no haber experimentado con matrices muy grandes, hay casos en los algunos algoritmos no son capaces de devolver una solución en un tiempo razonable incluso aún con la estrategia de aprendizaje, tal es el caso de BR y su adaptación.

Además, otra de las cosas que también se puede notar es que los algoritmos internos, como YYC

Tabla 5.6: Resultados de las dos versiones de YYC ante matrices con crecimiento proporcional

No.	Renglones	Columnas	TT	YYC original		YYC adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	2	4	2	3	0.00010	3	0.00006	2	0	0	0.00004
2	4	8	4	9	0.00006	9	0.00004	4	0	0	0.00002
3	6	12	8	21	0.00007	21	0.00003	6	0	0	0.00004
4	8	16	16	45	0.00015	45	0.00004	8	0	0	0.00011
5	10	20	32	93	0.00039	93	0.00004	10	0	0	0.00035
6	12	24	64	189	0.00110	189	0.00006	12	0	0	0.00104
7	14	28	128	381	0.00299	381	0.00010	14	0	0	0.00288
8	16	32	256	765	0.00769	765	0.00020	16	0	0	0.00749
9	18	36	512	1,533	0.01900	1,533	0.00040	18	0	0	0.01860
10	20	40	1,024	3,069	0.02715	3,069	0.00085	20	0	0	0.02630

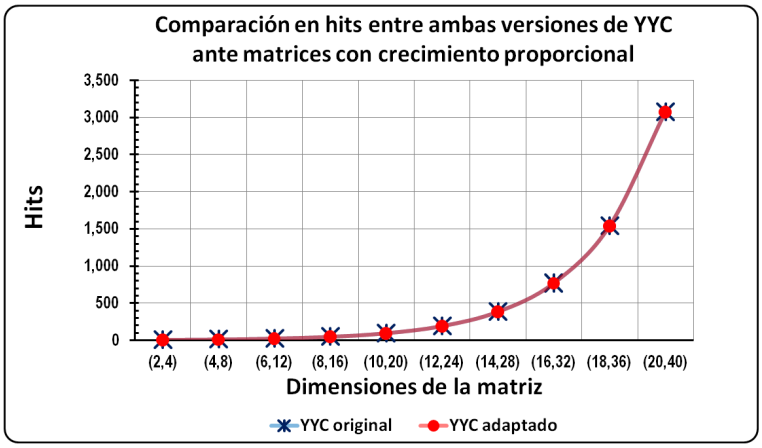


Figura 5.15: Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices con crecimiento proporcional

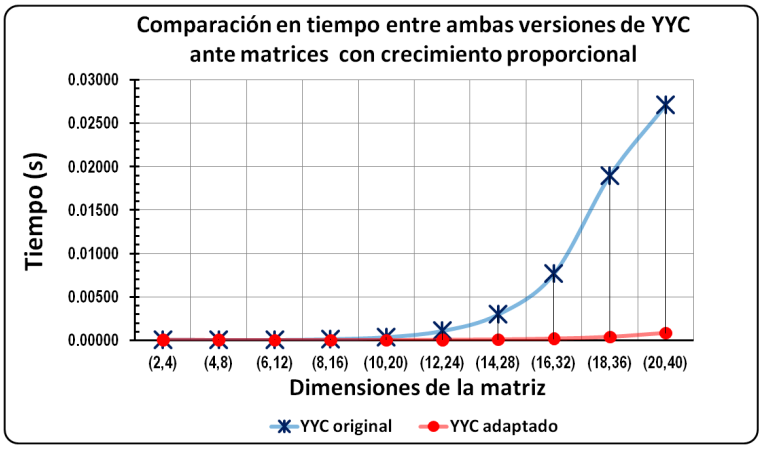


Figura 5.16: Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices con crecimiento proporcional

y su adaptación, son superiores en desempeño ante esta familia de problemas y por lo que se puede intuir son capaces de hacer frente a instancias mucho más grandes de esta familia.

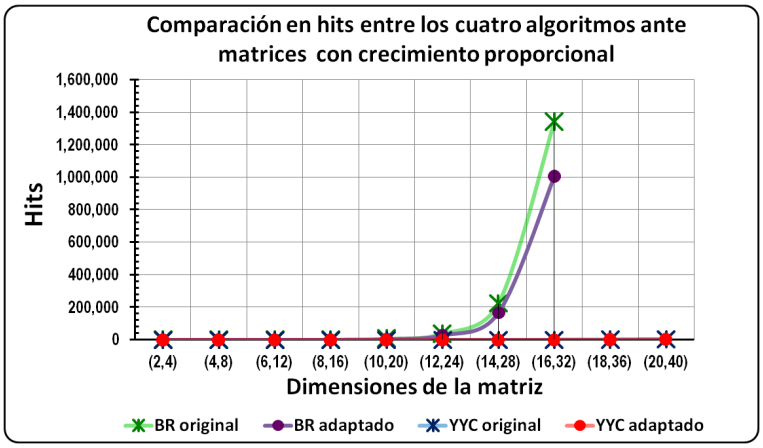


Figura 5.17: Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices con crecimiento proporcional

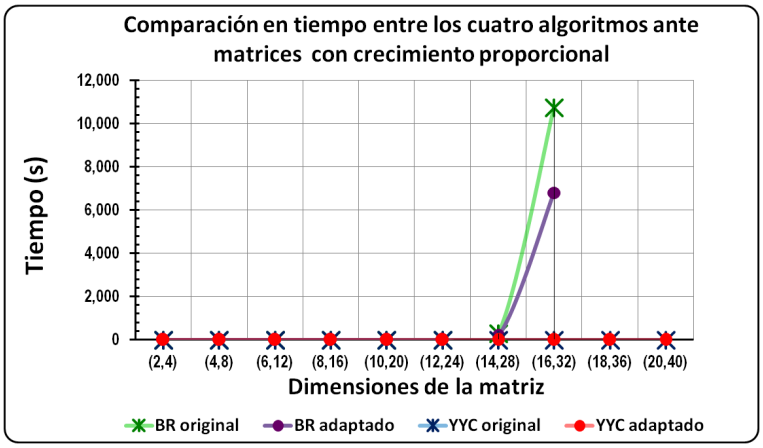


Figura 5.18: Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices con crecimiento proporcional

5.4. Escenario 4: Matrices identidad

En este escenario se muestra cuál es el beneficio que aporta la estrategia de aprendizaje a los ABTT ante matrices identidad, mediante la comparación del desempeño de las versiones adaptadas de BR y YYC con respecto sus versiones originales.

Los experimentos de este escenario se componen por un conjunto de diez matrices identidad. Estas matrices resultan ser un caso de estudio formidable con las cuales probar ABBT, ya que sin importar

el tamaño de la matriz ésta siempre tendrá solo un testor típico compuesto por el conjunto total de columnas. Además, cualquier subconjunto de columnas en la matriz pertenece a la clase **típico (no testor)**, clase que se caracteriza por no proporcionar información de aprendizaje.

Tabla 5.7: Resultados de las dos versiones de BR ante matrices identidad

No.	Renglones	Columnas	TT	BR original		BR adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	12	12	1	2,048	0.0445	2,048	0.0380	0	0	0	0.0064
2	13	13	1	4,096	0.1640	4,096	0.1027	0	0	0	0.0613
3	14	14	1	8,192	0.4980	8,192	0.3758	0	0	0	0.1221
4	15	15	1	16,384	1.6436	16,384	1.4077	0	0	0	0.2359
5	16	16	1	32,768	5.7017	32,768	5.4313	0	0	0	0.2704
6	17	17	1	65,536	21.1248	65,536	20.6961	0	0	0	0.4287
7	18	18	1	131,072	81.2013	131,072	78.7667	0	0	0	2.4347
8	19	19	1	262,144	347.9177	262,144	311.4950	0	0	0	36.4227
9	20	20	1	524,288	1,426.4357	524,288	1,305.6706	0	0	0	120.7651
10	21	21	1	1,048,576	5,936.5569	1,048,576	5,082.4512	0	0	0	854.1056

En los resultados que se reportan en las Tablas 5.7 y 5.8, puede observarse que las versiones adaptadas tanto de BR como de YYC no almacenaron ninguna dominación y tampoco descartaron algún elemento del espacio de búsqueda, lo cual se atribuye a la falta de información de aprendizaje en las matrices identidad. Por esta razón, ambos algoritmos presentan el mismo conteo de *hits* que sus versiones originales, ya que revisaron exactamente los mismos elementos del espacio de búsqueda. En las Figuras 5.19 y 5.21 puede visualizarse este comportamiento.

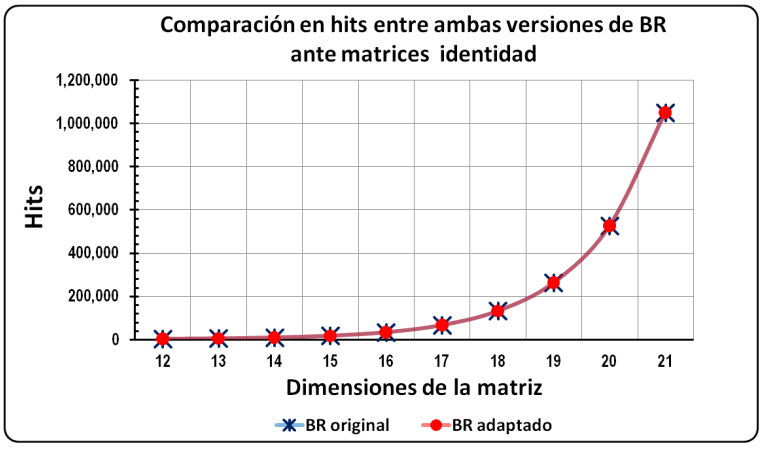


Figura 5.19: Gráfica comparativa de *hits* entre ambas versiones de BR ante matrices identidad

Como se muestra en la Figura 5.20, el desempeño en tiempo de ejecución de la versión adaptada de BR es ligeramente mejor que el de su versión original. Dado que en la gráfica que se presenta es poco visible esta mejora, se puede verificar en la columna ***Hits*** de la sección **Mejoras** en la Tabla 5.7.

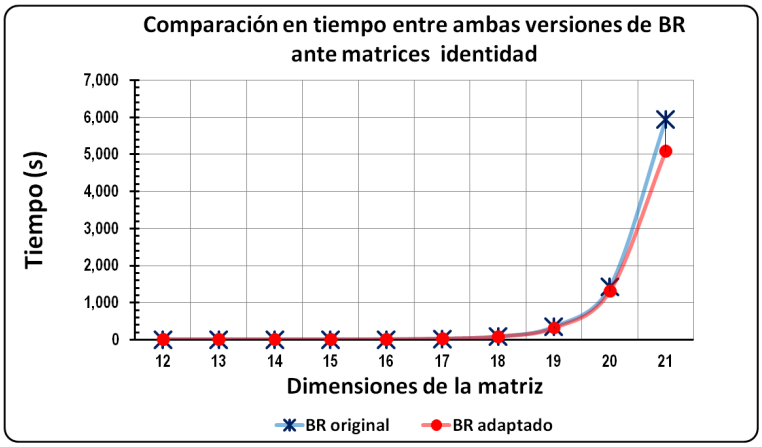


Figura 5.20: Gráfica comparativa del tiempo de ejecución entre ambas versiones de BR ante matrices identidad

Tabla 5.8: Resultados de las dos versiones de YYC ante matrices identidad

No.	Renglones	Columnas	TT	YYC original		YYC adaptado				Mejoras	
				Hits	Tiempo (s)	Hits	Tiempo (s)	Dominaciones	Descartados	Hits	Tiempo (s)
1	12	12	1	12	0.00009	12	0.00003	0	0	0	0.00006
2	13	13	1	13	0.00009	13	0.00003	0	0	0	0.00006
3	14	14	1	14	0.00010	14	0.00003	0	0	0	0.00007
4	15	15	1	15	0.00012	15	0.00004	0	0	0	0.00008
5	16	16	1	16	0.00013	16	0.00004	0	0	0	0.00009
6	17	17	1	17	0.00014	17	0.00004	0	0	0	0.00010
7	18	18	1	18	0.00016	18	0.00004	0	0	0	0.00012
8	19	19	1	19	0.00018	19	0.00004	0	0	0	0.00014
9	20	20	1	20	0.00020	20	0.00004	0	0	0	0.00016
10	21	21	1	21	0.00031	21	0.00004	0	0	0	0.00027

A diferencia de la adaptación de BR, el desempeño de la versión adaptada de YYC con respecto a su versión original es mucho mayor, tal como se muestra en la Figura 5.22. Esto se atribuye a la falta de dominaciones en la matriz identidad, ya que antes de generar una nueva combinación de columnas, la versión adaptada de YYC revisa las dominaciones que han sido almacenadas hasta el momento para no formar combinaciones erróneas, de manera que, al no haber alguna dominación entre las columnas de una matriz identidad, la combinación se realiza automáticamente, lo cual se traduce en una importante mejora del tiempo de ejecución con respecto a la versión original.

En las Figuras 5.23 y 5.24 se muestran las gráficas comparativas de los resultados obtenidos tanto por las versiones originales como por las versiones adaptadas de los algoritmos BR y YYC. En estas gráficas se observa que, ante matrices identidad, los algoritmos internos como YYC son superiores en desempeño a los algoritmos externos como BR, incluso aún sin estrategia de aprendizaje, ya que los algoritmos internos trabajan con los unos contenidos en la matriz y, por tanto, únicamente revisan la diagonal principal de la matriz como se puede ver al comparar la columna ***Hits***, de las dos versiones de YYC, contra las dimensiones de la matriz. Estableciendo, que tanto YYC como su adaptación son los más aptos ante matrices significativamente grandes que pertenezcan a esta familia de problemas.

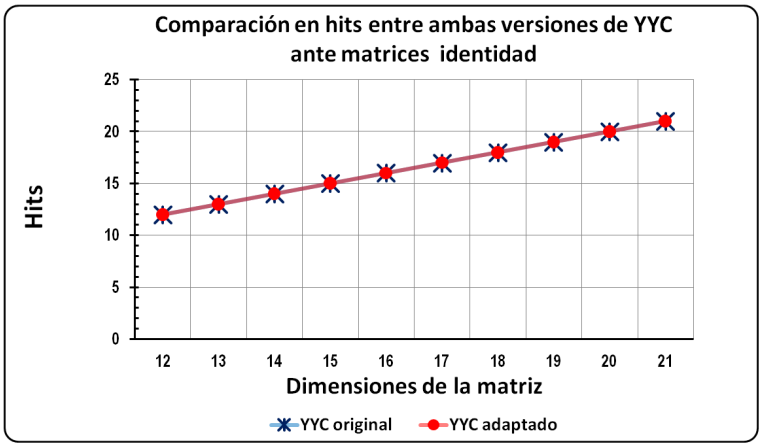


Figura 5.21: Gráfica comparativa de *hits* entre ambas versiones de YYC ante matrices identidad

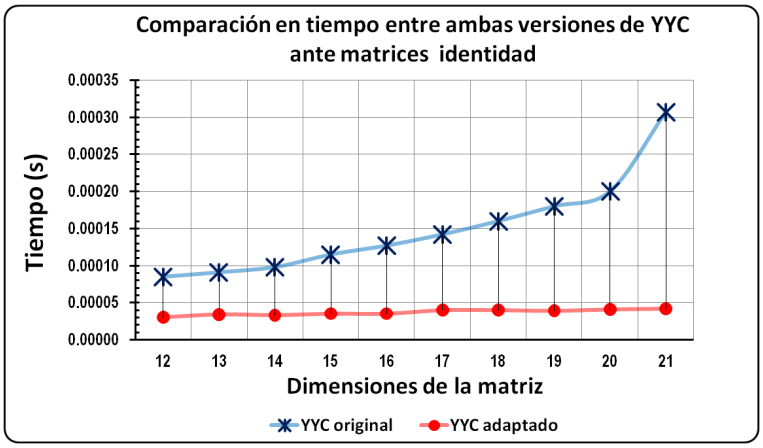


Figura 5.22: Gráfica comparativa del tiempo de ejecución entre ambas versiones de YYC ante matrices identidad

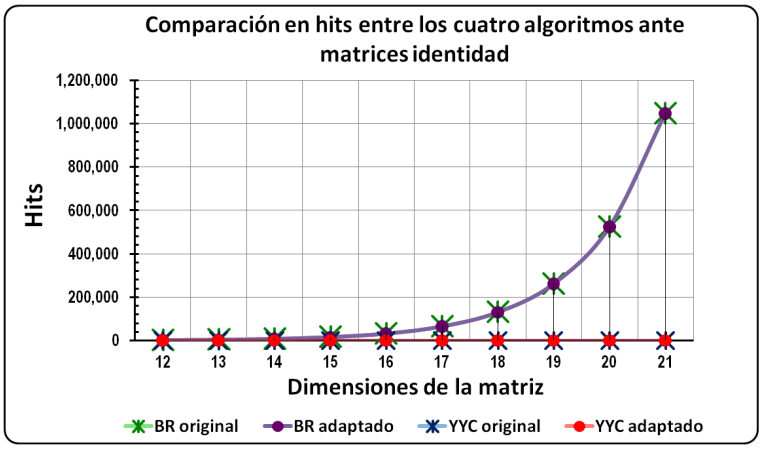


Figura 5.23: Gráfica comparativa de *hits* entre las cuatro versiones de los algoritmos ante matrices identidad

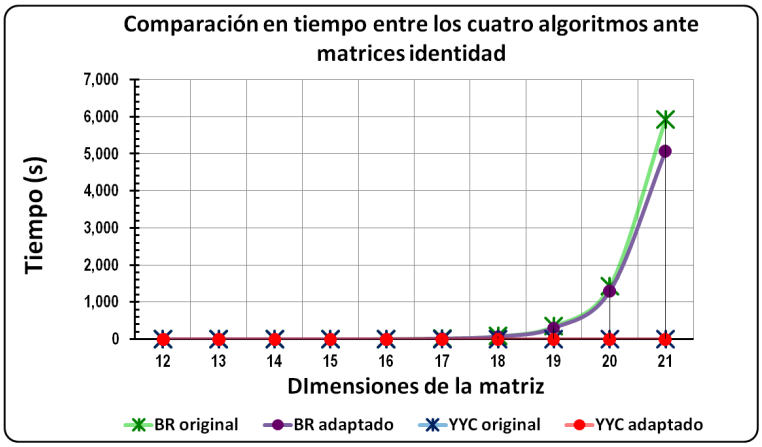


Figura 5.24: Gráfica comparativa del tiempo de ejecución entre las cuatro versiones de los algoritmos ante matrices identidad

5.5. Resumen general de resultados

En esta sección se recopila el resumen de los resultados obtenidos en cada uno de los escenarios de experimentación. De igual manera, se presentan algunas conclusiones acerca de lo observado en los experimentos.

La Tabla 5.9 muestra el promedio de las mejoras que se obtuvieron por las versiones adaptadas de los algoritmos con respecto a sus versiones originales. Las columnas en la sección denominada **Mejoras promedio** de la Tabla 5.9 contienen el promedio de las diferencias tanto en *hits* como tiempo de las dos versiones de los algoritmos en cada escenario, es decir, contienen el promedio de las columnas **Hits** y **Tiempo (s)** de la sección **Mejoras** de cada una de las tablas de resultados de cada escenario.

Analizando en primer lugar la comparación entre las dos versiones de BR, podemos observar que en todos los escenarios la versión adaptada evita revisar, en promedio, una gran cantidad de subconjuntos del espacio de búsqueda. Asimismo, la versión adaptada de BR resulta tener un mejor desempeño general en tiempo de ejecución, creando una brecha significativa en este rubro entre las dos versiones de BR.

Al analizar la comparación entre las dos versiones de YYC, podemos notar que en todos los escenarios la versión adaptada evita revisar, en promedio, cierta cantidad de subconjuntos del espacio de búsqueda; aunque particularmente en los últimos dos escenarios la versión adaptada de YYC se comporta exactamente de la misma manera que su versión original, lo cual nos dice que la estrategia al menos en este rubro no aporta ningún beneficio al algoritmo. Por otro lado, en tiempo de ejecución podemos notar que solo en tres de los escenarios existe una mejora promedio poco significativa y, particularmente en el primer escenario la versión adaptada de YYC tiene un déficit promedio del tiempo de ejecución muy alto, concluyendo así que bajo esa familia de problemas, la estrategia de aprendizaje no es una opción viable.

Además, considerando de forma general todos los datos que se presentan en la Tabla 5.9 resulta evidente notar, que los algoritmos externos como BR obtienen mayores beneficios al incorporar la estrategia de aprendizaje.

Tabla 5.9: Resumen general de resultados

Comparación	**Tipo de matriz**	**Mejoras promedio**	
		Hits	**Tiempo (s)**
BR adaptado vs BR original	Matrices con crecimiento lineal en columnas	609,580	242.2280
	Matrices con crecimiento exponencial en renglones	304,762	1,148.4683
	Matrices con crecimiento proporcional	40,308	402.5391
	Matrices identidad	609,580	242.2280
YYC adaptado vs YYC original	Matrices con crecimiento lineal en columnas	404,618	-462.2371
	Matrices con crecimiento exponencial en renglones	2,551	0.1023
	Matrices con crecimiento proporcional	0	0.0057
	Matrices identidad	0	0.0001

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones generales sobre el desarrollo de este trabajo y sobre los resultados obtenidos. De igual manera, se plantean futuras líneas de investigación que complementan este trabajo.

6.1. Conclusiones generales

Como se planteó en un inicio, este trabajo propone una estrategia de aprendizaje general que mejora el desempeño de los algoritmos de búsqueda de testores típicos, tanto internos como externos. Además, como resultado de la estrategia de aprendizaje, también se propuso un nuevo esquema de caracterización para el espacio asociado al problema de búsqueda de testores típicos. Ambas propuestas representan nuevas aportaciones a la teoría de testores y a su vez permitieron hacer una publicación en el Congreso Iberoamericano de Reconocimiento de Patrones (CIARP 2015) [22].

A pesar de que la estrategia de aprendizaje mejora el desempeño de los ABTT, se observó que con YYC ante la familia de problemas del primer escenario la estrategia no aporta ningún beneficio. Esto sucede principalmente porque los ABTT están diseñados para hacer frente al problema bajo dos diferentes formas de recorrer el espacio de búsqueda, es por ello que los ABTT se dividen en dos clases: algoritmos internos y algoritmos externos. De acuerdo a la clase a la que pertenezcan, habrá instancias del problema que no podrán solucionar de manera eficiente. Por ejemplo, si consideramos la clase de los algoritmos externos, particularmente el algoritmo BR, nos daremos cuenta que las instancias del problema que más afectan su desempeño son aquellas donde la matriz de entrada tiene un gran número de columnas, ya que este algoritmo trabaja con el conjunto potencia columnas de la matriz, lo cual es demasiado costoso computacionalmente. Por otro lado, si consideramos la clase de los algoritmos internos, particularmente el algoritmo YYC, nos daremos cuenta que las instancias que más afectan su desempeño son aquellas donde la matriz de entrada tiene un gran número de renglones, ya que este algoritmo analiza la matriz renglón por renglón y a medida que ésta tiene más renglones, más costoso analizarla. Sin embargo, algo que pudo observarse en los experimentos fue que la estrategia de aprendizaje beneficia en mayor medida a los algoritmos ante las instancias del problema que originalmente afectaban su desempeño.

Una cuestión importante que se observó durante la fase de experimentación, fue la necesidad de evaluar a partir de qué dimensiones de la matriz resulta prudente incorporar la estrategia de aprendizaje a un ABTT, ya que, si se hace para instancias pequeñas, el desempeño del algoritmo puede verse afectado.

Adicionalmente, otra consideración que debe tomarse en cuenta para incorporar la estrategia de aprendizaje en un ABTT, es el tipo de algoritmo al que se le incorporará (interno o externo), ya que el proceso de revisión del espacio de búsqueda en cada tipo de algoritmo es muy distinto. Los algoritmos externos adoptan de mejor manera la estrategia, debido a que en su proceso de revisión del espacio hay una mayor diversidad de subconjuntos, lo cual le permite utilizar todas las clases de la caracterización del espacio, así como también la información de las dominaciones. En contraste, los algoritmos internos utilizan menor cantidad de información de aprendizaje ya que únicamente pueden utilizar las dominaciones.

La consecuencia directa de incorporar aprendizaje a los ABTT es la posibilidad de hacer frente a matrices de entrada cada vez mayores, lo cual se garantiza bajo las condiciones adecuadas. Sin embargo, la estrategia de aprendizaje depende totalmente del algoritmo sobre el que ésta trabaje y por lo tanto no garantiza la generalización de un ABTT ante cualquier tipo de problema.

Evidentemente, la búsqueda de testores típicos sigue sin poder resolverse en tiempo polinomial. Sin embargo, la estrategia de aprendizaje propuesta en este trabajo aporta una nuevo enfoque de solución y nuevos elementos a la teoría de testores que pueden ser utilizados o extendidos en trabajos posteriores.

6.2. Trabajo futuro

Aunque la estrategia de aprendizaje permitió hacer frente a problemas más grandes, ésta dependía totalmente del algoritmo en el que fuera incorporada. Sin embargo, la paralelización del proceso de búsqueda de testores típicos es un enfoque diferente para atacar problemas con conjuntos de datos muy grandes, lo cual se logra mediante la división de la matriz de entrada en diferentes submatrices, encontrando los testores típicos en cada una de esas submatrices y posteriormente reintegrar los resultados. No obstante, aún no conoce un método universal para dividir la matriz de entrada y, por tanto tampoco la forma integrar los resultados parciales obtenidos.

Otro trabajo complementario a éste podría ser el diseño de una taxonomía de los ABTT existentes en teoría de testores, mediante la cual podría establecerse un marco de referencia que ayude decidir sobre que ABTT es más conveniente incorporar la estrategia de aprendizaje. Adicionalmente, se puede generar una comparación entre los diferentes ABTT para saber cual de ellos obtiene el mayor beneficio al incorporar la estrategia.

Dada la relación interdisciplinaria que genera MONET debido a sus diferentes equivalencias; resulta interesante estudiar de qué manera otras disciplinas han hecho frente a otras instancias equivalentes al problema MONET. Esto nos permite extraer ideas de estas disciplinas a fin de ampliar el conocimiento que se tiene en teoría de testores.

Referencias

[1] Afshar, Shahriar S., et al. Paradox in wave-particle duality. Foundations of Physics 37.2: 295-305 (2007).

[2] Matthias Hagen, Algorithmic and Computational Complexity Issues of MONET, Cuvillier Verlag, (2008).

[3] Lias-Rodríguez, A. Pons-Porrata, BR: A new method for computing all typical testors. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 433–440. Springer, Heidelberg (2009).

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

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

[6] Alba-Cabrera, E., Ibarra-Fiallo, J., Godoy-Calderon, S.: A theoretical and practical framework for assessing the computational behavior of typical testor-finding algorithms. In: Ruiz-Shulcloper, J., Sanniti di Baja, G. (eds.) CIARP 2013, Part I. LNCS, vol. 8258, pp. 351–358. Springer, Heidelberg (2013)

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

[8] Ortiz-Posadas, M., Martinez-Trinidad, F., Ruiz-Shulcloper, J.: A new approach to differential diagnosis of diseases. International Journal of Biomedical Computing 40(3), 179–185 (2001).

[9] Pons-Porrata, A., Gil-García, R.J., Berlanga-Llavori, R.: Using typical testors for feature selection in text categorization. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 643–652. Springer, Heidelberg (2007).

[10] Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R.: A Method for the Automatic Summarization of Topic-Based Clusters of Documents. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 596–603. Springer, Heidelberg (2003).

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

[12] Diaz-Sanchez, G., Piza-Davila, I., Sanchez-Diaz, G., Mora-Gonzalez, M., Reyes- Cardenas, O., Cardenas-Tristan, A., Aguirre-Salado, C.: Typical Testors Generation Based on an Evolutionary Algorithm. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 58–65. Springer, Heidelberg (2011).

[13] Sánchez Díaz, G., Lazo Cortés, M.: CT-EXT: An Algorithm for Computing Typical Testor set. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 506–514.Springer, Heidelberg (2007).

[14] Guillermo Sanchez-Diaz, Ivan Piza-Davila, Manuel Lazo-Cortes, Miguel Mora-Gonzalez, Javier Salinas-Luna: A Fast Implementation of the CT-EXT Algorithm for the Testor Property Identification. In: Advances in Soft Computing Volume 6438 of the series Lecture Notes in Computer Science pp 92-103 (2010).

[15] Guillermo Sanchez-Diaz, German Diaz-Sanchez, Miguel Mora-Gonzalez, Ivan Piza-Davila, Carlos A. Aguirre-Salado, Guillermo Huerta-Cuellar, Oscar Reyes-Cardenas, Abraham Cardenas-Tristan, A.: An evolutionary algorithm with acceleration operator to generate. a subset of typical testors. Pattern Recognition Letters 41 pp 34–42 (2014).

[16] Alba, E., Santana, R., Ochoa, A., Lazo, M.: Finding Typical Testors By Using an Evolutionary Strategy. In: Proc. of V Iberoamerican Workshop on Pattern Recognition, Lisbon, Portugal, pp. 267–278 (2000)

[17] Cheguis, I. A. and Yablonskii, S. V. About testors for electrical outlines. Uspieji Matematicheskij Nauk 4, (66) pp. 182-184 Moscow (In Russian), (1955).

[18] Dmitriev, A. N.; Zhuravlev, Yu. I. and Krendeleiev, F. P. On the mathematical principles of patterns and phenomena classification. Diskretnyi Analiz 7. pp. 3-15. Novosibirsk, Russia (In Russian), (1966).

[19] Alba-Cabrera E., Ibarra-Fiallo J., Godoy-Calderon S., YYC: A Fast Performance Incremental Algorithm for Finding Typical Testors, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Springer International Publishing, pp. 416-423, (2014).

[20] Mitchell, T. Machine Learning. New York: McGraw-Hill, (1997).

[21] Briscoe, G. and Caelli, T. Symbolic machine learning. Norwood, New Jersey: Ablex Publishing, (1996).

[22] González-Guevara, Víctor Iván, et al. .ª Mixed Learning Strategy for Finding Typical Testors in Large Datasets.Ïberoamerican Congress on Pattern Recognition. Springer International Publishing, (2015).

Apéndices

Apéndice A

Operadores para la generación de matrices sintéticas de prueba

A continuación se presenta breve explicación de los operadores utilizados para la generación de las matrices de prueba en este trabajo. Estos operadores de describen detalladamente en [6].

A.1. Operador concatenación (φ)

El operador φ genera matrices con un mayor número de columnas mediante la concatenación de dos matrices A y B. Sin embargo, para aplicar este operador es necesario que tanto A como B tengan exactamente el mismo número de renglones sin importar el número de columnas. Además, este operador cumple la propiedad asociativa; en consecuencia, es posible escribir $\varphi^N(A)$ para representar la matriz resultante después de haber aplicado N veces el operador φ sobre la matriz A. A continuación se muestra como se construyen las matrices mediante el uso de este operador.

$$\varphi(A,B) = \begin{bmatrix} a_{11}...a_{1n} & & b_{11}...b_{1n'} \\ & \dots & \\ a_{m1}...a_{mn} & & b_{m1}...b_{mn'} \end{bmatrix}$$

La fórmula A.1 se utiliza para calcular el número de testores típicos $\left|\Psi^*(\varphi^N(A))\right|$ contenidos en la matriz resultante después de aplicar el operador φ.

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

A.2. Operador combinación (θ)

El operador θ genera matrices con un mayor número de renglones, aunque con un ligero incremento en columnas, mediante la combinación de dos matrices A y B. Para aplicar este operador no es imperativo que las matrices tengan el mismo número de renglones o el mismo número de columnas. Al igual que φ, este operador cumple la propiedad asociativa, por lo que es posible escribir $\theta^N(A)$ para

representar la matriz resultante después de haber aplicado N veces el operador θ sobre la matriz A. A continuación se muestra como se construyen las matrices mediante el uso de este operador.

$$\theta(A,B) = \begin{bmatrix} a_{11}...a_{1n} & b_{11}...b_{1n'} \\ \dots & \dots \\ a_{11}...a_{1n} & b_{m'1}...b_{m'n'} \\ \dots & \dots \\ a_{m1}...a_{mn} & b_{11}...b_{1n'} \\ \dots & \dots \\ a_{m1}...a_{mn} & b_{m'1}...b_{m'n'} \end{bmatrix}$$

La fórmula A.1 se utiliza para calcular el número de testores típicos $\left|\Psi^*(\theta^N(A,B))\right|$ contenidos en la matriz resultante después de aplicar el operador θ.

$$\left|\Psi^*(\theta^N(A,B))\right| = |\Psi^*(A)| + |\Psi^*(B)| \tag{A.1}$$

A.3. Operador desplazamiento (γ)

El operador γ genera matrices con un crecimiento tanto en renglones como en columnas, posicionando la matriz A en la esquina superior izquierda, seguida por ceros en todas las demás columnas y posicionando la matriz B en la esquina inferior derecha precedida por ceros en todas las columnas. Este operador recibe su nombre porque pareciera que la matriz generada por este operador fuese el desplazamiento de dos matrices concatenadas. Para aplicar este operador no es imperativo que las matrices tengan el mismo número de renglones o el mismo número de columnas. Al igual que φ y θ, este operador cumple la propiedad asociativa, por lo que es posible escribir $\gamma^N(A)$ para representar la matriz resultante después de haber aplicado N veces el operador γ sobre la matriz A. A continuación se muestra como se construyen las matrices mediante el uso de este operador.

$$\theta(A,B) = \begin{bmatrix} a_{11}...a_{1n} & 0 \\ \dots & \dots \\ a_{m1}...a_{mn} & 0 \\ \dots & \dots \\ 0 & b_{11}...b_{1n'} \\ \dots & \dots \\ 0 & b_{m'1}...b_{m'n'} \end{bmatrix}$$

La fórmula A.2 se utiliza para calcular el número de testores típicos $\left|\Psi^*(\gamma^N(A,B))\right|$ contenidos en la matriz resultante después de aplicar el operador γ.

$$\left|\Psi^*(\gamma^N(A,B))\right| = |\Psi^*(A)|\,|\Psi^*(B)| \tag{A.2}$$

Para construir las matrices de prueba con las que se llevaron a cabo los experimentos de este trabajo, se utilizaron las matrices A.3 y A.4, sobre las que se aplicaron todos los operadores aquí descritos.

$$MS = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{A.3}$$

$$MM = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{A.4}$$

La aplicación de los operadores para la construcción de las matrices de prueba se hizo de la siguiente manera:

- Escenario 1: Se aplicó sucesivamente el operador φ sobre la matriz MS partiendo desde $\varphi^{16}(MS)$ hasta $\varphi^{25}(MS)$.
- Escenario 2: Se aplicó sucesivamente el operador *theta* sobre la matriz MM partiendo desde $\theta^{2}(MM)$ hasta $\theta^{10}(MM)$, incluyendo a la matriz MM como parte de las matrices de prueba.
- Escenario 3: Se aplicó sucesivamente el operador *gamma* sobre la matriz MM partiendo desde $\gamma^{2}(MM)$ hasta $\gamma^{10}(MM)$, incluyendo a la matriz MM como parte de las matrices de prueba.