



UNIVERSIDAD SIMÓN BOLÍVAR

Ingeniería de la Computación

Búsqueda Heurística con Bases de Datos de Patrones

Por

Diego Guerrero y Pedro Piñango

Proyecto de Grado

Presentado ante la Ilustre Universidad Simón Bolívar

como Requerimiento Parcial para Optar el Título de

Ingeniero en Computación

Sartenejas, Octubre de 2007

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL DEL PROYECTO DE GRADO

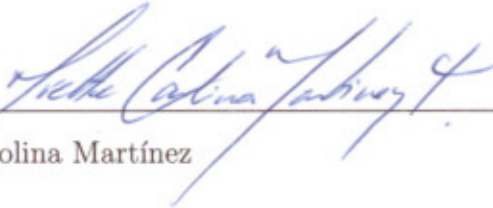
BÚSQUEDA HEURÍSTICA CON BASES DE DATOS DE PATRONES

Presentado Por:
DIEGO GUERRERO Y PEDRO PIÑANGO

Este proyecto de Grado ha sido aprobado por el siguiente jurado examinador:



Prof. Hilmar Castro



Prof. Ivette Carolina Martínez



Prof. Blai Bonet (Tutor Académico)

SARTENEJAS, 24 de octubre de 2007

Búsqueda Heurística con Bases de Datos de Patrones

Por

Diego Guerrero y Pedro Piñango

RESUMEN

Una base de datos de patrones (*PDB*) es una función heurística conformada por una tabla que almacena los costos estimados de las soluciones óptimas para instancias de un subproblema [Felner y Adler, 2005]. La búsqueda heurística con bases de datos patrones es la alternativa que ha brindado los mejores resultados para la obtención de soluciones óptimas al problema del *N-Puzzle* [Felner *et al.*, 2004a]. Por esta razón, el objetivo principal de esta investigación consistió en construir bases de datos de patrones para resolver instancias del *15-Puzzle* y *24-Puzzle* utilizando el algoritmo *IDA**. De igual manera, se realizó una comparación entre los distintos algoritmos de indexación de *PDB*, con la finalidad de determinar que alternativa ofrece el mejor desempeño.

Adicionalmente, en este trabajo se desarrolló un nuevo procedimiento para la indexación de las bases de datos de patrones. Este método está basado en la propiedad de que para la mayoría de la configuraciones del tablero, si se considera la ubicación de los componentes del patrón y las fichas indistinguibles, varias posiciones de la casilla vacía pudiesen agruparse y almacenarse como un solo elemento en la *PDB*. Esto es posible ya que los movimientos entre el blanco y las fichas indistinguibles no son tomados en cuenta en el costo final asociado a un patrón. A este procedimiento se le denominó indexación por regiones de blancos, debido a que las regiones que forman las fichas indistinguibles adyacentes se consideran como una única posición válida para el blanco. Este procedimiento no estaba contemplado en los objetivos del trabajo y representa el principal aporte de esta investigación.

Por medio del análisis de los resultados, se determinó que gracias al manejo más efectivo de la memoria, el algoritmo de indexación lexicográfica resulta en los mejores tiempos de búsqueda en comparación con las otras alternativas. Asimismo, el método propuesto para indexar las *PDB* por regiones de blancos resultó en los mejores valores heurísticos y por ende la menor cantidad de nodos generados, pero debido al costo agregado de mantener y manejar una mayor cantidad de información en memoria (bases de datos más grandes y arreglos precalculados), la tasa de generación de nodos no resultó ser lo suficientemente elevada para superar el desempeño del algoritmo lexicográfico.

Agradecimientos

En primer lugar, gracias a Dios.

Asimismo, a nuestros padres María Eugenia Rincón, Alberto Guerrero, y en especial a Pedro Piñango y Gisela Lizardo quienes se encargaron de mantenernos vivos y alimentados en los momentos más difíciles.

Agradecemos a nuestro tutor Blai Bonet por ofrecernos la oportunidad de trabajar con él y por su ayuda invaluable para encontrar las repuestas necesarias para alcanzar los objetivos planteados.

Igualmente, damos las gracias a todos los integrantes del Grupo de Inteligencia Artificial, en especial a Fabiola Di Bartolo, Francelice Sardá y Celso Gorrín quienes nos acompañaron en nuestra última lambada.

Finalmente, agradecemos a cada una de las chicas del GIA: Adriana, Shakira, Fergie, Brooke, Elisha, Paris y a nuestra importada PuiPui, quienes trabajaron día y noche incondicionalmente por el éxito de esta investigación.

Índice general

Agredecimientos	III
Índice general	IV
Índice de Cuadros	VI
Índice de Figuras	VII
Capítulo 1. Introducción	1
Capítulo 2. Marco Teórico	5
2.1. Búsqueda No Informada	5
2.1.1. <i>Breadth-First Search (BFS)</i>	9
2.1.2. <i>Depth-First Iterative Deepening Search</i>	10
2.2. Búsqueda Heurística	11
2.2.1. A^*	14
2.2.2. <i>Iterative Deepening A^*</i>	16
2.3. <i>N-Puzzle</i>	18
2.4. Construcción de Heurísticas Admisibles	19
2.4.1. Distancia Manhattan	20
2.5. Bases de Datos de Patrones	21
2.5.1. Bases de Datos de Patrones Aditivas	24
2.5.2. Compresión de Bases de Datos de Patrones	26
2.5.3. Indexación de Bases de Datos de Patrones	27
2.5.4. Propiedad simétrica del <i>N-Puzzle</i>	31
Capítulo 3. Implementación	35
3.1. Compresión por Regiones de Blancos	35
3.2. Enumeración de Instancias del <i>N-Puzzle</i>	38

3.2.1.	Enumeración No Lexicográfica	38
3.2.2.	Enumeración Lexicográfica	39
3.2.3.	Enumeración de Combinaciones	40
3.2.4.	Enumeración por Regiones de Blancos	41
3.3.	Generación de Bases de Datos de Patrones	43
3.3.1.	<i>Breadth First Search</i> Modificado	44
3.4.	Resolución de Instancias del <i>N-Puzzle</i>	47
3.4.1.	Cálculo de sucesores	47
3.4.2.	Construcción de la heurística	47
Capítulo 4. Experimentos y Resultados		50
4.1.	Generación de Bases de Datos de Patrones	52
4.2.	Comparación de Bases de Datos de Patrones	53
4.3.	Comparación de Algoritmos de Enumeración	55
4.4.	Resolución de Instancias del <i>N-Puzzle</i>	55
Capítulo 5. Conclusiones y Recomendaciones		60
Bibliografía		62
Apéndice A. Distribución de Patrones		63
Apéndice B. Casos de Prueba		67
Apéndice C. Resultados Individuales		70

Índice de cuadros

1.	Desempeño del algoritmo BFS	10
2.	Desempeño del algoritmo DFID	11
3.	Correspondencia entre permutaciones y valores numéricos únicos	29
4.	Distribución de regiones de blancos para el <i>15-Puzzle</i> y un patrón de 8 elementos	37
5.	Número de unos en la representación binaria del índice	40
6.	Configuración de computadoras utilizadas en los experimentos	51
7.	Abreviaciones utilizadas en los resultados	51
8.	Generación de las <i>PDB</i> para el <i>15-Puzzle</i>	52
9.	Generación de las <i>PDB</i> para el <i>24-Puzzle</i>	52
10.	Regiones de blancos vs. sólo patrón	53
11.	Diferencias de valores heurísticos entre RB y SP	54
12.	Porcentaje de patrones según el número de regiones de blancos	54
13.	Tiempo de enumeración de 10 millones de instancias del <i>24-Puzzle</i>	55
14.	Desempeño de <i>Iterative Deepening A*</i> según funciones agregadas	56
15.	Resultados promedio para las 100 instancias del <i>15-Puzzle</i>	57
16.	Resultados promedio para las 20 instancias del <i>24-Puzzle</i>	57
17.	Número de patrones según su valor heurístico para el <i>15-Puzzle</i>	63
18.	Número de patrones según su valor heurístico para el <i>24-Puzzle</i>	65
19.	100 instancias del <i>15-Puzzle</i>	69
20.	20 instancias del <i>24-Puzzle</i>	69
21.	Resultados individuales de las 100 instancias del <i>15-Puzzle</i>	72
22.	Resultados individuales de las 20 instancias del <i>24-Puzzle</i>	72

Índice de figuras

1.	Estrategia de expansión de BFS	9
2.	Algoritmo <i>Breath-First Search</i>	10
3.	Algoritmo <i>Depth-first Iterative Deepening Search</i>	12
4.	Estrategia de expansión de DFID	13
5.	Algoritmo A^*	15
6.	Algoritmo <i>Iterative Deepening A^*</i>	17
7.	<i>15-Puzzle</i> y <i>24-Puzzle</i> en el estado objetivo	18
8.	Ejemplo de un patron de siete fichas para el <i>15-Puzzle</i>	21
9.	Patrones objetivo en las bases de datos denominadas <i>Corner</i> y <i>Fringe</i>	22
10.	Ejemplo de bases de datos aditivas para el <i>15-Puzzle</i> y el <i>24-Puzzle</i>	24
11.	Diferencia entre no considerar y considerar la posición del blanco	26
12.	Algoritmos de correspondencia no lexicográfica entre permutaciones y números enteros	29
13.	Reflejo de un camino	32
14.	Cálculo del reflejo $p' = D \circ p \circ D$	33
15.	Base de datos de patrones aditivas para el <i>24-Puzzle</i> y su reflejo	34
16.	Valores de la <i>PDB</i> con el método de compresión por regiones de blancos	36
17.	Diferencia de valores en la <i>PDB</i> : método sin blanco h_1 vs. regiones de blancos h_2	37
18.	Enumeración de estados por regiones de blancos	41
19.	<i>BFS</i> modificado para la generación de bases de datos de patrones	45
20.	Instancia del <i>24-Puzzle</i> y sus sucesores con los índices lexicográficos especificados	49
21.	Distribución de patrones según su valor heurístico para el <i>15-Puzzle</i>	64
22.	Distribución de patrones según su valor heurístico para el <i>24-Puzzle</i>	66

Capítulo 1

Introducción

El *N-Puzzle* es un juego que consiste de N fichas distintas y una casilla vacía o blanco contenidas en un marco cuadrado, donde cualquier ficha adyacente vertical u horizontalmente al blanco puede ser deslizada a esa posición[Korf y Felner, 2002]. El objetivo del juego consiste en reorganizar las fichas partiendo desde la configuración inicial hasta obtener una configuración objetivo particular. El *15-Puzzle* y *24-Puzzle* son versiones de este juego que cuentan con aproximadamente 10^{13} y 10^{25} posibles configuraciones cada uno respectivamente.

Encontrar una secuencia cualquiera de acciones que consiga el objetivo del juego resulta bastante sencillo, pero hallar el número mínimo de acciones necesarias es un problema NP-Completo[Russell y Norvig, 2003], es decir, se requiere de un tiempo polinomial no determinístico de ejecución para lograrlo. Por esta razón, es necesario utilizar algoritmos de búsqueda heurística como A^* o *Iterative Deepening A^** (IDA^*), para encontrar las soluciones asociadas a instancias del *N-Puzzle*, debido a que sería prácticamente imposible atacar este problema con estrategias de búsqueda tradicional.

Estos algoritmos emplean una función, denominada heurística, para estimar el costo del camino más corto hasta el objetivo, y si esta función nunca sobreestima este valor, entonces el algoritmo garantiza hallar la mínima solución, si ésta existe.

La distancia *manhattan*[Russell y Norvig, 2003] es una de las funciones heurísticas más utilizadas para resolver instancias del *N-Puzzle*, pero ya que solamente toma en cuenta el costo individual de cada ficha, para problemas de gran envergadura como el *24-Puzzle*, el algoritmo IDA^* tardaría demasiado tiempo en encontrar soluciones óptimas[Korf y Felner, 2002]. Como alternativa a esta heurística surgen las bases de datos de patrones, que consisten en calcular el costo de resolver un subconjunto de las fichas originales, suponiendo que el resto de ellas son indistinguibles, para todas las posibles permutaciones de las fichas seleccionadas, es decir, para todos los patrones existentes[Culberson y Schaeffer, 1996]. Estos valores

se precalculan, se almacenan en memoria y se consultan durante el proceso de búsqueda para cada caso en particular.

El objetivo principal de este trabajo es construir bases de datos de patrones para resolver instancias del *15-Puzzle* y *24-Puzzle* utilizando el algoritmo *IDA**. Particularmente, el objetivo específico consiste en evaluar y comparar los distintos algoritmos de indexación de bases de datos de patrones para determinar si existe diferencia en el desempeño de *IDA** debido a la distribución lexicográfica y no lexicográfica de la información almacenada en ellas.

El método de almacenamiento para las bases de datos de patrones consiste en un arreglo unidimensional de tamaño igual al número de patrones existentes. Por esta razón, es indispensable el uso de funciones de enumeración de patrones para generar los índices necesarios para acceder dicho arreglo. Este proceso de corresponder patrones a valores numéricos únicos es lo que se denomina indexación. Debido al reciente descubrimiento de un algoritmo de enumeración lexicográfica en orden lineal[Korf y Schultze, 2005], se llevó a cabo un estudio acerca de la influencia del ordenamiento de las bases de datos de patrones sobre el desempeño de *IDA**. Este análisis representa un aporte de esta investigación.

Asimismo, para dar cumplimiento al objetivo general planteado, en primer lugar se procedió a reproducir el trabajo realizado en [Korf y Felner, 2002], dado que representa la mejor opción conocida para la búsqueda de soluciones óptimas al *N-Puzzle*[Felner *et al.*, 2004a]. En dicho trabajo, al momento de la generación de las bases de datos de patrones, la posición de la casilla vacía solamente se toma en cuenta para el cálculo del costo asociado a una configuración particular de las fichas en el tablero, más no se considera para realizar la indexación en la base de datos. Luego, como mencionan los autores, para poder garantizar que la función heurística no sobrestime el costo asociado a una configuración cualquiera de las fichas, se almacena en la base de datos de patrones el mínimo valor para todas las posibles posiciones de la casilla vacía.

La principal limitante por la que no se toma en cuenta la casilla vacía para la indexación de las bases de datos de patrones es que por cada aumento en el número de fichas seleccionadas, el incremento del tamaño de la base de datos es aún mayor.

Debido a que la técnica de [Korf y Felner, 2002] sacrifica información al tomar el mínimo de los valores para todas las posibles posiciones del blanco, en este trabajo fue propuesto y desarrollado un nuevo método denominado regiones de blancos. Este método permite preservar la información correspondiente a la casilla vacía, sin incurrir en el mismo incremento de tamaño experimentado por las otras bases de datos de patrones.

Esta nueva técnica utiliza una propiedad del blanco que proviene del hecho de que para la generación de las bases de datos de patrones, los movimientos entre la casilla vacía y las fichas que no pertenecen al patrón no tienen costo, por lo tanto, el blanco puede moverse libremente por grupos de fichas indistinguibles adyacentes sin generar costo adicional. Entonces, si se agrupan estas casillas adyacentes y se almacenan como un sólo elemento en la base de datos de patrones se habrá logrado reducir el espacio desperdiciado por elementos iguales. Esta técnica representa el principal aporte de este trabajo.

Para estudiar el desempeño de las distintas bases de datos de patrones utilizadas durante la búsqueda de soluciones óptimas para el *15-Puzzle* y *24-Puzzle*, se evaluaron los siguientes criterios: total de nodos generados y tiempo total de ejecución necesario para conseguir la solución. Ambas medidas son importantes ya que representan respectivamente el tamaño del árbol inspeccionado durante la búsqueda, y la eficiencia con la se realiza la misma. Como se mencionó previamente, el proceso de búsqueda se llevó a cabo por medio del algoritmo *Iterative Deepening A** utilizando las bases de datos de patrones como función heurística.

Para el estudio comparativo de los algoritmos de indexación se consideraron tres opciones diferentes: en orden no lexicográfica, en orden lexicográfica y por regiones de blancos.

Luego del análisis de los resultados, se determinó que gracias al manejo más efectivo de la memoria, la indexación lexicográfica resulta en los mejores tiempos de búsqueda en comparación con la indexación no lexicográfica y por regiones de blanco. Asimismo, el método propuesto para indexar las *PDB* por regiones de blancos resultó en los mejores valores heurísticos y por ende la menor cantidad de nodos generados, pero debido al costo agregado de mantener y manejar una mayor cantidad de información en memoria (bases de datos más grandes y arreglos precalculados), la tasa de generación de nodos no resultó ser lo

suficientemente elevada para superar el desempeño del algoritmo lexicográfico.

Finalmente, la estructura de este trabajo consta de 5 capítulos. El Capítulo 2 describe en detalle la búsqueda heurística con bases de datos de patrones, incluyendo los algoritmos necesarios para la indexación de las mismas. El Capítulo 3 presenta la implementación de los distintos algoritmos de indexación y la nueva estrategia de compresión por regiones de blancos. El Capítulo 4 detalla los experimentos realizados, los resultados obtenidos y el análisis de los mismos. Finalmente, el Capítulo 5 presenta las conclusiones y recomendaciones para futuros trabajos.

Capítulo 2

Marco Teórico

Este capítulo describe la base teórica sobre la cual se sustenta la rama de la **inteligencia artificial** (IA) denominada **búsqueda heurística**, en particular lo concerniente al tópico de las **bases de datos de patrones** (*PDB* por sus siglas en inglés), siendo el objetivo final de este estudio el uso de esta técnica para determinar soluciones óptimas a un problema clásico de la IA, el *N-Puzzle*, empleando la menor cantidad de tiempo y cómputo posible.

2.1. Búsqueda No Informada

Un tema de relevante importancia dentro del campo de la IA es la **búsqueda sobre grafos**, pero para poder ahondar en este tópico es necesario conocer algunas definiciones.

Definición 1. *Un grafo $G = (V, E)$ es un tipo de dato abstracto que está conformado por un conjunto finito no vacío de nodos V , y por un conjunto finito de lados E donde se establecen las relaciones entre pares de nodos [Meza y Ortega, 1993].*

Debido a la amplia teoría desarrollada y previamente probada que existe sobre este tópico, y dado el alto nivel de abstracción con el cual fue concebido, este tipo de datos es frecuentemente utilizado en la computación hoy en día. Estas cualidades convierten a los grafos en herramientas muy útiles para representar distintos problemas y sus contextos de una manera que facilita la búsqueda de soluciones.

Según [Russell y Norvig, 2003], se puede establecer una correlación entre un problema y un grafo si el primero se desglosa en los siguientes componentes:

- Un **estado inicial** que representa el punto de partida para la búsqueda.
- Un **conjunto de acciones** válidas que permita generar o alcanzar nuevos estados a partir de otro. Una forma común de representar este conjunto es por medio de lo que

se denomina una **función de sucesores**, la cual toma como valores de entrada un estado y una acción válida, y genera como resultado otro estado.

El estado inicial en conjunto con la función de sucesores definen el **espacio de búsqueda** del problema, es decir, el conjunto de estados alcanzables¹ desde el estado inicial. Este espacio de búsqueda se puede representar por medio de un grafo donde a cada estado se asocia un nodo y cada acción representa un lado que conecta dos nodos.

De la misma manera, el estado inicial y la función de sucesores definen el **árbol de búsqueda** del problema, el cual representa todos los posibles caminos existentes desde el estado inicial hasta cada uno de los estados que pertenecen al espacio de búsqueda.

- La **prueba de finalización** que determina si algún estado es equivalente al estado objetivo. Este último no tiene que ser necesariamente un estado o conjunto de estados en particular, también puede estar representado por un grupo de condiciones que se tienen que cumplir para dar por concluida la búsqueda. Tal es el caso del ajedrez, en donde la prueba de finalización debe determinar si el rey está siendo atacado y no puede escapar, a lo que se denomina “jaque mate”.
- Una **función de costos** (denotada $g(n)$) que asigna un valor numérico a cada camino. Este valor o costo viene determinado por el contexto particular de cada problema. Tomando como ejemplo el problema del “agente viajero” [Russell y Norvig, 2003], el costo de un camino es equivalente a la distancia total recorrida desde el nodo inicial hasta el nodo final del mismo, es decir, el costo total es igual a la sumatoria de los costos individuales de cada lado que une a cada par de nodos en la secuencia del camino. Por lo tanto, el costo total para un camino $c = n_0, \dots, n_k$ compuesto de $k + 1$ nodos n_i ($\forall i = 0, \dots, k$) se puede determinar por medio de la Ecuación 1.

¹Se dice que un estado es alcanzable con respecto a otro si existe entre ellos un camino que los comunique. Asimismo, un camino de longitud k se define como una secuencia de k nodos conectados por medio de una secuencia de $k - 1$ lados.

$$g(c) = \sum_{i=1}^k \text{Costo}(n_{i-1}, n_i) \quad (1)$$

donde $\text{Costo}(n_{i-1}, n_i)$ es el costo particular del lado que une a los nodos n_{i-1} y n_i ($\forall i = 0, \dots, k$).

Los caminos que parten desde un estado cualquiera y finalizan en un estado objetivo se denominan **soluciones**. Resulta de particular interés en el área de la búsqueda sobre grafos la determinación de **soluciones óptimas**, es decir, aquellas soluciones que tienen el mínimo costo posible dentro de todas las soluciones factibles. Por lo tanto, se afirma que π es una solución óptima si y sólo si $g(\pi) \leq g(\pi')$ para toda solución π' .

Una vez establecidos todos estos conceptos entonces surge la interrogante acerca de cómo se debe realizar la búsqueda de soluciones óptimas, y es aquí donde toman un papel protagonista los algoritmos de búsqueda. Un ciclo de ejecución tipo de este tipo de algoritmo consiste en evaluar un estado y probar si éste corresponde a un estado objetivo, en caso de resultar positiva la prueba, la búsqueda ha finalizado, en caso contrario, se le debe aplicar la función de sucesores para generar un nuevo conjunto de estados en donde continuar la búsqueda. Este procedimiento se debe repetir hasta encontrar un estado objetivo o hasta que se acaben los estados dentro del espacio de búsqueda, lo que significaría que no existe un camino entre el estado inicial y el estado objetivo. La escogencia de cual nodo se expande depende de la estrategia de la búsqueda, la cual varía entre un algoritmo y otro.

Definición 2. *Un algoritmo de búsqueda consiste en un conjunto de instrucciones bien definidas que tienen como objetivo encontrar una solución a un problema previamente descrito [Russell y Norvig, 2003].*

Es importante resaltar la diferencia que existe entre los términos estado y nodo, dado que ambos tienden a confundirse. Los estados representan diferentes configuraciones del problema y existen en la cantidad que el espacio del problema abarca, mientras que los nodos son estructuras que representan estados del problema y que interrelacionan a los

mismos por medio de los lados y caminos del grafo. Una de las principales diferencias radica, por ejemplo, en que durante la ejecución de una búsqueda puede existir más de un nodo que represente al mismo estado, debido a que éste puede ser alcanzado por medio de distintos caminos. Otra diferencia significativa es que un nodo tiene un costo asociado que depende del camino por el cual fue alcanzado, mientras que un estado no.

Asimismo, es conveniente definir algunos conceptos que ayudarán a caracterizar el desempeño de los algoritmos cuando se describan en detalle.

Definición 3. *La completitud de un algoritmo es la garantía que ofrece éste de encontrar una solución, siempre y cuando el modelo del problema sea soluble [Russell y Norvig, 2003].*

Definición 4. *La optimalidad de un algoritmo es la que garantiza que ofrece éste de encontrar una solución óptima [Russell y Norvig, 2003].*

Definición 5. *La complejidad temporal se refiere a la cantidad de tiempo que consume el algoritmo en hallar una solución [Russell y Norvig, 2003].*

Definición 6. *La complejidad espacial describe la cantidad de memoria consumida por el algoritmo durante su ejecución [Russell y Norvig, 2003].*

Para el estudio de complejidad temporal y espacial se utiliza la notación- O descrita en [Cormen *et al.*, 2001], la cual es utilizada para especificar cotas superiores de funciones.

Adicionalmente, resulta práctico para la realización del estudio de desempeño suponer que el espacio de búsqueda cumple las siguientes características:

- Si existe, la solución más cercana se encuentra a una profundidad d .
- El factor de ramificación es b , es decir, cada nodo tiene b sucesores. Se asume b finito.

Teniendo en cuenta estas aclaratorias y retomando el tema de los algoritmos, una primera aproximación para hallar soluciones óptimas es por medio de lo que se denomina **búsqueda no informada** o ciega, la cual se refiere al proceso de búsqueda tomando en cuenta

solamente la información planteada en la definición del problema, es decir, los posibles estados y acciones. Esto trae como consecuencia que el comportamiento de estos algoritmos esté limitado a generar sucesores y distinguir estados objetivos dentro de los nodos generados para poder encontrar soluciones. A continuación se detallan dos algoritmos que utilizan la metodología antes planteada.

2.1.1. *Breadth-First Search (BFS)*

El algoritmo *BFS* utiliza una estrategia de expansión muy sencilla, ésta consiste en expandir en primer lugar el nodo raíz (o nodo inicial) del grafo, seguidamente se toma cada uno de estos sucesores y se generan sus respectivos hijos; repitiéndose el proceso para cada generación de sucesores. Debido a esta dinámica es que surge el nombre *Breadth-First Search*, o búsqueda preferente por amplitud, dado que por cada nivel de profundidad d en la búsqueda se realiza la evaluación de todos sus nodos y la generación de los sucesores de ellos antes de realizar el mismo proceso sobre los nodos de la siguiente profundidad ($d + 1$). Este proceso se detalla en la Figura 1.

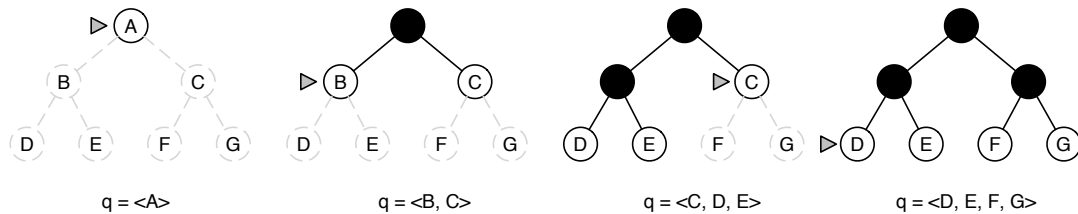


Figura 1: Estrategia de expansión de BFS

Como se observa en la Figura 2, esta estrategia se puede implementar por medio del uso de una estructura de datos de tipo *FIFO* (*first-in-first-out*) como una cola, donde todos los nodos recién generados se colocan al final de ella, y todos aquellos generados en una profundidad menor serán evaluados primero.

En el Cuadro 1 se describe el desempeño de este algoritmo según los criterios establecidos en la Sección 2.1.

Algoritmo 2.1.1: BREADTH-FIRST-SEARCH(*start*)

```

Queue q
ENQUEUE(q, < start, 0 >)
while not EMPTY(q)
  do {
    DEQUEUE(q, < n, c >)
    if GOAL(n)
      then { return (< n, c >)
    else {
      for each s ∈ SUCCESSORS(n)
        do ENQUEUE(q, < s, c + COST(n, s) >)
  return (failure)

```

Figura 2: Algoritmo *Breadth-First Search*

Criterio	Descripción
<i>Completitud</i>	Siempre y cuando el nodo objetivo más cercano se encuentre a una profundidad finita d
<i>Optimalidad</i>	Sólo si los costos de las acciones son iguales
<i>Complejidad temporal</i>	$O(b^{d+1})$
<i>Completitud espacial</i>	$O(b^{d+1})$

Cuadro 1: Desempeño del algoritmo BFS

2.1.2. *Depth-First Iterative Deepening Search*

Este algoritmo tiene como base una estrategia denominada *Depth-First Search (DFS)* [Russell y Norvig, 2003], o búsqueda preferente por profundidad, en donde siempre se expande primero el nodo de mayor profundidad en la frontera del árbol. Esto se lleva a cabo realizando una búsqueda hasta lo más profundo del grafo (nodos que no tienen sucesores), para luego realizar un proceso de regresión hasta encontrar el primer nodo que tenga sucesores sin explorar, y nuevamente se repite el procedimiento.

La implementación de esta estrategia se puede llevar a cabo por medio de una estructura de datos de tipo *LIFO (last-in-first-out)* como una pila, o por medio de una función recursiva que se llame así misma con cada uno de los sucesores del nodo en evaluación.

El problema con esta estrategia de búsqueda es que el algoritmo puede explorar indefinidamente un camino debido a la presencia de ciclos² en el grafo. Para evitar este problema se le establece al algoritmo un límite de profundidad T [Russell y Norvig, 2003], y durante la búsqueda no se realiza la expansión de los nodos encontrados a dicha profundidad. El dilema con esto es cómo saber cual es escoger el valor correcto de T , dado que para $T < d$ no existirá solución, y para $T > d$ las soluciones encontradas podrían ser no óptimas.

Para de solucionar este problema, el algoritmo *Depth-First Iterative Deepening Search*, o búsqueda por profundización iterativa, realiza un incremento unitario iterativo de T partiendo desde cero hasta encontrar la profundidad d del nodo objetivo. En la Figura 3 se puede observar una implementación en pseudocódigo de este algoritmo donde se detalla este procedimiento.

Para observar el comportamiento de la estrategia de expansión de este algoritmo se puede examinar la Figura 4. Finalmente, en el Cuadro 2 se describe el desempeño del algoritmo *DFID*.

Criterio	Descripción
<i>Completitud</i>	Siempre y cuando d y b sean finitos
<i>Optimalidad</i>	Sólo si los costos de las acciones son iguales
<i>Complejidad temporal</i>	$O(b^d)$
<i>Completitud espacial</i>	$O(bd)$

Cuadro 2: Desempeño del algoritmo DFID

2.2. Búsqueda Heurística

El principal problema de los algoritmos de búsqueda no informados es que son ineficientes para problemas con espacios de búsqueda muy grande, como es el caso de este trabajo, y la principal razón por la que esto sucede es porque ellos no toman en cuenta información específica del problema aparte de la definición del mismo, por lo tanto, no distinguen cuáles nodos son “más convenientes” para explorar [Russell y Norvig, 2003].

²Un ciclo es un camino de un grafo que tiene como punto de partida y llegada el mismo nodo [Meza y Ortega, 1993].

Algoritmo 2.1.2: DEPTH-FIRST-ITERATIVE-DEEPENING(*start*)

```

procedure RECURSIVE-DFID(n, T, d)
  if GOAL(n)
    then return (s)
  if (T = d)
    then return (null)
  for each s ∈ SUCCESSORS(n)
    do {
      sol ← RECURSIVE-DFID(s, T, d + 1)
      if not (sol = null)
        then return (sol)
    }
  return (null)

main
  for T ← 0 to ∞
    do {
      sol ← RECURSIVE-DFID(start, T, 0)
      if not (sol = null)
        then return (< sol, T >)
    }

```

Figura 3: Algoritmo *Depth-first Iterative Deepening Search*

Como respuesta a esta problemática surgieron los algoritmos de **búsqueda informada**. Estos algoritmos, en general, emplean una estrategia de expansión denominada *best-first search* [Russell y Norvig, 2003], en donde el nodo a expandir se escoge en base a una función de evaluación $f(n)$. Luego, por medio de una cola de prioridades, siempre se expande primero aquel nodo que posea el mejor valor, según sea el caso.

Aparte de la estrategia de expansión, los algoritmos de búsqueda informada se diferencian según la función de evaluación utilizada. Existe una familia particular de estos algoritmos catalogados como **búsqueda heurística**, que son de gran importancia para este trabajo. Estos algoritmos utilizan una herramienta llamada **función heurística** que les permite manejar conocimiento específico del problema para determinar el mejor nodo a expandir.

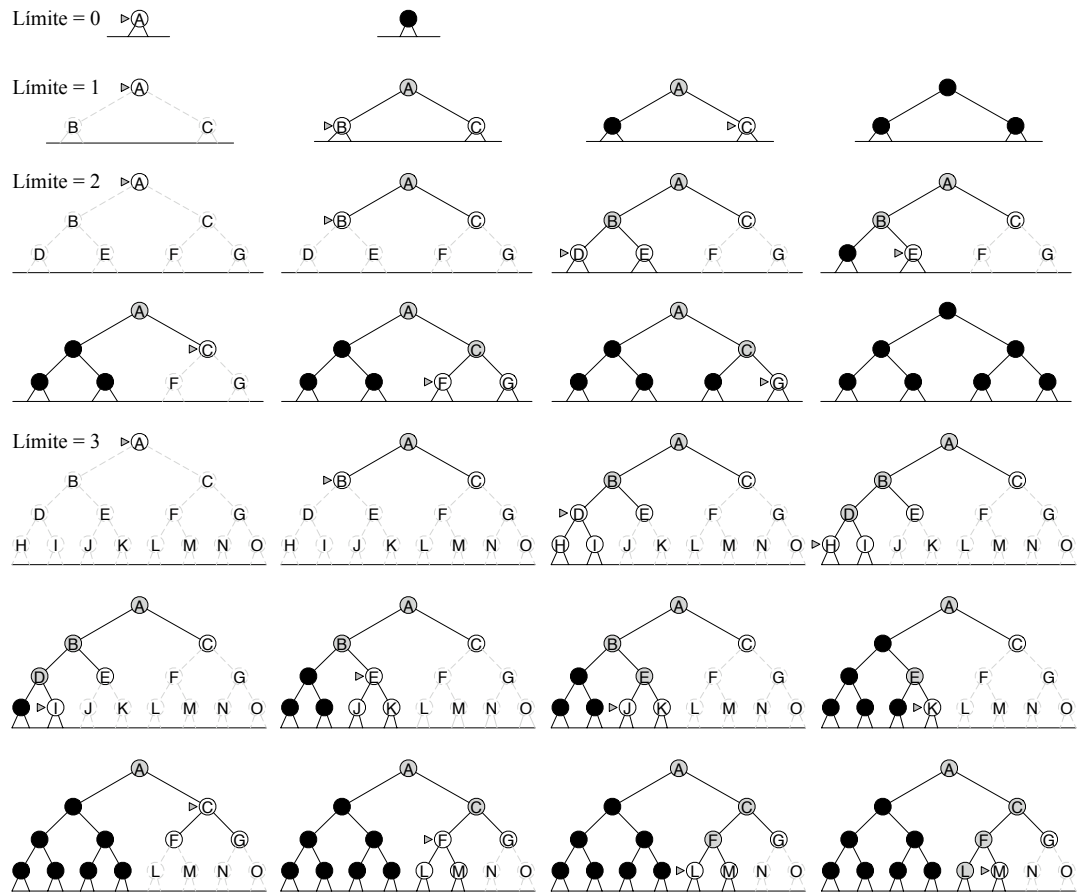


Figura 4: Estrategia de expansión de DFID

Definición 7. Una función heurística $h(n)$ calcula el costo estimado del camino más económico desde el nodo n hasta el nodo objetivo [Russell y Norvig, 2003].

Algunas características importantes de estas funciones son las siguientes:

- Si n representa el nodo objetivo, entonces $h(n) = 0$.
- Aunque $h(n)$ toma como entrada un nodo n , el valor de la función depende únicamente del estado que éste represente.
- Se dice que $h(n)$ es **admisible** si para cualquier nodo n la función nunca sobrestima el costo de alcanzar el estado objetivo. La admisibilidad de la heurística es un elemento fundamental para asegurar la optimalidad de este tipo de algoritmos.
- Se dice que h_1 domina a h_2 si para todo nodo n se cumple $h_1(n) \geq h_2(n)$. Generalmente, esto se traduce en un mejor rendimiento del algoritmo siempre y cuando el costo de calcular h_1 no sea muy elevado comparado con h_2 .

En las secciones presentadas a continuación (2.2.1 y 2.2.2) se detallan dos algoritmos de búsqueda heurística que sirven de base para la implementación realizada en este trabajo. Posteriormente se describe el problema del *N-Puzzle* (Sección 2.3) y algunas funciones heurísticas admisibles para él (Sección 2.4 en adelante).

2.2.1. A^*

A^* es un algoritmo de búsqueda informada que consigue el camino óptimo para ir desde de un nodo inicial hasta el nodo objetivo más cercano gracias a la utilización de una función heurística h , la cual representa una cota inferior de la longitud de este camino. La heurística, en combinación con la función de costos $g(n)$, se utiliza para expandir primero aquellos nodos que parezcan más prometedores. De esta manera, si se define f como:

$$f(n) = g(n) + h(n)$$

Algoritmo 2.2.1: $A^*(start)$

```

Priority_Queue  $q$ 
INSERT( $q, \langle start, h(start) \rangle$ )
while not EMPTY( $q$ )
do {
  POP( $q, \langle n, c \rangle$ )
  if GOAL( $n$ )
  then { return  $\langle n, c - h(n) \rangle$  }
  else {
    for each  $s \in \text{SUCCESSORS}(n)$ 
    do INSERT( $q, \langle s, (c - h(n)) + \text{COST}(n, s) + h(s) \rangle$ )
  }
}
return ( $failure$ )

```

Figura 5: Algoritmo A^*

se puede decir que $f(n)$ es un estimado de la longitud del camino más económico que parte desde n hasta el nodo objetivo. Luego, mientras más bajo sea el valor de esta función, el nodo n tendrá una mayor prioridad de ser expandido, ya que la implementación de este algoritmo utiliza una cola de prioridades ordenada ascendentemente según f . Una versión de A^* se puede observar en la Figura 5.

En este algoritmo, la optimalidad está ligada a la admisibilidad de la función heurística, por lo que se puede afirmar que si h nunca sobrestima el costo de ir del nodo n al nodo objetivo, entonces la solución encontrada por el algoritmo será óptima.

En cuanto a la complejidad temporal del algoritmo, es importante destacar que también se encuentra directamente relacionada con la función h escogida para el problema. En el peor de los casos, la cantidad de nodos generados será exponencial con respecto a la longitud d del camino solución, pero si la condición descrita en la Ecuación 2 se cumple, entonces esta cantidad será polinomial [Russell y Norvig, 2003].

$$|h(n) - h^*(n)| \leq O(\log(h^*(n))) \quad (2)$$

La gran limitante de este algoritmo es que necesita almacenar en memoria simultánea-

mente un número exponencial de nodos, lo que restringe significativamente en tamaño de los espacios de búsqueda en los que se puede aplicar.

2.2.2. *Iterative Deepening A**

El algoritmo IDA^* surge como una alternativa ante la limitación espacial que presenta A^* , particularmente para aquellos problemas en los que el espacio de búsqueda es muy grande, como el *15-Puzzle* y *24-Puzzle*. Para solventar este problema, IDA^* combina búsqueda heurística con la estrategia de expansión utilizada por $DFID$ para recorrer el grafo del problema.

Como se explica en [Korf, 1985], IDA^* trabaja de la siguiente manera: en cada iteración de la cota superior T se realiza una búsqueda en profundidad sobre las diferentes ramas del árbol y se utiliza como condición de parada de la profundización que el costo total estimado del nodo en evaluación ($f(n)$) exceda el límite establecido.

A pesar de las similitudes, este algoritmo tiene un par de optimizaciones con respecto a $DFID$. Primero, T es inicializado con el valor heurístico asociado al estado inicial y segundo, en cada iteración la cota se incrementa al mínimo costo que haya excedido el límite para la iteración anterior. Estas mejoras se pueden observar en la implementación que se muestra en la Figura 6.

Una característica particular de este algoritmo es que por cada incremento de la cota T , el proceso de búsqueda vuelve a recorrer el mismo sector del árbol visitado en la iteración anterior. Por esta razón se dice que IDA^* es un algoritmo sin memoria; éste es el motivo por el que siempre genera una mayor cantidad de nodos en comparación con A^* . A pesar de esto, debido a que IDA^* no incurre en el costo de procesamiento que involucra mantener ordenada una cola de prioridades, este algoritmo logra procesar un mayor número de nodos por segundo en comparación con A^* .

Asimismo, debido a que emplea una estrategia de búsqueda por profundización, IDA^* es un algoritmo con requerimientos mínimos de memoria ($O(bd)$), lo que permite aplicarlo a problemas con espacios de búsqueda de tamaños mayores donde A^* no tiene cabida.

Algoritmo 2.2.2: ITERATIVE-DEEPENING- $A^*(start)$

```

procedure RECURSIVE-IDA*( $n, T, g$ )
  if ( $g + h(n) > T$ )
    then return ( $\langle g + h(n), \text{null} \rangle$ )
  if GOAL( $n$ )
    then return ( $\langle g, s \rangle$ )
   $newT \leftarrow \infty$ 
  for each  $s \in \text{SUCCESSORS}(n)$ 
    do {
       $\langle T', sol \rangle \leftarrow \text{RECURSIVE-IDA}^*(s, T, g + \text{COST}(n, s))$ 
      if not ( $sol = \text{null}$ )
        then return ( $\langle T', sol \rangle$ )
       $newT \leftarrow \text{MIN}(newT, T')$ 
    }
  return ( $\langle newT, \text{null} \rangle$ )

main
   $T \leftarrow h(start)$ 
   $sol \leftarrow \text{null}$ 
  while ( $sol = \text{null}$ ) and ( $T < \infty$ )
    do {  $\langle T, sol \rangle \leftarrow \text{RECURSIVE-IDA}^*(start, T, 0)$ 
  }
  return ( $\langle T, sol \rangle$ )

```

Figura 6: Algoritmo *Iterative Deepening A**

Adicionalmente, toda esta memoria que no es empleada por el algoritmo puede utilizarse para mejorarlo significativamente, ya sea para constuir mejores funciones heurísticas o simplemente para implementar una tabla que impida la transposición³ de nodos.

Por último, *Iterative Deepening A** garantiza la optimalidad de la solución siempre y cuando la función $h(n)$ no sobrestime el costo real de alcanzar el estado objetivo a partir de cualquier nodo n del espacio de búsqueda.

³Una transposición significa alcanzar un mismo estado por medio de diferentes caminos [Russell y Norvig, 2003].

2.3. *N-Puzzle*

El *N-Puzzle* es un rompecabezas de piezas deslizantes que consta de un marco que contiene N fichas diferenciables y una casilla vacía que le da movilidad al resto de las piezas. El propósito del juego es reordenar las fichas moviendo cualquiera que se encuentre horizontalmente o verticalmente adyacente a la casilla vacía, para llegar desde la configuración inicial del tablero hasta el estado objetivo del mismo. En la Figura 7 se muestran los estados objetivos del *15-Puzzle* y *24-Puzzle*.

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Figura 7: *15-Puzzle* y *24-Puzzle* en el estado objetivo

Para los fines del estudio que se realiza en este trabajo, cada estado del juego puede representarse como una secuencia de $N + 1$ números correspondientes al contenido de cada casilla del tablero, o lo que es similar, como una **permutación** de los números enteros que van desde 0 hasta N inclusive. Por ejemplo, para el estado objetivo del *15-Puzzle* la permutación correspondiente es la siguiente:

$$\langle 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \rangle$$

De acuerdo con [Nieto, 2005], este juego fue creado por Sam Loyd en el año 1878, quien ofreció 1000\$ a la persona que resolviese una configuración particular del tablero propuesta por él. Al cabo de un tiempo se demostró que era imposible lograr lo que Loyd quería, debido a que existe una propiedad de este juego que divide sus estados en dos conjuntos disjuntos, o configuraciones del tablero, totalmente disjuntos. Esta propiedad se basa en el siguiente concepto:

Definición 8. Una configuración del tablero es par si y sólo si el número de inversiones presentes en él es par, donde una inversión es un par de fichas (a_i, a_j) tal que $a_i > a_j$ e

$i < j$, siendo i y j las posiciones respectivas de las fichas en la secuencia que representa al estado del N -Puzzle.

Partiendo de esto se afirma que dos estados son alcanzables entre sí siempre y cuando compartan la misma paridad, motivo por el cual se induce la partición de estados antes descrita.

Por lo tanto, a pesar de que existen $16!$ posibles permutaciones de las fichas para el 15 -Puzzle, sólo $\frac{16!}{2}$ configuraciones del tablero son alcanzables desde el estado objetivo por medio de acciones válidas.

2.4. Construcción de Heurísticas Admisibles

En general, las funciones heurísticas admisibles se pueden construir buscando la solución exacta de una versión simplificada o relajada del problema original, es decir, un subproblema en donde existe un menor número de restricciones sobre el conjunto de acciones posibles.

En [Russell y Norvig, 2003] se afirma que una solución óptima del problema original también será solución para una versión simplificada, y el costo asociado a ella será mayor o igual a la solución óptima de la versión relajada, luego el costo de esta última se puede considerar como una heurística admisible del problema original.

Ampliando esta idea, es posible construir n versiones relajadas diferentes de un problema cualquiera, obteniéndose así n heurísticas admisibles distintas $h_i(x)$ (con $i = 1, \dots, n$), y para el caso en que ninguna de ellas domine al resto, se puede definir otra heurística que tome el mejor valor de ellas para cada caso en particular. Según [Russell y Norvig, 2003], esta función se puede definir de la siguiente manera:

$$h(x) = \max\{h_1(x), \dots, h_n(x)\} \quad (3)$$

Debido a que todos los componentes de la función h son admisibles, entonces por su definición ella también lo es. Por la misma razón, se puede afirmar que h es dominante sobre cualquiera de las n heurísticas que la componen.

En las secciones sucesivas se describen varias heurísticas admisibles para el problema del *N-Puzzle* presentado en la Sección 2.3, el cual forma parte del objeto de estudio de esta investigación.

2.4.1. Distancia Manhattan

En el juego del *N-Puzzle*, para poder mover una ficha de su posición es necesario que una de las casillas adyacentes no contenga ficha, es decir, que sea la casilla vacía. Si se ignora esta restricción, se obtiene un modelo simplificado del juego original en donde se puede mover cualquier ficha hasta cualquier posición adyacente, permitiendo inclusive que varias de ellas ocupen simultáneamente la misma posición del tablero.

Dada esta libertad de movimiento, se estaría eliminando la necesidad de interacción entre las fichas del problema. Debido a esto, el costo de resolver óptimamente el problema según este nuevo modelo consistiría simplemente en contar el número mínimo de movimientos necesarios para llevar cada ficha del tablero desde su posición inicial hasta la casilla que le corresponde en el estado solución. A esta operación se le denomina distancia *manhattan* es análoga al concepto geométrico entre dos puntos que lleva el mismo nombre.

Una vez que se tiene la solución óptima de la versión simplificada del problema, ésta representa una cota inferior del costo de resolver el problema original, razón por la cual puede ser utilizada como una heurística admisible para el *N-Puzzle*.

Sin embargo, debido a que esta estrategia no considera la interacción entre las fichas del tablero, los valores heurísticos calculados no resultan tan precisos como uno pudiese desear [Russell y Norvig, 2003]. Por lo tanto, su utilidad es cuestionable para versiones de este juego como el *24-Puzzle*, el cual cuenta con un espacio de búsqueda de aproximadamente 10^{25} estados. De hecho, según predicciones hechas en [Korf y Felner, 2002], el tiempo promedio de resolución de una instancia de este problema usando distancia *manhattan* como función heurística sería aproximadamente de 50.000 años.

Como respuesta a esta problemática, [Culberson y Schaeffer, 1996] desarrollaron una estrategia que toma en cuenta la interacción entre los elementos del tablero. Esta técnica

consiste en tomar un subconjunto de las fichas y calcular el costo asociado a solucionar el problema que surge de cada posible distribución de ellas en el tablero, suponiendo que el resto de las fichas son indistinguibles. Cada una de estas posibles distribuciones del conjunto de fichas escogidas se denominada **un patrón**, y los valores que se calculan en relación a ellos se almacenan en una base de conocimiento que se denomina **base de datos de patrones**, o *PDB* por sus siglas en inglés, que servirá como función heurística admisible para el *N-Puzzle*.

2.5. Bases de Datos de Patrones

Como se mencionó anteriormente, las bases de datos de patrones surgen de resolver un conjunto de subproblemas del problema original. Por ejemplo, en la Figura 8 se puede observar el subproblema generado a partir de una instancia cualquiera del *15-Puzzle*, donde los elementos indistinguibles junto con el patrón correspondiente a las fichas $\{1, 2, 3, 4, 5, 6, 7\}$ deben ser reorganizados en el menor número de pasos posibles hasta formar el patrón objetivo.

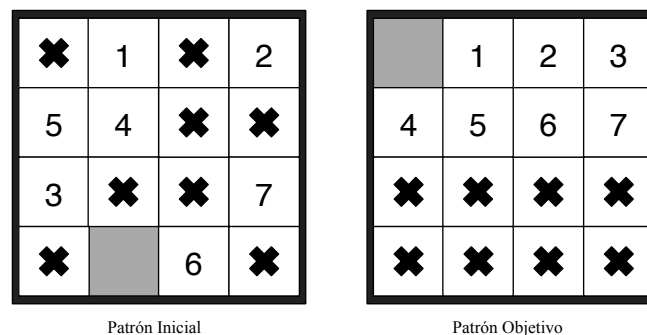


Figura 8: Ejemplo de un patrón de siete fichas para el *15-Puzzle*

Según [Culberson y Schaeffer, 1996], los conceptos de patrón y patrón objetivo se definen de la siguiente manera:

Definición 9. *Un patrón es una especificación parcial de un estado del problema, en donde sólo se establece la posición de algunas fichas.*

Definición 10. *El patrón objetivo es la especificación parcial del estado que representa la*

solución del problema a considerar.

El objetivo de las bases de datos de patrones es precalcular la longitud del camino más corto desde cada posible patrón hasta el patrón objetivo o, lo que es lo mismo, calcular el mínimo número de acciones necesarias para transformar la configuración inicial del patrón en la configuración objetivo del mismo. Este procedimiento se hace a través de un análisis retrógrado del problema, partiendo desde el patrón objetivo y utilizando las acciones válidas para generar todos los patrones alcanzables desde éste. Es preciso señalar que los movimientos realizados sobre los elementos que no pertenecen al patrón son tomados en cuenta al momento de construir las bases de datos.

Como se mencionó anteriormente, cuando se tiene una colección de heurísticas admisibles se puede tomar el máximo valor generado por ellas y esto seguirá comportándose como una cota inferior al costo de resolver el problema original sin simplificaciones. Basándose en esto, [Culberson y Schaeffer, 1996] generaron dos bases de datos de patrones diferentes y utilizaron como heurística el máximo de los valores arrojados por ambas *PDB* para cada caso en particular. En la Figura 9 se detallan los patrones objetivo escogidos por los autores para generar las bases de datos de patrones.

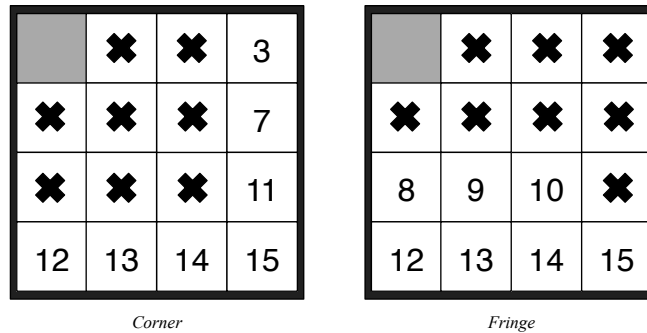


Figura 9: Patrones objetivo en las bases de datos denominadas *Corner* y *Fringe*

Por otro lado, el número de fichas escogidas para formar parte del patrón es un factor importante en la definición de una *PDB*, ya que influye tanto en el tamaño final de la misma como en la precisión de los valores heurísticos obtenidos para cada patrón. Por ejemplo, si se decidiera escoger un patrón que contiene todas las fichas del tablero, se estaría obteniendo

una *PDB* que almacena el costo exacto de resolver todas las configuraciones posibles del problema original. Es evidente que el espacio necesario para precalcular todos estos valores es inmanejable para problemas como el *15-Puzzle* y el *24-Puzzle*, por lo que la limitante a considerar al momento de elegir el patrón objetivo a partir del cual se va a generar la *PDB* es el espacio en memoria necesario para almacenarla.

Los ejemplos mostrados anteriormente en las Figuras 8 y 9 constan de patrones de 7 fichas más la posición del blanco. Para determinar el número total de patrones existentes se debe contar de cuántas formas se pueden colocar 8 elementos distinguibles en 16 casillas diferentes, lo cual es equivalente a 16^8 . Por lo tanto, se necesitaría almacenar el valor para 518.918.400 de patrones diferentes, lo que se traduce en disponer de por lo menos 500MB de memoria RAM por cada *PDB* de este tamaño que se desee utilizar, asumiendo que cada entrada de las mismas ocupa un *byte*.

Análogamente, si se quisiera construir una base de datos de patrones tomando en cuenta 8 fichas más la posición del blanco se tendría que considerar un total de 4.151.347.200 de patrones diferentes, lo que significaría que se necesita disponer de aproximadamente 4GB de memoria RAM para poder almacenarla, lo cual resulta ser complicado de manejar en la práctica.

Es importante destacar que cada *PDB* se calcula una sola vez para cada patrón objetivo escogido y el costo de este cálculo es amortizado a la resolución de múltiples instancias que persigan el mismo estado objetivo.

Una vez que las bases de datos de patrones han sido almacenadas en memoria, se utiliza el algoritmo *IDA** para buscar la solución óptima de cualquier instancia válida del problema. Como función heurística para el algoritmo se consulta, para cada estado alcanzado n , la *PDB* con las posiciones del blanco y de las fichas pertenecientes al patrón en n , como parámetro. Por medio de esta estrategia, [Culberson y Schaeffer, 1996] lograron resolver instancias del *15-Puzzle* generando 1000 veces menos nodos y reduciendo el tiempo de corrida en un factor de 12, en comparación con utilizar distancia *manhattan* como función heurística.

Sin embargo, como mencionan [Korf y Felner, 2002], esta propuesta sigue sin resultar

factible para enfrentar problemas como el 24 -Puzzle, donde una PDB generada a partir de 6 fichas y el blanco tiene un total de 25^7 entradas, lo que implica disponer de más de 2GB de memoria RAM por cada PDB que se desee almacenar. Sumado a esto, es posible que los valores arrojados por una PDB como ésta, que solamente toma en cuenta 6 fichas de las 24 existentes, sean menores que la sumatoria de las distancias *manhattan* de todas las fichas del tablero.

Por esta razón, los mismos autores proponen un nuevo esquema para la construcción de bases de datos de patrones. A diferencia de las anteriormente explicadas, éstas sólo consideran los movimientos de las fichas pertenecientes al patrón para totalizar el costo asociado al mismo. Luego, será posible sumar los valores de varias PDB para obtener la heurística de un estado, siempre y cuando ninguna ficha pertenezca a más de un patrón. A este nuevo método se le denomina **bases de datos de patrones aditivas**.

2.5.1. Bases de Datos de Patrones Aditivas

Para construir este tipo de bases de datos usando como caso de estudio el N -Puzzle, se particiona el conjunto de fichas en grupos disjuntos de tal forma que ninguna ellas pertenezca a más de un grupo. Posteriormente se precalcula el mínimo número de movimientos necesarios para llevar las fichas pertenecientes a cada agrupación hasta su posición en el estado objetivo, para cada una de las posibles configuraciones alcanzables [Korf y Felner, 2002]. Es importante recordar que en este tipo de base de datos solamente se toman en cuenta los movimientos realizados por las fichas pertenecientes al patrón que se está construyendo. En la Figura 10, se muestra una posible partición para el 15 -Puzzle y el 24 -Puzzle.

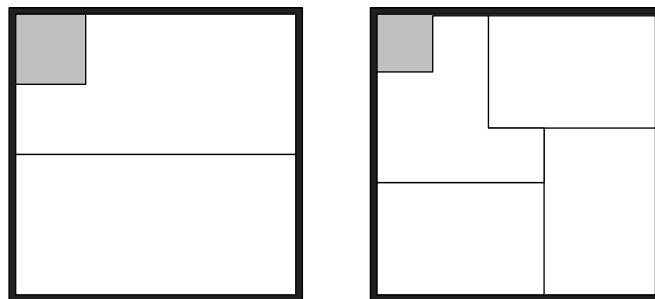


Figura 10: Ejemplo de bases de datos aditivas para el 15 -Puzzle y el 24 -Puzzle

Como se mencionó anteriormente, la principal diferencia entre las bases de datos de patrones propuestas por [Culberson y Schaeffer, 1996] y las bases de datos de patrones aditivas es que las primeras toman en cuenta todos los movimientos de las fichas del tablero a la hora de calcular cual es el costo asociado a cada patrón, incluso los movimientos que afectan aquellas fichas que resultan ser indistinguibles. En consecuencia, al considerar varias *PDB* de este estilo, aun cuando los conjuntos de fichas elegidos para la construcción de las mismas sean disjuntos, se están adoptando movimientos comunes en cada una de ellas, perdiendo la condición de admisibilidad en caso de que se quiera construir una nueva heurística a partir de la suma de las generadas por cada base de datos. Esto no sucede en el planteamiento hecho por [Korf y Felner, 2002], en donde al sólo contar los movimientos de las fichas pertenecientes a cada patrón, no se va a considerar dos veces la misma acción en diferentes bases de datos. Razón por la cual es posible sumar los valores obtenidos y crear una nueva heurística que mantenga la condición de admisibilidad.

Una segunda diferencia entre estos dos planteamientos es que en las bases de datos disjuntas el blanco no afecta el número de entradas posibles, ya que éste no es considerado a la hora de almacenar el costo asociado a un patrón determinado. Por esta razón esta propuesta resulta ser más manejable en cuanto a memoria se refiere, dando la posibilidad de generar bases de datos tomando en cuenta un número mayor de fichas.

Por otra parte, es importante aclarar que si bien la posición del blanco no es tomada en cuenta para almacenar los valores asociados a cada patrón en la base de datos, ésta sí debe ser considerada durante el proceso necesario para calcular este costo, ya que como consecuencia se generan mejores valores heurísticos.

Esto se debe a que en el caso de no tomar en cuenta la posición del blanco, el tablero en el problema relajado sólo constaría de posiciones vacías y las fichas pertenecientes al patrón, dando la posibilidad de que una determinada configuración genere como sucesores a patrones que pudiesen ser encontrados por primera vez más adelante en el árbol del problema, obteniendo un valor heurístico más acertado. Esto se ve reflejado en el ejemplo mostrado en la Figura 11, en donde en el caso de no considerar el blanco, es posible mover la ficha 3

hacia arriba, mientras que de lo contrario, sería imposible realizar esa acción sin alterar la posición de ninguna otra ficha del patrón.

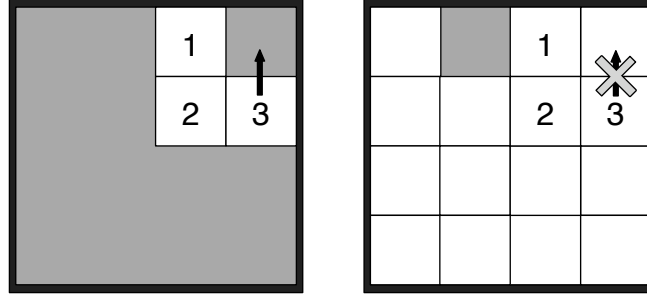


Figura 11: Diferencia entre no considerar y considerar la posición del blanco

Por esta razón, una vez que el tablero se ha dividido en k grupos disjuntos, los cuales llamaremos G_1, \dots, G_k , la transformación con la cual se obtendrá el patrón correspondiente a cada grupo a partir de una instancia cualquiera del problema original pudiese ser definida de la siguiente manera:

$$\varphi_i(x) = \begin{cases} x & \text{si } x \in G_i, \\ \text{blanco} & \text{si } x = \text{blanco}, \\ \text{indistinguible} & \text{cualquier otro caso.} \end{cases} \quad (4)$$

2.5.2. Compresión de Bases de Datos de Patrones

Las bases de datos de patrones que toman en cuenta un mayor número de fichas del tablero proveen mejores valores heurísticos, dado que toman en cuenta la interacción entre más elementos. El principal problema de las mismas es que, a medida que crece el número de fichas consideradas, el tamaño de las bases de datos crece aún más. Por ejemplo, para el *15-Puzzle* una *PDB* de 8 fichas requiere un poco menos de 500MB para su almacenamiento, mientras que de aumentar ese número en una unidad a 9 fichas incrementa ese requerimiento a casi 4GB. Este último número representa una cantidad de memoria que no se dispone en los equipos utilizados para los experimentos, por lo tanto resulta impráctico.

Una manera de solucionar el problema del planteamiento anterior es comprimir las bases

de datos de patrones; esto puede realizarse de varias maneras. Un método propuesto en [Korf y Felner, 2002] propone, por ejemplo, para un patrón de tamaño $k + 1$, compuesto de k fichas y el blanco, almacenar en la *PDB* solamente el mínimo valor de todas las posibles $N - k$ posiciones del blanco, siendo N el tamaño del tablero. Con esto se logra almacenar dicha base de datos en el espacio que requeriría una de k elementos, conservando parcialmente la interacción de la ficha adicional y preservando la admisibilidad de los valores. Este proceso de compresión se lleva a cabo durante la generación de las bases de datos patrones.

2.5.3. Indexación de Bases de Datos de Patrones

Un aspecto importante respecto a la utilización de las bases de datos de patrones es la escogencia del método de almacenamiento. La solución más ingenua es utilizar un arreglo k -dimensional, donde k es el tamaño del patrón, y cada dimensión debe almacenar las n posibles posiciones de cada ficha en el tablero. Por ejemplo, para el caso del *15-Puzzle* y una *PDB* con patrón de 8 fichas esto se traduce en un arreglo de 16^8 entradas, el cual ocuparía en memoria 4GB si cada elemento se almacena en un *byte*. Como se había mencionado anteriormente, esta cantidad resulta impráctica y, además, no considera la nueva estrategia planteada en la sección anterior, ya que ésta implica un patrón de 8 fichas más el blanco. Asimismo, otra desventaja de esta opción es la cantidad de espacio que desperdicia, especialmente para patrones de gran tamaño como el mencionado en el ejemplo, ya que como se verá más adelante sólo se requiere de un poco menos de 500MB para almacenar la *PDB* completamente.

Una segunda posibilidad es la que [Felner *et al.*, 2004b] denomina “*packed mapping*”, o correspondencia compacta, la cual busca almacenar el número exacto de entradas que debe tener la *PDB*. Para el ejemplo anterior, esto resulta equivalente a un arreglo mono-dimensional de $16 \times 15 \times \dots \times 9$ entradas (518.918.400 *bytes* o 495MB). La explicación para el cálculo de esta cantidad es muy sencilla. Una vez puesta la primera ficha en alguna de las 16 casillas del tablero, sólo quedarán 15 posibles posiciones donde colocar la siguiente, y así sucesivamente.

Para poder acceder a este arreglo es necesario el uso de funciones de correspondencia

entre permutaciones y valores numéricos únicos, las cuales deben tener un buen desempeño para no perjudicar el rendimiento de algoritmos como *IDA**, debido a que el acceso a las *PDB* depende únicamente del valor que se calcula en ellas. Por esta razón, en las siguientes secciones se describen varias de estas funciones, todas estudiadas y probadas con anterioridad, que sirvieron como base para la implementación realizada en este trabajo.

Enumeración no Lexicográfica de Permutaciones

La primera técnica utilizada en este trabajo para enumerar permutaciones fue desarrollada por [Myrvold y Ruskey, 2001], y como resultado de su utilización se obtiene un ordenamiento no lexicográfico de las mismas. En dicho trabajo no sólo se provee del algoritmo necesario para realizar esta operación, sino que también se provee del procedimiento para llevar a cabo la operación inversa, es decir, dado un valor numérico se puede determinar la permutación asociada a él. Tal y como afirman los autores, el orden final de las permutaciones no resulta muy sencillo de comprender, razón por la cual esta explicación se deja fuera del alcance de este trabajo. En la Figura 12 se pueden detallar ambos algoritmos.

Para el caso particular del algoritmo de enumeración de permutaciones (Figura 12(a)), los parámetros de entrada son los siguientes: n es el tamaño de la permutación, π es el arreglo que contiene la permutación, y π^{-1} es el arreglo de las posiciones de los elementos de la permutación (la primera casilla contiene la posición del primer elemento, y así sucesivamente). Asimismo, los parámetros para el algoritmo inverso son: n es el tamaño de la permutación, r es el valor numérico que indentifica la permutación, y π es el arreglo donde se desea reproducir y almacenar la permutación.

La importancia de estos algoritmos para este trabajo radica en que para la generación de las bases de datos de patrones, debido a limitaciones en cuanto a memoria disponible que serán explicadas en el Capítulo 3, fue necesario realizar tanto el proceso de enumeración como el proceso inverso. Adicionalmente, ambas funciones garantizan una complejidad espacial y temporal $O(n)$, lo que resulta beneficioso para el desempeño de los algoritmos utilizados en los experimentos.

Algoritmo 2.5.1: RANK(n, π, π^{-1})

```

if ( $n = 1$ )
  then return (0)
 $s \leftarrow \pi[n - 1]$ 
SWAP( $\pi[n - 1], \pi[\pi^{-1}[n - 1]]$ )
SWAP( $\pi^{-1}[s], \pi^{-1}[n - 1]$ )
return ( $s + n \cdot \text{RANK}(n - 1, \pi, \pi^{-1})$ )
  
```

(a) Algoritmo de enumeración

Algoritmo 2.5.2: UNRANK(n, r, π)

```

if ( $n > 0$ )
  then  $\begin{cases} \text{SWAP}(\pi[n - 1], \pi[r \bmod n]) \\ \text{UNRANK}(n - 1, \lfloor r/n \rfloor, \pi) \end{cases}$ 
  
```

(b) Algoritmo inverso

Figura 12: Algoritmos de correspondencia no lexicográfica entre permutaciones y números enteros

Enumeración Lexicográfica de Permutaciones

Una correspondencia lexicográfica entre permutaciones y valores numéricos es equivalente a ordenar permutaciones por su representación numérica en orden alfabético o lexicográficamente [Korf y Schultze, 2005]. Por ejemplo, para una permutación de tres elementos se desea obtener los valores presentados en el Cuadro 3.

Permutación	012	021	102	120	201	210
Valor numérico	0	1	2	3	4	5

Cuadro 3: Correspondencia entre permutaciones y valores numéricos únicos

En [Korf y Schultze, 2005] se describe un algoritmo que permite representar permutaciones por medio de números enteros en el rango desde 0 hasta $n! - 1$ inclusive, siendo n el tamaño de la permutación. Para comprender este algoritmo es necesario tener a la mano la siguiente definición:

Definición 11. Un número en base factorial tiene la forma $d_{n-1} \cdot (n-1)! + d_{n-2} \cdot (n-2)! +$

$\dots + d_1 \cdot 1!$, con $0 \leq d_i \leq i$.

Para construir la correspondencia deseada son necesarios los siguientes pasos:

1. Crear una correspondencia entre la permutación y una secuencia de dígitos en base factorial. Para realizar esta operación se debe restar a cada componente de la permutación el número de elementos menores a él ubicados a su izquierda. Aplicando este procedimiento al ejemplo anterior resulta en los siguientes valores: 012-000, 021-010, 102-100, 120-110, 201-200, 210-210.
2. Transformar los dígitos en base factorial a un número entero. Para llevar esto a cabo simplemente se realizan las operaciones aritméticas descritas en la definición de número en base factorial. El resultado de estas operaciones es el descrito en el Cuadro 3.

La importancia del uso de una enumeración lexicográfica radica en lo siguiente: accesos consecutivos a una base de datos de patrones resulta en un conjunto de permutaciones sucesivas que comparten un gran número de elementos entre sí. Por lo tanto, producto del uso de una enumeración lexicográfica, los índices correspondientes a dichas permutaciones tenderán a estar relativamente cerca, y esto traerá como beneficio que los accesos a memoria gocen de localidad espacial, es decir, elementos cercanos al último consultado tendrán una alta probabilidad de estar precargados en memoria caché, resultando en una mejora substancial del desempeño del programa.

Enumeración de Combinaciones

Para este trabajo se utilizará la propuesta de [Knuth, 2005] para la enumeración de combinaciones, donde se plantea una estrategia para enumerar lexicográficamente todas las posibles formas de tomar t elementos de un conjunto de tamaño n .

Una combinación puede ser representada al listar todos los elementos $c_t \dots c_2 c_1$ que han sido seleccionados. Es conveniente suponer que el conjunto sobre el cual se van a elegir estos t elementos está representado por los números entre 0 y $n - 1$ inclusive, en consecuencia, al listar los elementos pertenecientes a la combinación en orden decreciente, se cumple que:

$$n > c_t > \dots > c_2 > c_1 \geq 0 \quad (5)$$

Una vez aclarado todo esto, sólo queda considerar el siguiente teorema propuesto por el autor en la investigación mencionada anteriormente:

Teorema 1. *Al generar las combinaciones considerando un orden lexicográfico, el número de combinaciones visitadas antes de la combinación $c_t \dots c_2 c_1$ es igual a la siguiente expresión:*

$$\binom{c_t}{t} + \dots + \binom{c_2}{2} + \binom{c_1}{1} \quad (6)$$

Luego, una forma para asignar un número único a cada posible combinación de tamaño t sobre un conjunto de n elementos viene dada por la siguiente expresión:

$$\sum_{k=1}^t \binom{c_k}{k} \quad (7)$$

2.5.4. Propiedad simétrica del *N-Puzzle*

Existen ciertas propiedades particulares del *N-Puzzle* que pueden ser usadas para generar nuevas heurísticas a partir de otras previamente definidas. Una de ellas es la que se denomina **simetría con respecto a la diagonal principal**.

Esta propiedad afirma que para todo camino P existe otro camino de igual longitud P' obtenido al reflejar cada una de las acciones que componen a P con respecto a la diagonal principal. Esto es equivalente a aplicar la siguiente transformación a cada uno de los movimientos que forma parte el camino:

$$\delta(x) = \begin{cases} izquierda & \text{si } x = arriba \\ arriba & \text{si } x = izquierda \\ derecha & \text{si } x = abajo \\ abajo & \text{si } x = derecha \end{cases} \quad (8)$$

En la Figura 13, se muestra un ejemplo de aplicar esta transformación a un camino P cualquiera.

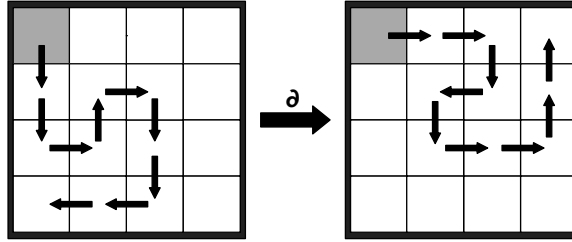


Figura 13: Reflejo de un camino

Por medio de esta función, si se tiene el camino que lleva a un estado cualquiera del problema, resulta sencillo conseguir su reflejo respecto a la diagonal principal. La dificultad surge cuando se desea construir el reflejo asociado a un tablero, sin disponer de la secuencia de acciones que lo generó.

Según [Culberson y Schaeffer, 1996], si se toma el caso del *15-Puzzle*, se pudiese definir el reflejo respecto a la diagonal principal de la siguiente manera:

$$D = \langle 0 \ 4 \ 8 \ 12 \ 15 \ 9 \ 13 \ 2 \ 6 \ 10 \ 14 \ 3 \ 7 \ 11 \ 15 \rangle$$

De este modo D actúa como un espejo, induciendo las sustituciones hechas por la transformación δ para cada movimiento sobre cualquier camino P . Adicionalmente, los autores plantean que calcular el reflejo p' de un estado p cualquiera, es equivalente a resolver la siguiente composición:

$$p' = D \circ p \circ D \tag{9}$$

La primera parte de esta expresión, $p \circ D$, significa que las posiciones del tablero son permutadas primero por D y posteriormente son correlacionadas a las fichas correspondientes por p . Finalmente, al aplicar D por la izquierda, el resultado es asignar a cada una de las fichas al valor que le compete en el tablero original. En el ejemplo mostrado en la Figura 14, se pueden observar las etapas del proceso para calcular el tablero reflejo del estado

$p = \langle 4 \ 1 \ 2 \ 3 \ 8 \ 6 \ 10 \ 7 \ 9 \ 5 \ 14 \ 11 \ 0 \ 12 \ 13 \ 15 \rangle$, generado a partir de las acciones que se detallan en la Figura 13.

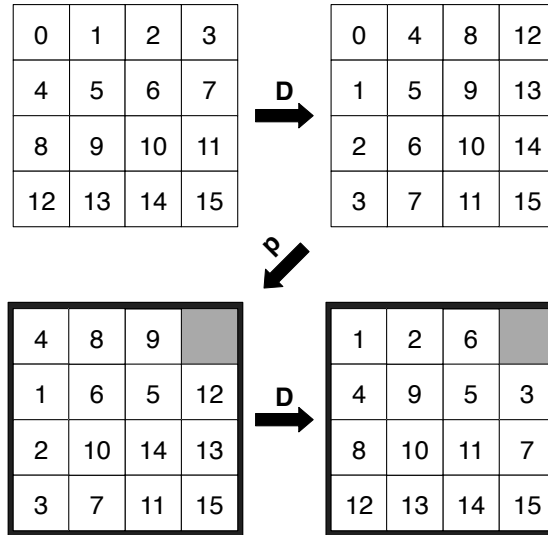


Figura 14: Cálculo del reflejo $p' = D \circ p \circ D$

En lo que respecta a la presente investigación, la propiedad más interesante que comparten p y p' , es que ambos se encuentran a la misma distancia con respecto al estado solución. Por lo que cualquier heurística admisible para p lo será también para p' y viceversa [Korf y Felner, 2002]. Por esta razón, para calcular el valor heurístico asociado a un estado, primero se calcula su reflejo, luego se consultan ambos en las bases de datos de patrones y, finalmente, se toma el máximo de los valores obtenidos. Este proceso es equivalente a consultar un estado en dos *PDB* diferentes, donde una es el reflejo de la otra. Con esta estrategia, se estaría obteniendo la información equivalente a tener dos bases de datos, pero con la ventaja de tener que almacenar en memoria una sola de ellas. En la Figura 15 se puede observar una configuración de bases de datos de patrones aditiva y su reflejo para el *24-Puzzle*, utilizando una partición de 4 grupos de 6 fichas cada uno.

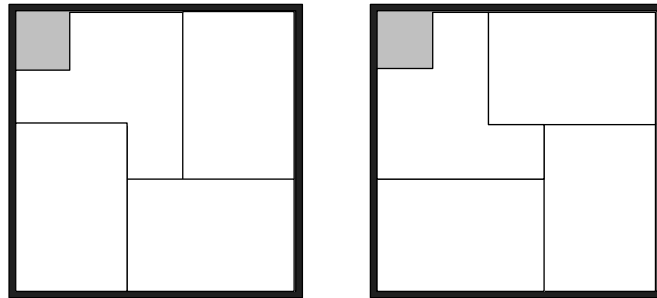


Figura 15: Base de datos de patrones aditivas para el *24-Puzzle* y su reflejo

Capítulo 3

Implementación

En este capítulo se describen las modificaciones y optimizaciones particulares que se le hicieron a los algoritmos utilizados durante la etapa experimental de la investigación, todos previamente especificados y analizados en el capítulo anterior. Para llevar a cabo este proceso de implementación y mejoramiento se aprovecharon propiedades particulares del *N-Puzzle* y su espacio de búsqueda, las cuales se describen en detalle en las próximas secciones. Finalmente, es importante mencionar que toda la implementación del proyecto fue codificada en el lenguaje de programación C++, sin la utilización de ninguna librería fuera del estándar del mismo.

3.1. Compresión por Regiones de Blancos

El problema con la estrategia planteada por [Korf y Felner, 2002] para la compresión de las bases de datos de patrones, es que a pesar de tomar en cuenta una ficha extra, los valores finales de la *PDB* sólo reflejan parcialmente la interacción adicional, es decir, se pierde cierta cantidad de información.

Una estrategia que tome en cuenta las dos ideas planteadas en la Sección 2.5.2, al mismo tiempo que solucione la problemática de cada una de ellas, resultaría en mejores valores heurísticos sin el inconveniente de ser impráctica. Una nueva estrategia que busca alcanzar este objetivo se detalla a continuación y constituye el principal aporte de este trabajo.

Debido a una particularidad de la ficha blanca se desarrolló un procedimiento que consigue una alta compresión de las bases de datos de patrones y por ejemplo, para el caso del *15-Puzzle*, una *PDB* de 9 elementos (8 fichas más el blanco) se logró comprimir en tan sólo 1.38GB, un espacio mucho menor a los 4GB estimados originalmente. Esta peculiaridad se refiere a que para la mayoría de las configuraciones del tablero, si se considera la ubicación de los componentes del patrón y las fichas indistinguibles, varias posiciones del blanco pudiesen

agruparse y almacenarse como un solo elemento en la *PDB*, debido a que los movimientos entre el blanco y las fichas indistinguibles no son tomados en cuenta en el costo final asociado a un patrón. A este método se le denominó **compresión por regiones de blancos**, debido a que las regiones que forman las fichas indistinguibles adyacentes se consideran como una única posición válida para el blanco. Para el resto de esta sección se utilizará la siguiente notación: el método que no toma en cuenta la posición del blanco será denominado h_1 , y el método de las regiones de blanco será h_2 .

Definición 12. *Una región de blancos es un conjunto de fichas indistinguibles adyacentes donde la casilla vacía puede moverse libremente sin interactuar con las fichas del patrón.*

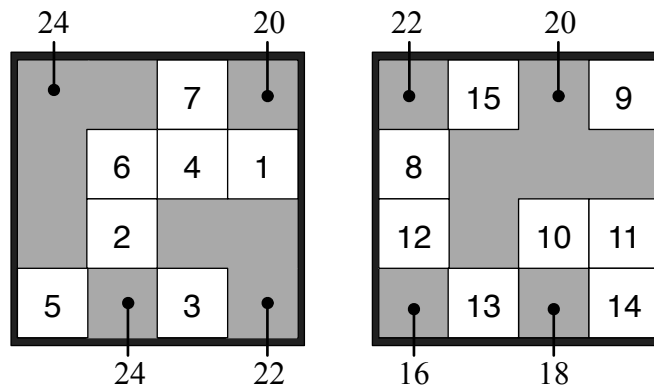


Figura 16: Valores de la *PDB* con el método de compresión por regiones de blancos

Si se toman como ejemplo las configuraciones del *15-Puzzle* desplegadas en la Figura 16, se puede observar que no existe la necesidad de almacenar un valor distinto para cada posible posición válida del blanco, basta con agrupar los conjuntos de fichas indistinguibles adyacentes en 4 regiones de blancos para ambos casos y se habrá logrado reducir el espacio de almacenamiento a menos de la mitad para el primer patrón (de 9 posibles posiciones a 4 regiones) y exactamente a la mitad para el segundo (de 8 a 4).

No sólo este nuevo método ofrece una reducción significativa del tamaño de las bases de datos, sino que lo hace sin la pérdida de información incurrida por el procedimiento h_1 propuesto en [Korf y Felner, 2002]. Esto último se logra dado que para cada patrón se almacenan los valores correspondientes a cada región de blancos, y no únicamente el mínimo

de todos ellos, como si sucede para h_1 . El valor asociado a cada una de las regiones de blancos de un patrón es equivalente al costo del camino más corto que va desde el patrón objetivo hasta el patrón en cuestión, con el blanco ubicado en algunas de las casillas pertenecientes a cada región, según sea el caso. Estos valores se calculan durante la generación de las bases de datos de patrones, procedimiento que se detalla en la Sección 3.3.

Como se observó en la Figura 16, dependiendo de la región en la que esté ubicado el blanco, el valor de la heurística h_2 puede variar significativamente, más aún cuando se toma en cuenta la suma de las bases de datos (ver Figura 17). Por el contrario, el método h_1 ofrece el mismo valor heurístico independientemente de la ubicación del blanco, ya que sólo se almacena el mínimo costo del patrón. El método de regiones de blancos preserva el principio de admisibilidad, dado que solamente tienen costo asociado los movimientos entre el blanco y las fichas del patrón; por lo tanto, es posible sumar varias *PDB* siempre y cuando los conjuntos de fichas de los patrones sean disjuntos.

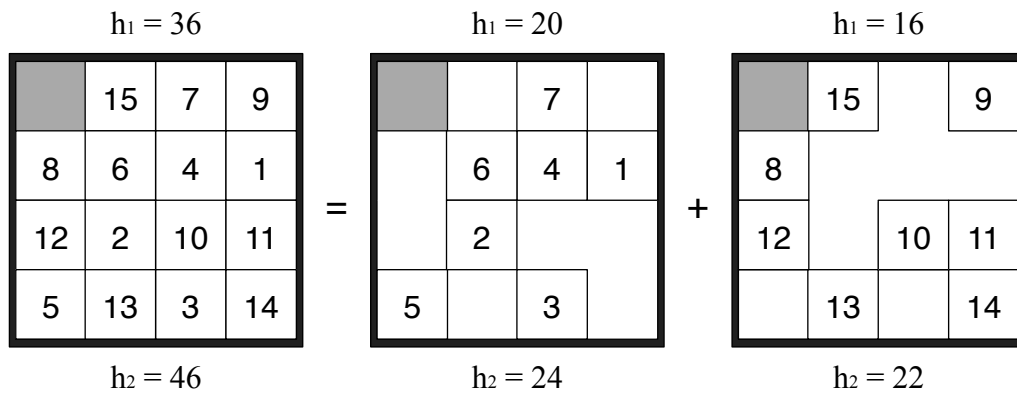


Figura 17: Diferencia de valores en la *PDB*: método sin blanco h_1 vs. regiones de blancos h_2

Número de regiones	1	2	3	4	5	6	7	8
Porcentaje de patrones	11.45	31.46	35.24	17.03	4.45	0.91	0.06	0.02

Cuadro 4: Distribución de regiones de blancos para el *15-Puzzle* y un patrón de 8 elementos

Finalmente, en el Cuadro 4 se detalla para el caso particular del *15-Puzzle* y un patrón de 8 elementos, el porcentaje total de patrones agrupados según el número de regiones de blancos que posee cada uno. Aquí se puede observar claramente el por qué del alto nivel

de compresión que se logra con esta nueva estrategia: si suman los primeros cuatro valores del cuadro se obtiene que aproximadamente el 94 % de los patrones poseen cuatro o menos regiones de blancos, lo que implica al menos un 50 % de ahorro en espacio para la mayoría de los patrones cuando se agrupan las posiciones del blanco por regiones.

3.2. Enumeración de Instancias del *N-Puzzle*

El proceso de enumeración de las distintas configuraciones de los grupos en los cuales fue dividido el problema es fundamental para la construcción y utilización de bases de datos de patrones. Por esta razón, se presta particular atención a este componente de la implementación realizada en este trabajo.

Como se había mencionado en la Sección 2.5.3, es necesario que la enumeración asociada a cada patrón sea única y continua; por lo tanto, si i es el índice correspondiente a un patrón de k elementos en un problema de n posiciones, se puede afirmar que:

$$0 \leq i \leq n^k - 1$$

Es importante que este proceso de enumeración sea ejecutado de la forma más eficiente posible, ya que representa uno de los elementos con mayor frecuencia de utilización durante la búsqueda. Para lograr esto se emplearon los algoritmos previamente detallados en la Sección 2.5.3, los cuales fueron adaptados para operar con estados abstractos, donde un estado abstracto representa una permutación de n elementos en la que solamente k son distinguibles.

3.2.1. Enumeración No Lexicográfica

De acuerdo con [Myrvold y Ruskey, 2001], si el algoritmo descrito en la Sección 2.5.3 es detenido en la k -ésima iteración se obtiene un índice único para los elementos ubicados entre las posiciones $n - k$ y $n - 1$ de la permutación. Por lo tanto, este algoritmo se puede utilizar para enumerar estados abstractos si se construye el arreglo π de tal manera que los elementos que forman parte del patrón se almacenen entre las posiciones $n - k$ y $n - 1$

del mismo, mientras que las posiciones de las fichas indistinguibles son almacenadas en las casillas restantes.

Las mismas modificaciones se pueden aplicar sobre el algoritmo inverso para, de esta manera, poder reconstruir estados abstractos a partir de su valor numérico.

3.2.2. Enumeración Lexicográfica

Tal y como se mencionó anteriormente, la enumeración de un patrón de k elementos para un tablero de n casillas varía entre 0 y $n^k - 1$; por el contrario, la enumeración de permutaciones de tamaño n varía entre 0 y $n! - 1$. Esta diferencia viene dada por el número de configuraciones distintas que existen para cada caso. Luego, teniendo esto en cuenta, si se sustituyen las constantes que multiplican a cada dígito en base factorial se obtiene la siguiente expresión que define la enumeración lexicográfica de un patrón cualquiera:

$$\sum_{i=0}^{k-1} d_i \cdot n^i$$

Por otro lado, tomando en cuenta la explicación ofrecida en [Korf y Schultze, 2005], para poder garantizar la ejecución en tiempo lineal ($O(n)$) de este algoritmo es necesario utilizar una estrategia que calcule en tiempo constante los dígitos en base factorial. Para lograr esto se utiliza una cadena de *bits* de tamaño igual al número de elementos del patrón (k). En esta cadena, inicializada en su totalidad con ceros, se van encendiendo los *bits* correspondientes para cada elemento a medida que se van encontrando, considerando que el *bit* que se encuentra más a la izquierda corresponde al primer elemento y el que está más a la derecha para el último.

Luego, para determinar el número de elementos menores ubicados a la izquierda (dígito en base factorial) para el elemento c del patrón, basta con contar el número de unos en los primeros c *bits* de la cadena formada hasta el momento. Para poder realizar esta operación en tiempo constante ($O(c)$), basta con usar estos c *bits* como un índice para un arreglo precalculado que almacena por cada entrada el número de unos presentes en la representación binaria del índice. El tamaño de esta tabla precalculada es de 2^k entradas y, para un patrón

de 3 elementos, por ejemplo, los valores presentados en el Cuadro 5 muestran su contenido.

Índice del arreglo	0	1	2	3	4	5	6	7
Representación binaria	000	001	010	011	100	101	110	111
Valor en el arreglo	0	1	1	2	1	2	2	3

Cuadro 5: Número de unos en la representación binaria del índice

3.2.3. Enumeración de Combinaciones

Para enumerar combinaciones se utilizó el algoritmo descrito en la Sección 2.5.3, con la diferencia de que las combinaciones se representan por medio de una cadena de *bits* $a_{n-1} \dots a_1 a_0$, donde $a_k = 1$ significa que el elemento c_k pertenece a la combinación, es decir, a_k es igual a uno si y sólo si la casilla k del tablero contiene un elemento que forma parte del patrón que se está considerando. La ventaja de esta alternativa es que la construcción de esta estructura puede hacerse en orden lineal, mientras que el ordenamiento de los elementos que pertenecen a la combinación tiene un orden $O(n \log n)$. El componente (1) de la Figura 18 muestra el arreglo de *bits* correspondiente a una instancia del *N-Puzzle* con un patrón de 8 elementos.

Una vez que se construye la cadena de *bits*, el índice de la combinación que ésta representa se puede calcular por medio de la siguiente expresión:

$$\sum_{k=0}^{n-1} a_k \binom{k}{v}$$

donde n es el número de elementos de la combinación y v es igual al número de elementos pertenecientes a la combinación que han sido encontrados hasta la k -ésima iteración o, lo que es lo mismo, $\sum_{i=0}^{k-1} a_i$.

Finalmente, se puede asegurar que el índice asociado a una combinación se puede determinar en orden lineal gracias a este procedimiento.

3.2.4. Enumeración por Regiones de Blancos

Si bien es cierto que en la Sección 3.1 se mencionaron las ventajas que ofrece la estrategia de compresión por regiones de blancos, en ningún momento se indicó cómo se realiza la enumeración de las distintas configuraciones de las fichas del patrón y el blanco. Para llevar a cabo esta operación se implementó un procedimiento que combina varios de los algoritmos planteados en la Sección 2.5.3, y que se puede detallar en general en la Figura 18.

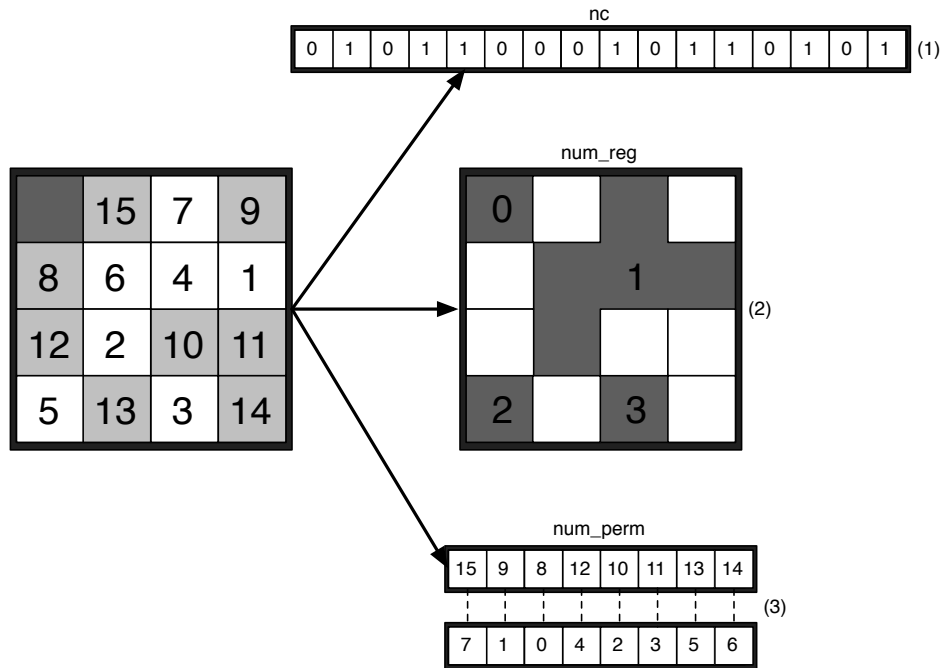


Figura 18: Enumeración de estados por regiones de blancos

Como se explicó previamente una región de blancos es un conjunto de fichas adyacentes que no pertenecen al patrón y en donde el blanco puede moverse libremente interactuando solamente con las fichas indistinguibles. Cada disposición diferente de las fichas distinguibles en las casillas del tablero induce una partición de r regiones de blancos distintas.

Tomando en cuenta que para un tablero de n posiciones y un patrón de k elementos existen $\binom{n}{k}$ maneras de escoger las posiciones en las cuales se van a ubicar las fichas del patrón, y para cada una de ellas se tienen $k!$ formas diferentes de colocar las fichas en estas posiciones y r regiones distintas de blancos, se construyó la siguiente expresión para enumerar los distintos estados del espacio de búsqueda:

$$\mathbf{index} = \mathbf{index_acc}(nc) + \mathbf{num_reg}(nc, pos_blanco) \cdot k! + \mathbf{num_perm}(tablero)$$

donde:

- nc es el número de la combinación inducida por las posiciones de las fichas del patrón. Este número se calcula por medio del algoritmo detallado en la Sección 3.2.3 utilizando una cadena binaria similar al componente (1) de la Figura 18.
- pos_blanco es la posición de la casilla vacía.
- $tablero$ es la secuencia que representa un estado del N -Puzzle.
- **num_perm** es la función que enumera la permutación de los k elementos del patrón según su orden dentro de las k casillas ocupadas por ellos. Para realizar esta enumeración, se construye una secuencia de tamaño k con los elementos del patrón en el mismo orden en el que aparecen en el tablero, y se realiza una correspondencia entre ellos y números entre 0 y $k - 1$, como se ilustra en el componente (3) de la Figura 18. Esta correspondencia consiste en asignar al menor de los elementos el número 0, al siguiente en orden ascendente el número 1, y así sucesivamente hasta llegar a $k - 1$. Finalmente, se aplica sobre esta secuencia el algoritmo descrito en la Sección 2.5.3 para enumeración lexicográfica de permutaciones.
- **num_reg** es una función que determina la región de blancos en la que se encuentra ubicada la casilla vacía. Para realizar este cálculo se utiliza el número de la combinación nc y la posición del blanco pos_blanco . Con la finalidad de lograr esta operación en orden constante, **num_reg** se implementa como un arreglo de dimensiones $\binom{n}{k} \times n$, donde se precálculan los números de regiones según la posición de la casilla vacía para cada posible combinación. Un ejemplo de esta enumeración de regiones se puede visualizar en el componente (2) de la Figura 18.

- Para una combinación cualquiera con índice nc , **index_acc**(nc) representa el número total de patrones inducidos por las combinaciones enumeradas desde 0 hasta $nc - 1$ inclusive, según el orden establecido por el algoritmo de la Sección 3.2.3. El valor de esta función está dado por la siguiente expresión:

$$\sum_{i=0}^{nc-1} r_i \cdot k!$$

donde r_i es el número de regiones distintas de blancos establecidas por la combinación con índice igual a i . De la misma manera que **num_reg**, esta función se implementó por medio de un arreglo unidimensional precalculado de tamaño $\binom{n}{k}$, donde se guarda el número de regiones para cada posible valor de nc .

A pesar del gran número de operaciones que involucra este procedimiento, el orden de ejecución se mantiene en $O(n)$ gracias a los algoritmos utilizados, dado que tanto la enumeración de la permutación como la de la combinación se ejecutan en orden lineal, y las operaciones con **num_reg** y **index_acc** son de orden constante ya que nada más involucran el acceso a arreglos, por lo tanto, como asegura [Cormen *et al.*, 2001]:

$$O(n) + O(n) + O(c) + O(c) = \mathbf{O(n)}$$

para un c constante.

3.3. Generación de Bases de Datos de Patrones

Como se discutió en la Sección 2.5, una forma de generar las bases de datos de patrones es por medio de lo que se denomina un estudio retrógrado. Como menciona [Felner *et al.*, 2004a], este procedimiento se puede llevar a cabo por medio de un *BFS* en sentido contrario que inicia la búsqueda en el patrón objetivo y finaliza cuando haya generado todas las posibles configuraciones del patrón. A medida que se van construyendo por primera vez los distintos patrones, el número de movimientos necesarios para su generación es almacenado en la *PDB*.

Debido a que las bases de datos de patrones que se desean generar son de tipo aditivas, para efectos del *BFS* los movimientos de las fichas indistinguibles no tienen costo, por lo tanto, los valores finales de la *PDB* sólo reflejan los movimientos entre las fichas del patrón y el blanco.

La primera implementación utilizada del *BFS* se rigió estrictamente por los lineamientos establecidos en la Sección 2.1.1, pero su funcionamiento padecía de un grave problema: para los casos del *15-Puzzle* con patrón de 8 elementos y *24-Puzzle* con patrón de 6 elementos, el tamaño de la cola resultaba tan grande que era inmanejable en memoria principal, ocasionando que la aplicación terminara abruptamente.

La razón de esta falla es que para ciertos niveles de profundidad durante la búsqueda el número de nodos en la cola es muy grande, lo que resulta en una cantidad inmanejable de información en memoria si se toma en cuenta que por cada nodo es necesario almacenar la posición del blanco y de las fichas del patrón al igual que la profundidad a la que fue generado. Particularmente problemático resultó el caso del *15-Puzzle* con patrón de 8 elementos, ya que comprende más de 500 millones de estados sin siquiera tomar en cuenta el blanco.

Para solventar esta problemática se ideó una versión del algoritmo *BFS* que garantiza el uso de una cantidad constante de memoria. Esta cantidad se calcula previo a la ejecución del programa para determinar la factibilidad de la culminación exitosa del mismo. En la siguiente sección se explican las modificaciones realizadas.

3.3.1. *Breadth First Search* Modificado

El principal problema con la implementación tradicional del algoritmo *BFS* es que el tamaño de la cola resulta difícil de predecir y controlar. Para solventar esta situación, se propuso en este trabajo una estrategia que utiliza dos colas binarias de tamaño constante, Q_{actual} y $Q_{siguiente}$, durante todo el proceso de búsqueda.

Durante la ejecución, por cada nivel de profundidad se expanden todos los estados presentes en Q_{actual} y sus sucesores se almacenan en $Q_{siguiente}$. Seguidamente, se vacía Q_{actual} y se intercambian los apuntadores de ambas colas para canjear su contenido. Por último, se

aumenta en una unidad la variable *Prof* que indica la profundidad actual de la búsqueda y se reinicia el proceso para el siguiente nivel. Este esquema general de funcionamiento se puede observar en la Figura 19.

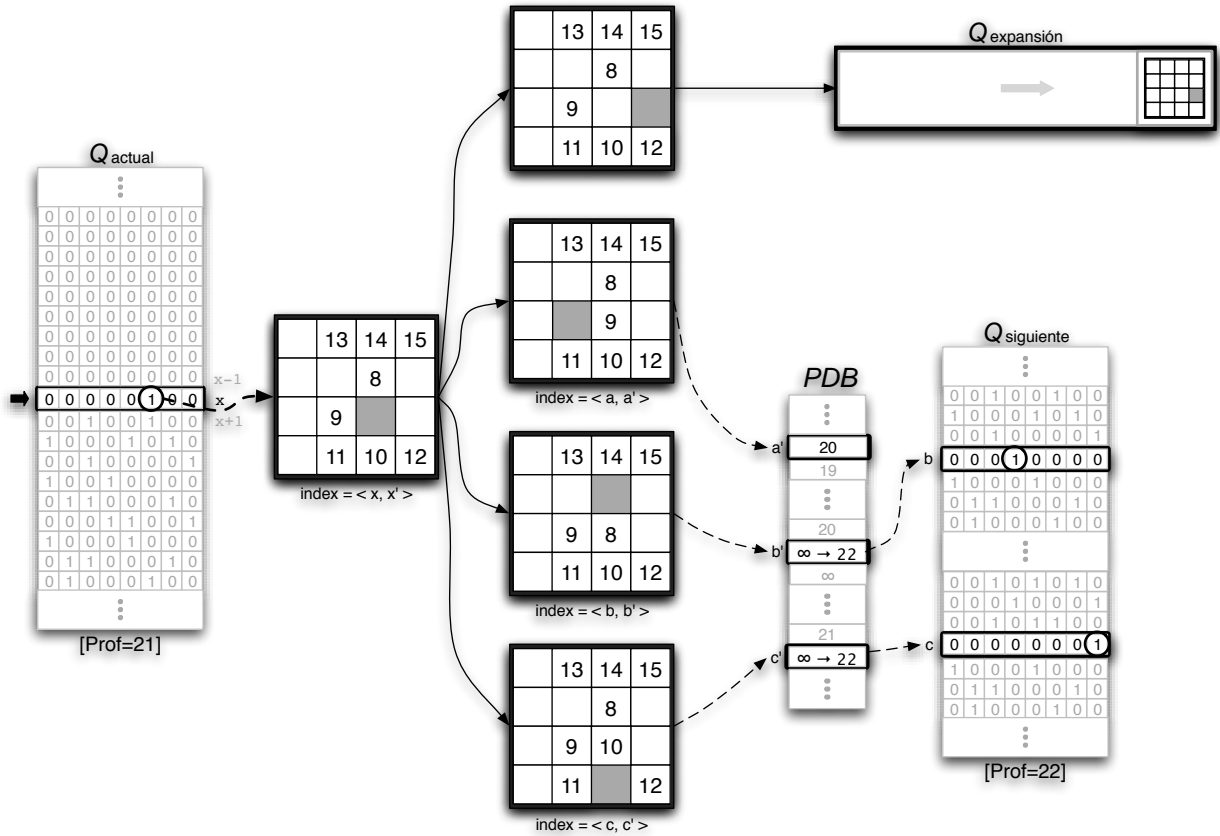


Figura 19: *BFS* modificado para la generación de bases de datos de patrones

Las colas de tamaño constante son implementadas por medio de arreglos que tienen tantas entradas como posibles configuraciones distintas del conjunto de fichas del patrón existen. Cada una de estas entradas consiste en una cadena de $n - k$ bits, donde n es el número de casillas del tablero y k el tamaño del patrón, es decir, cada casilla del arreglo es equivalente a un *bit-string* de tamaño igual a las posibles posiciones válidas del blanco.

Para poder ejecutar el *BFS* sin la necesidad de almacenar la posición de las fichas y el blanco se utiliza la función de enumeración y su inversa descritas en la Sección 3.2.1. Gracias a estas funciones, para un patrón cualquiera basta con almacenar en las colas solamente el número de la posición válida del blanco en la correspondiente cadena de bits, determinada

según la enumeración del patrón. De esta forma, cada posible patrón definido según las posiciones de las fichas y el blanco se almacena utilizando tan sólo un *bit*. Luego, si se multiplica el número total de patrones del espacio de búsqueda y la cantidad de posiciones válidas del blanco, se obtiene el tamaño final de las colas. Por ejemplo, si se toma el caso del *15-Puzzle* y un patrón de 8 elementos, para el que existen 518.918.400 estados, las colas tendrían un tamaño final de 495MB, ya que existen ocho posibles posiciones válidas del blanco por patrón, y cada una ocupa un *bit*.

Durante la búsqueda, para impedir la expansión duplicada de nodos se utilizan dos estrategias distintas según el tipo de bases de datos que se desean generar:

- Para las bases de datos de tipo [Korf y Felner, 2002], donde no se toma en cuenta la posición del blanco para almacenar en la *PDB*, se utiliza una lista de cerrados que se implementa por medio de un arreglo igual al utilizado para las colas.
- Con el método de compresión por regiones de blancos, en donde sí se distinguen los patrones según la posición de la casilla vacía, se utiliza la misma *PDB* como lista de cerrados, tal y como se muestra en la Figura 19.

Por otro lado, cuando se realiza la expansión de un patrón se utiliza una cola adicional (Q_{exp} en la Figura 19) con la finalidad de almacenar temporalmente los nodos generados por la función de sucesores en los cuales no hubo interacción con elementos del patrón. De esta manera, estos nodos se pueden expandir progresivamente para generar aquellos que sí pertenecen a la siguiente profundidad, es decir, donde sí hubo movimiento de alguna ficha del patrón.

Finalmente, es importante aclarar que por cada base de datos que se desea generar se utiliza el índice respectivo para poder almacenar los datos en ella; por el contrario, para el proceso de guardado y extracción de información de las colas binarias siempre se utiliza la función de enumeración no lexicográfica y su inversa explicadas anteriormente.

3.4. Resolución de Instancias del *N-Puzzle*

Una vez generadas las bases de datos de patrones, el próximo paso es utilizar *IDA** (Sección 2.2.2) para resolver instancias válidas del problema. La implementación de este algoritmo se realizó siguiendo fielmente los lineamientos generales previamente propuestos, aprovechando ciertas características del contexto del *N-Puzzle* para definir la función heurística y la función de sucesores, las cuales se describen en detalle en las secciones siguientes.

3.4.1. Cálculo de sucesores

Un estado cualquiera del *N-Puzzle* tiene tantos sucesores como fichas adyacentes hay a la posición que se mantiene libre, donde cada uno de ellos se genera de intercambiar el contenido de estas casillas. Por lo tanto, se puede decir que cada interacción de las fichas representa una acción diferente dependiendo de la dirección del movimiento.

Una característica peculiar del *N-Puzzle* es que todas las acciones son reversibles, es decir, para toda acción existe otra que conduce al estado previo. En consecuencia, al realizar la búsqueda del camino solución, para todo nodo existe la posibilidad de generar a su padre como un sucesor en el grafo, produciendo un aumento considerable en la cantidad de transposiciones, lo que se traduce en elevados números de nodos generados y mayores tiempos de búsqueda.

Si se almacena en memoria la acción que produjo el nodo en expansión, se puede impedir recrear el estado que surge de aplicar la inversa de dicho movimiento. Luego, gracias a esta operación, se evita la generación y posterior expansión de un gran número de nodos previamente visitados.

3.4.2. Construcción de la heurística

La heurística utilizada durante la ejecución del algoritmo *Iterative Deepening A** se construyó a partir de los valores almacenados en las bases de datos de patrones. Como se menciona en la Sección 2.5.1, el conjunto de fichas del *N-Puzzle* se divide en varios grupos disjuntos, dos para el *15-Puzzle* y cuatro para el *24-Puzzle*, y para cada uno de ellos se genera una *PDB*. Luego, durante la realización de la búsqueda, estas bases de datos son consultadas

utilizando los índices asociados a cada patrón y los valores obtenidos se suman para construir la heurística asociada a un estado del problema. Obviamente esta última operación es posible gracias a la propiedad aditiva de la cual gozan estas bases de datos de patrones.

De manera similar, este procedimiento también se lleva a cabo sobre la reflexión del tablero, con el objetivo de obtener una segunda heurística admisible para el mismo estado. Finalmente, de ambos valores generados se toma el máximo para lograr el valor heurístico deseado.

Es importante resaltar que la operación necesaria para generar el tablero reflejo, previamente descrita en la Sección 2.5.4, es realizada una sola vez al inicio de la búsqueda y, durante el resto de la ejecución, este tablero es actualizado por medio de la transformación δ , la cual se encarga de reflejar con respecto a la diagonal principal las acciones aplicadas al tablero no reflejo.

Por otro lado, de acuerdo con lo explicado en la Sección 3.2, el cálculo del índice de un patrón de un estado se ejecuta en tiempo lineal, pero, a pesar de esto, evitar en la medida de lo posible recalcular innecesariamente los índices mejoraría el tiempo de ejecución total de la búsqueda.

Si se toma en cuenta que los grupos de fichas que conforman los patrones de un estado son disjuntos y, que al ejecutar una acción solamente se altera la posición de una ficha, se puede deducir fácilmente que todo estado difiere de sus sucesores en tan sólo un índice. Un ejemplo de esta propiedad se puede observar en la Figura 20, en donde para una instancia del *24-Puzzle* que cuenta con cuatro patrones de seis fichas, todos los tableros sucesores comparten exactamente tres índices con respecto al tablero que los generó. Por esta razón y, a manera de conclusión, se puede afirmar que un estado puede heredar a sus sucesores todos los índices excepto aquel asociado al patrón modificado por la acción realizada, el cual deberá ser recalculado.

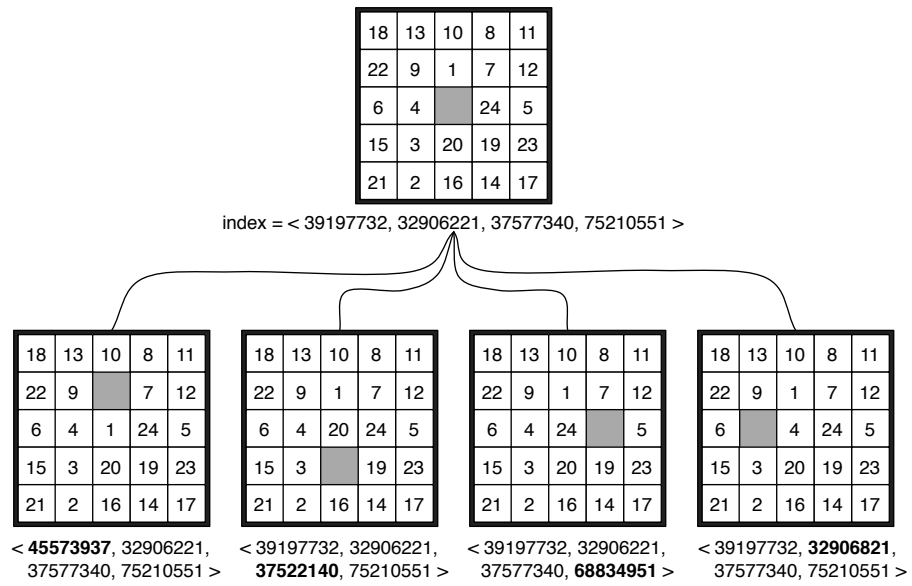


Figura 20: Instancia del *24-Puzzle* y sus sucesores con los índices lexicográficos especificados

Capítulo 4

Experimentos y Resultados

En este capítulo se describe el proceso y los resultados de la experimentación llevada a cabo para evaluar y analizar el desempeño de los algoritmos descritos previamente, especialmente el algoritmo *IDA** que constituye la principal herramienta para la resolución del problema del *N-Puzzle*.

Para llevar a cabo esta evaluación, se diseñó una serie de experimentos que intentan reflejar el proceso que tuvo lugar durante la implementación de la teoría y los algoritmos estudiados.

En primer lugar, se presentan los resultados correspondientes a la generación de las bases de datos de patrones, las cuales representan el principal tópico de estudio de esta investigación. Asimismo, se realiza un análisis de los requerimientos computacionales necesarios para su creación y se estudia comparativamente las distintas estrategias de compresión utilizadas y la calidad de los valores heurísticos obtenidos.

Seguidamente, se lleva a cabo un análisis acerca de los algoritmos de enumeración discutidos en la Sección 3.2, con el fin de mostrar el desempeño particular de cada uno de ellos, además de establecer puntos de comparación que serán de gran utilidad para la discusión acerca de la búsqueda de soluciones óptimas del *N-Puzzle*.

Posteriormente, se analizan las diferentes funciones agregadas al algoritmo *IDA**, de modo de comprender el aporte de cada una de ellas en el desempeño general del algoritmo. Todas estas funciones agregadas al algoritmo son dependientes del dominio del problema, es decir, se derivan de propiedades particulares del *N-Puzzle* y su espacio de búsqueda.

Luego, se presentan los resultados correspondientes a la búsqueda de soluciones óptimas para el *15-Puzzle* y *24-Puzzle*. En base a estos resultados se llevó a cabo un estudio comparativo acerca de la influencia de los distintos métodos de enumeración en el desempeño del algoritmo *IDA**, así como también se establecen las ventajas y desventajas de las distintas

técnicas de compresión de bases de datos de patrones utilizadas.

Para la realización de los experimentos descritos se utilizaron dos computadoras configuradas como se muestra en el Cuadro 6. La primera de ellas, *paris.gia.usb.ve*, se utilizó solamente para la generación de las bases de datos de patrones, debido a su alta capacidad de almacenamiento primario (3GB de memoria RAM). El resto de los experimentos, tanto el de comparación de los algoritmos de enumeración como el de búsqueda de soluciones óptimas del *N-Puzzle*, se llevaron a cabo en la computadora *puipui ldc.usb.ve*, la cual cuenta con componentes más novedosos y de mayor capacidad de cómputo, lo que se traduce en mejores tiempos de ejecución de los experimentos.

Computadora	<i>paris.gia.usb.ve</i>	<i>puipui ldc.usb.ve</i>
Procesador	Intel Pentium 4 2.8Ghz	Intel Xeon 1.86Ghz
Memoria Caché L2	512KB	4MB
Memoria RAM	3GB	2GB
Sistema Operativo	Arch Linux	Arch Linux

Cuadro 6: Configuración de computadoras utilizadas en los experimentos

A continuación se define una serie de abreviaciones utilizadas en este capítulo para simplificar la presentación de los resultados (Ver Cuadro 7).

Abreviación	Significado
RB	Enumeración por regiones de blancos
SP	Enumeración según los elementos del patrón sin la casilla vacía
<i>PDB-k</i>	Base de datos de patrones con patrón de k elementos
AP	Acción previa
CSI	Cálculo de un solo índice
TR	Tablero reflejo
H_0	Valor heurístico inicial

Cuadro 7: Abreviaciones utilizadas en los resultados

4.1. Generación de Bases de Datos de Patrones

Durante el proceso de generación de las bases de datos de patrones, la cantidad de memoria utilizada es un factor que condiciona la correcta finalización del mismo. Debido a que se desconoce la máxima cantidad de estados presentes en cada nivel de profundidad de la búsqueda, no se pueden determinar previamente los requerimientos espaciales mínimos para su ejecución.

Por esta razón, en la Sección 3.3.1 se propuso reemplazar la cola dinámica generalmente utilizada en el algoritmo *BFS*, por colas binarias de tamaño constante, en donde se dispone de un *bit* por cada posible estado a considerar. La ventaja de esta variante es que desde un principio se conocen los requerimientos espaciales necesarios para generar cualquier *PDB*, por lo que puede determinarse previamente la factibilidad del proceso.

En los Cuadros 8 y 9 se puede apreciar la cantidad de memoria y el tiempo necesario para generar cada base de datos de patrones, distinguiéndolas según el tamaño del patrón y el método de enumeración utilizado.

Patrón	Enumeración	Memoria Utilizada			Tiempo
		<i>PDB</i>	Total	<i>PDB</i> /Total	
7	SP	55MB	240MB	0,23	12m57,965s
7	RB	133MB	257MB	0,52	13m53,432s
8	SP	495MB	1980MB	0,25	84m35,161s
8	RB	1359MB	2349MB	0,58	113m30,430s

Cuadro 8: Generación de las *PDB* para el *15-Puzzle*

Patrón	Enumeración	Memoria Utilizada			Tiempo
		<i>PDB</i>	Total	<i>PDB</i> /Total	
6	SP	122MB	988MB	0,12	44m3,769s
6	RB	173MB	750MB	0,23	60m49,924s

Cuadro 9: Generación de las *PDB* para el *24-Puzzle*

De estos resultados presentados es importante destacar que, dado que para generar las *PDB* usando la enumeración por regiones de blanco no se necesita disponer de la lista de

cerrados, entonces el cociente entre la memoria ocupada por la *PDB* y la memoria total utilizada para su generación es aproximadamente el doble en comparación con las *PDB* que no toman en cuenta la casilla vacía. De esta observación se puede afirmar que el método de las regiones de blancos utiliza de manera más eficiente la memoria requerida para el proceso de generación, es decir, aparte de la memoria necesaria para almacenar la *PDB*, la cantidad de memoria adicional indispensable para su generación es mucho menor para el caso de las regiones de blancos. Como se mencionó en la Sección 3.2.4, dado que la enumeración por regiones de blancos toma en cuenta la posición de la casilla vacía, entonces la misma *PDB* sirve como lista de cerrados durante la ejecución del *BFS*.

4.2. Comparación de Bases de Datos de Patrones

Una vez generadas las bases de datos de patrones, la información almacenada en ellas se utiliza para construir los valores heurísticos asociados a diferentes estados del *N-Puzzle*. Cuando se trabaja con *PDB* generadas de maneras diferentes, es interesante comparar la información que se obtiene de cada una de ellas. En este caso se desea comparar las *PDB* generadas por medio de la compresión por regiones de blancos en contraste con las que no toman en cuenta la posición de la casilla vacía. Para realizar esta comparación se crearon aleatoriamente 10 millones de instancias del *15-Puzzle* y *24-Puzzle*, y se consultaron los valores asociados a cada una de ellas en ambas *PDB*. En el Cuadro 10 se observan los porcentajes de las instancias en donde una heurística domina a la otra y donde son iguales, y en el Cuadro 11 se totaliza el número de instancias según la diferencia de valores de ambas *PDB*.

	RB > SP	RB = SP	RB < SP
<i>15-Puzzle</i>	32,51 %	67,49 %	0,0 %
<i>24-Puzzle</i>	10,34 %	89,66 %	0,0 %

Cuadro 10: Regiones de blancos vs. sólo patrón

Como se puede observar en el Cuadro 10, la heurística de las regiones de blancos domina para la totalidad de los casos de prueba, lo que concuerda con el hecho de que los elementos

tomados en cuenta sin la casilla vacía comprenden un subconjunto de los utilizados por el método de las regiones de blancos. Luego, para cualquier estado del *N-Puzzle*, la heurística obtenida a partir de las regiones de blancos siempre será mayor o igual a la conseguida sin utilizar el blanco.

	Diferencia RB – SP				
	2	4	6	8	10
<i>15-Puzzle</i>	2.896.905	338.947	14.955	269	5
<i>24-Puzzle</i>	986.356	46.818	1.267	23	-

Cuadro 11: Diferencias de valores heurísticos entre RB y SP

	# Regiones							
	1	2	3	4	5	6	7	8
<i>15-Puzzle PDB-7</i>	17,76 %	37,52 %	31,99 %	10,14 %	2,34 %	0,21 %	0,03 %	-
<i>15-Puzzle PDB-8</i>	11,45 %	31,24 %	34,99 %	16,92 %	4,42 %	0,90 %	0,06 %	0,02 %
<i>24-Puzzle PDB-6</i>	62,79 %	32,72 %	4,28 %	0,22 %	0,01 %	-	-	-

Cuadro 12: Porcentaje de patrones según el número de regiones de blancos

Otro resultado interesante de analizar es la diferencia que existe entre los porcentajes donde el método de las regiones de blancos es superior para el *15-Puzzle* (32,51 %) y *24-Puzzle* (10,34 %). La disparidad entre estos valores, como se muestra en el Cuadro 12, es una consecuencia de que para la mayoría de los patrones del *24-Puzzle* sólo existe una región de blancos, lo que implica que para ninguno de ellos las regiones de blancos ofrecerá una mejora sobre los valores obtenidos sin tomar en cuenta el blanco, mientras que para ambas *PDB* del *15-Puzzle* más del 80 % de los patrones poseen al menos dos regiones de blancos.

En los gráficos 21 y 22 del Apéndice A se puede observar la distribución de los patrones según su valor heurístico para las bases de datos de patrones sin la casilla vacía y con regiones de blancos, para los casos del *15-Puzzle* y *24-Puzzle* respectivamente. Para el *15-Puzzle* se evidencia una diferencia mayor entre ambas bases de datos en comparación con el *24-Puzzle*, lo que corrobora los resultados presentados en el Cuadro 11. Sin embargo, ambas gráficas muestran que para las *PDB* con regiones de blancos, el porcentaje de patrones en los niveles

superiores es mayor en comparación con el otro método.

4.3. Comparación de Algoritmos de Enumeración

Al trabajar con bases de datos de patrones, uno de los elementos determinantes para el desempeño del proceso de búsqueda son los algoritmos de enumeración o indexación escogidos. Debido al gran número de consultas que se realizan sobre las *PDB*, la correspondencia entre valores numéricos y estados del problema es una de las operaciones realizadas con mayor frecuencia durante la búsqueda, por lo que es necesario que este proceso se efectúe de la manera más rápida y eficiente posible.

Por esta razón, se decidió comparar el rendimiento de los tres algoritmos para enumerar patrones propuestos en la Sección 3.2. Para realizar este análisis se generaron diez millones de instancias aleatorias del *24-Puzzle* y se midió el tiempo que cada algoritmo necesitó para enumerar cada uno de los patrones presentes en ellas.

	No Lexicográfico	Lexicográfico	Regiones de Blanco
Tiempo	8m17.363s	6m59.646s	7m11.495s

Cuadro 13: Tiempo de enumeración de 10 millones de instancias del *24-Puzzle*

Como se observa en el Cuadro 13, a pesar de que el algoritmo de enumeración lexicográfica realiza la tarea en un menor tiempo que los demás, la diferencia en desempeño no es mayor al 15% en comparación con la enumeración lexicográfica, la cual resultó en el mayor tiempo de ejecución. Esto concuerda con el hecho de que todos los algoritmos que se evaluaron en este experimento son de orden lineal.

4.4. Resolución de Instancias del *N-Puzzle*

Luego de haber realizado un estudio exhaustivo sobre la teoría detrás de las bases de datos de patrones y acerca de los diferentes algoritmos para la indexación de las mismas, así como también haber llevado a cabo un análisis experimental de ambos en las secciones previas, a continuación se presentan un conjunto de resultados que representan la respuesta al

objetivo principal establecido para esta investigación, es decir, encontrar soluciones óptimas para instancias del *N-Puzzle* utilizando bases de datos de patrones.

Las instancias utilizadas para los siguientes experimentos fueron tomadas de [Korf, 1985] para el *15-Puzzle* y [Korf y Felner, 2002] para el *24-Puzzle*, y se encuentran detalladas en los Cuadros 19 y 20 del Apéndice B.

Como se mencionó en la Sección 3.4, para la resolución de instancias del *N-Puzzle* se utilizó el algoritmo *IDA** implementado según los lineamientos establecidos en el Capítulo 2 y, aprovechando algunas características particulares del objeto de estudio, se implementaron ciertas funciones con el fin de mejorar el desempeño del mismo. Para corroborar esto, se llevó a cabo una prueba utilizando las diez instancias del *15-Puzzle* con mayor número de nodos generados por solución, mientras que para el *24-Puzzle* se tomaron las cinco de menor complejidad.

Funciones	<i>15-Puzzle</i>			<i>24-Puzzle</i>		
	H_0	Nodos	Tiempo	H_0	Nodos	Tiempo
Ninguna	49,6	55.104.164,2	7,242s	-	-	-
AP	49,6	709.875,7	0,131s	73,8	1.876.094.251,3	10m7,398s
AP + CSI	49,6	709.875,7	0,121s	73,8	1.876.094.251,4	4m46,813s
AP + CSI + TR	51,2	213.239,4	0,071s	75,4	342.940.721,1	1m38,390s

Cuadro 14: Desempeño de *Iterative Deepening A** según funciones agregadas

Como se puede observar en el Cuadro 14, la adición de cada una de estas funciones resulta en una mejora sustancial en el desempeño del algoritmo. Particularmente relevantes son la función de eliminación de la acción inversa que produjo a un nodo (*AP*), ya que reduce significativamente el número de nodos generados gracias a la eliminación de una gran cantidad de transposiciones, y la función de creación y actualización del tablero reflejo (*TR*), dado que resulta en un incremento notable en los valores heurísticos asociados a los distintos estados del problema. La función de cálculo de un solo índice (*CSI*) produce una mejora en el desempeño especialmente para el *24-Puzzle*, ya que mediante su uso se logra disminuir de 8 a 2 la cantidad de índices recalculados para cada nodo generado. Esto se debe a que por

cada acción realizada se modifica un patrón del tablero normal y otro del tablero reflejo.

Una vez comprobada la utilidad de cada una de estas funciones agregadas a *IDA**, se procedió a buscar las soluciones óptimas de 100 instancias del *15-Puzzle* y de 20 del *24-Puzzle*. Los resultados promedio de estos experimentos se pueden observar en los Cuadros 15 y 16, para el *15-Puzzle* y *24-Puzzle* respectivamente. En el Apéndice C se listan los resultados individuales para cada instancia evaluada.

Enumeración	H ₀	Nodos	Tiempo	Nodos/s
No Lexicográfica	45,59	41.042,36	0,02304144	1.781.241,10
Lexicográfica	45,59	41.042,36	0,01376085	2.982.545,41
Regiones de Blancos	46,43	22.347,84	0,01400088	1.596.173,95

Cuadro 15: Resultados promedio para las 100 instancias del *15-Puzzle*

Enumeración	H ₀	Nodos	Tiempo	Nodos/s
No Lexicográfica	78,95	6.501.022.874,05	49m59,884s	2.167.090,93
Lexicográfica	78,95	6.501.022.874,05	30m58,007s	3.498.922,18
Regiones de Blancos	79,05	3.916.378.084,55	37m21,772s	1.747.000,71

Cuadro 16: Resultados promedio para las 20 instancias del *24-Puzzle*

La primera observación que se deriva de analizar los resultados presentados en estos cuadros, es que mediante una enumeración lexicográfica se logra resolver en menor tiempo las instancias del *N-Puzzle*, mientras que indexar por regiones de blancos implica generar una menor cantidad de nodos durante la búsqueda.

Esta mejora conseguida para el desempeño del algoritmo *IDA**, producto de la enumeración lexicográfica de patrones, se puede comprender por medio de los siguientes argumentos:

- Como se probó en la Sección 4.3, a pesar de que la diferencia en el desempeño de los algoritmos no resultó ser significativa, la enumeración lexicográfica fue la que obtuvo los mejores resultados. Pero tomando en cuenta que la mayor diferencia en cuanto a tiempos de enumeración fue de un 15 %, esta disparidad por sí sola no resulta suficiente para explicar la mejora en rendimiento de *IDA** al utilizar este tipo de enumeración.

- Como se discutió en la Sección 2.5.3, los accesos consecutivos a una *PDB* generalmente son producto de un conjunto de patrones sucesivos. Por lo tanto, debido al uso de una enumeración lexicográfica, los índices correspondientes a dichos patrones tenderán a estar relativamente cerca, y esto traerá como beneficio que los accesos a memoria gocen de localidad espacial, lo que se traduce en una mejora substancial del desempeño del algoritmo *IDA** [Korf y Schultze, 2005].

Es importante señalar que la información almacenada en las bases de datos de patrones asociadas a la enumeración lexicográfica y no lexicográfica es exactamente la misma, pero está distribuida de una manera distinta internamente según el orden que establece cada algoritmo de enumeración. Esta es la razón por la que el número de nodos generados en los resultados es igual para ambos casos.

Tomando en cuenta el hecho de que la enumeración por regiones de blancos no realiza la correspondencia entre patrones y valores numéricos de manera lexicográfica, resulta correcto establecer comparaciones entre los resultados obtenidos con las bases de datos de patrones generadas por este método y las producidas por el algoritmo no lexicográfico, si lo que se desea es evaluar la influencia de la calidad de la información almacenada en ellas sobre el desempeño de la búsqueda.

De esta manera, analizando los valores de los cuadros antes descritos para estas *PDB*, se puede observar una reducción de aproximadamente 40% en el número de nodos generados y más de 25% para los tiempos totales de búsqueda a favor del método de regiones de blancos. Es importante destacar que la diferencia entre estos porcentajes proviene del hecho de que el nuevo método propuesto incurre en el costo adicional de manejar en memoria tanto bases de datos de patrones más grandes como una gran cantidad de valores precalculados necesarios para la función de enumeración, lo que se refleja principalmente en la disminución de la tasa de generación de nodos.

Otro resultado interesante del uso del método de regiones de blancos es el aumento que tiene su valor heurístico inicial en comparación con el de las otras bases de datos de patrones, particularmente para el *15-Puzzle*. Este incremento es una consecuencia de almacenar los

valores heurísticos tomando en cuenta la ubicación de la casilla vacía, es decir, sin pérdida de información. En el caso del *24-Puzzle*, la diferencia no es tan notoria debido a que para la mayoría de los posibles patrones (62,79%), el número de regiones de blancos es tan solo una, lo que implica que el método propuesto no ofrece ninguna ventaja sobre los valores producidos por las otras *PDB* en estos casos. Sería necesario trabajar con patrones de mayor cantidad de elementos para lograr aumentar el número de regiones promedio por patrón, pero esto sería impráctico debido al tamaño de las *PDB* que se generarían.

Finalmente, a pesar de que la enumeración lexicográfica obtiene los mejores tiempos de búsqueda, el método de compresión por regiones de blanco comprende una alternativa que tiene la ventaja de reducir la cantidad de nodos generados, producto de mejores valores heurísticos asociados a los distintos patrones del espacio de búsqueda.

Capítulo 5

Conclusiones y Recomendaciones

Como resultado de esta investigación, se logró resolver instancias del *N-Puzzle*, en sus versiones del *15-Puzzle* y el *24-Puzzle*, utilizando bases de datos de patrones, para las que se consideraron diferentes tipos de indexación y compresión, con la finalidad de obtener el mejor rendimiento posible.

La construcción de las *PDB* partió del planteamiento realizado en [Korf y Felner, 2002], en donde se comprimen las bases de datos de patrones al no tomar en cuenta la posición de la casilla vacía al momento de almacenar el valor asociado a un determinado patrón. El método propuesto en este trabajo ofrece una alternativa para comprimir las *PDB* sin sacrificar información relevante para la solución del problema. A esta estrategia se le denominó compresión por regiones de blancos, y con ella se logró construir una heurística admisible que domina a la generada a partir de la *PDB* que no considera el blanco, obteniéndose una reducción en la cantidad de nodos generados para alcanzar la solución de un 45,5% y 39,7% para el *15-Puzzle* y el *24-Puzzle* respectivamente.

Por otro lado, se logró comprobar que un factor que resulta ser determinante en el desempeño general de la búsqueda, son los algoritmos de enumeración utilizados para consultar las *PDB* y obtener el valor heurístico correspondiente a cada estado.

Con la finalidad de estudiar el impacto que puede tener una determinada estrategia de indexación, se compararon dos *PDB* que contenían la misma información, pero organizada internamente diferente. La disposición interna de una de ellas consideraba un orden lexicográfico de los estados del problema, mientras que la otra no guardaba un orden particular.

Los resultados obtenidos mostraron un desempeño superior del algoritmo de búsqueda al utilizar una enumeración lexicográfica. La ventaja que ofrece este tipo de indexación es que la diferencia entre los índices correspondientes a estados que son adyacentes en el árbol de la búsqueda es mucho menor, ya que ellos difieren en muy pocos elementos de su

configuración, por lo tanto los valores heurísticos asociados a cada uno de ellos van a tener una alta probabilidad de estar precargados en memoria caché al mismo tiempo, disminuyendo el costo de consultar en la *PDB* y permitiendo generar una mayor cantidad de nodos por segundo.

Asimismo, a pesar de que el método de compresión por regiones de blancos resultó en los mejores valores heurísticos y por ende la menor cantidad de nodos generados, debido al costo agregado de mantener y manejar una mayor cantidad de información en memoria (bases de datos más grandes y arreglos precalculados), la tasa de generación de nodos no resultó ser lo suficientemente elevada para superar el desempeño del algoritmo lexicográfico. Por lo tanto, se determinó que la indexación lexicográfica resulta en los mejores tiempos de búsqueda debido a que ésta realiza un manejo más efectivo de la memoria.

Todos estos argumentos evidencian la gran cantidad de factores que se deben tomar en cuenta a la hora de realizar búsqueda heurística con bases de datos de patrones, para lo que resulta necesario aprovechar la mayor cantidad posible de recursos disponibles si se desea tener la capacidad de resolver problemas como el *24-Puzzle*, donde el espacio de búsqueda es considerablemente grande.

Finalmente, las futuras investigaciones que sigan desarrollando las ideas propuestas en este trabajo deben estar dirigidas a extender o modificar el método de enumeración por regiones de blancos para acomodar un ordenamiento lexicográfico de los patrones. Con esto se lograría combinar el uso eficiente de la memoria caché que se obtiene a partir de este ordenamiento, con los valores heurísticos dominantes que ofrece la estrategia de las regiones de blancos, lo que finalmente se traduciría en una mayor tasa de generación de nodos y una menor cantidad de ellos generados.

Bibliografía

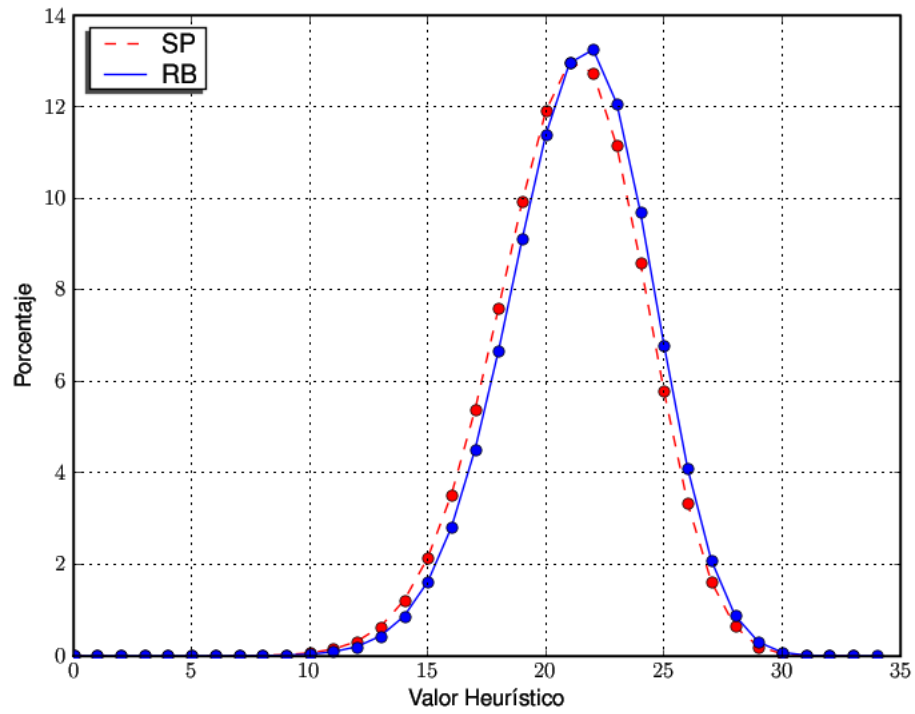
- [Cormen *et al.*, 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition.
- [Culberson y Schaeffer, 1996] Culberson, J. C. y Schaeffer, J. (1996). Searching with pattern databases. In *Canadian Conference on AI*, pages 402–416. Disponible en: citeseer.ist.psu.edu/culberson96searching.html.
- [Felner y Adler, 2005] Felner, A. y Adler, A. (2005). Solving the 24 puzzle with instance dependent pattern databases. In *SARA*, pages 248–260.
- [Felner *et al.*, 2004a] Felner, A., Korf, R. E., y Hanan, S. (2004a). Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)*, 22:279–318. Disponible en: <http://dblp.uni-trier.de/db/journals/jair/jair22.html#FelnerKH04>.
- [Felner *et al.*, 2004b] Felner, A., Meshulam, R., Holte, R. C., y Korf, R. E. (2004b). Compressing pattern databases. In *AAAI*, pages 638–643.
- [Knuth, 2005] Knuth, D. E. (2005). *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions*. Addison-Wesley Professional.
- [Korf, 1985] Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artif. Intell.*, 27(1):97–109.
- [Korf y Felner, 2002] Korf, R. E. y Felner, A. (2002). Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22.
- [Korf y Schultze, 2005] Korf, R. E. y Schultze, P. (2005). Large-scale parallel breadth-first search. In Veloso, M. M. y Kambhampati, S., editors, *AAAI*, pages 1380–1385. AAAI Press / The MIT Press. Disponible en: <http://dblp.uni-trier.de/db/conf/aaai/aaai2005.html#Korfs05>.
- [Meza y Ortega, 1993] Meza, O. y Ortega, M. (1993). *Grafos y Algoritmos*. Equinoccio, 1st edition edition.
- [Myrvold y Ruskey, 2001] Myrvold, W. y Ruskey, F. (2001). Ranking and unranking permutations in linear time. *Inf. Process. Lett.*, 79(6):281–284.
- [Nieto, 2005] Nieto, J. H. (2005). Permutaciones y el juego del 15. *Boletín de la Asociación Matemática Venezolana*, 12:259–264. Disponible en: <http://www.emis.de/journals/BAMV/conten/vol12/jnieto.pdf>.
- [Russell y Norvig, 2003] Russell, S. y Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

Apéndice A

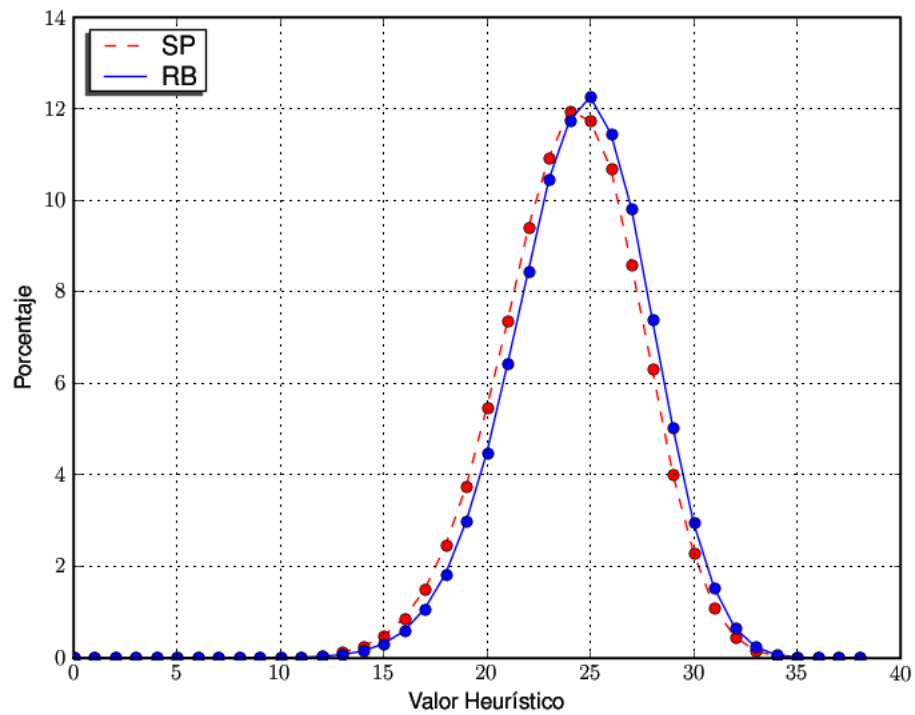
Distribución de Patrones

Valor heurístico	PDB-7		PDB-8	
	SP	RB	SP	RB
0	1	1	1	1
1	2	2	4	4
2	6	6	10	10
3	24	24	44	44
4	95	100	155	161
5	311	335	498	564
6	904	1002	1454	1626
7	2392	2756	3856	4662
8	6056	7392	9753	11808
9	15091	19546	23740	31058
10	36696	50316	57104	76094
11	84265	123044	131720	189664
12	181968	282201	294927	438515
13	368821	605793	624860	1004446
14	699428	1214114	1278710	2126668
15	1236038	2265795	2456938	4387212
16	2030914	3927282	4529270	8360857
17	3100164	6302160	7768314	15369962
18	4385247	9340077	12755050	26110622
19	5727394	12748130	19446958	42509752
20	6865358	15938114	28359574	63900663
21	7483204	18111087	38145480	91480130
22	7345188	18523648	48739135	120278860
23	6424533	16879874	56731544	148898294
24	4955682	13560935	62044160	167327095
25	3330386	9500728	60881276	174646354
26	1922268	5726976	55472562	163267401
27	935172	2923853	44521972	139814942
28	371883	1240359	32781777	105465585
29	116971	422006	20794252	71738196
30	26821	110309	11928080	42062209
31	4025	19731	5692908	21758720
32	285	2126	2396445	9349503
33	7	96	783878	3399106
34	-	2	216751	948664
35	-	-	39436	203040
36	-	-	5465	26569
37	-	-	322	1930
38	-	-	17	49
Total de Patrones	57657600	139849920	518918400	1425191040

Cuadro 17: Número de patrones según su valor heurístico para el 15-Puzzle



(a) *PDB-7*

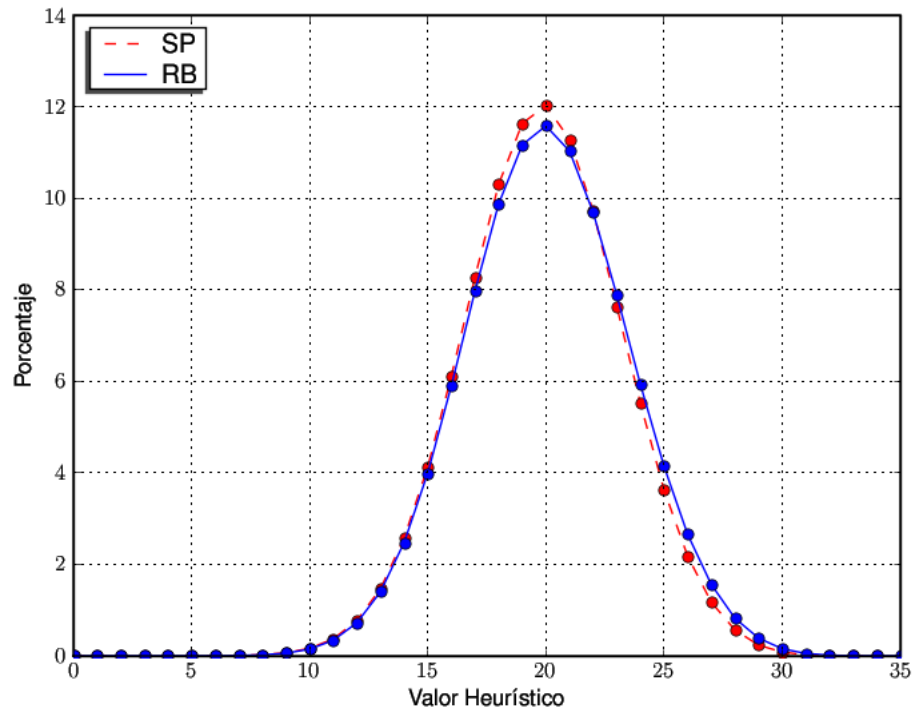


(b) *PDB-8*

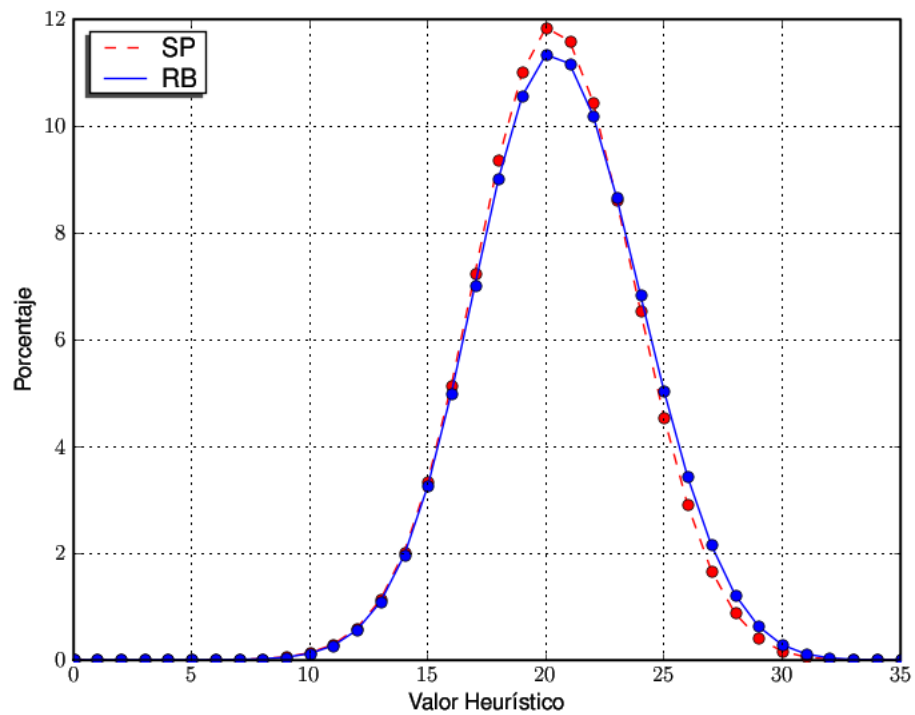
Figura 21: Distribución de patrones según su valor heurístico para el *15-Puzzle*

Valor heurístico	PDB-6 Irregular		PDB-6 Regular	
	SP	RB	SP	RB
0	1	1	1	1
1	5	5	2	2
2	22	22	9	9
3	95	103	54	55
4	370	397	273	280
5	1232	1385	1074	1155
6	3791	4320	3757	4129
7	10563	12651	11553	13356
8	28437	34384	32980	39117
9	71058	88761	85905	106101
10	168426	213419	210543	265914
11	367936	482235	472673	616664
12	756771	1009584	985781	1308777
13	1438714	1964433	1871683	2535817
14	2573547	3538326	3284912	4482475
15	4254808	5890359	5262740	7229511
16	6558453	9042318	7799786	10683502
17	9236626	12705389	10556162	14433632
18	11950095	16327060	13134678	17871658
19	14030534	19118948	14834746	20210658
20	15094797	20512342	15330827	20960619
21	14783693	20216127	14391820	19963596
22	13318514	18447110	12400468	17541704
23	10973204	15659774	9738365	14269210
24	8341436	12377348	7035973	10739111
25	5792559	9116498	4622876	7498469
26	3699450	6200945	2779377	4805688
27	2133436	3895004	1496425	2823636
28	1115979	2216440	723202	1488576
29	511246	1135170	300217	703149
30	206439	508256	106335	282320
31	67647	202524	29673	96111
32	18264	65498	6250	26648
33	3438	16890	832	5734
34	408	3350	48	592
35	6	624	-	24
Total de Patrones	127512000	181008000	127512000	181008000

Cuadro 18: Número de patrones según su valor heurístico para el *24-Puzzle*



(a) *PDB-6* irregular



(b) *PDB-6* regular

Figura 22: Distribución de patrones según su valor heurístico para el *24-Puzzle*

Apéndice B

Casos de Prueba

Nº Caso	Configuración del tablero	Prof. Solución
1	7 15 8 2 13 6 3 12 11 0 4 10 9 5 1 14	53
2	14 13 15 7 11 12 9 5 6 0 2 1 4 8 10 3	57
3	13 5 4 10 9 12 8 14 2 3 7 1 0 15 11 6	55
4	14 7 8 2 13 11 10 4 9 12 5 0 3 6 1 15	59
5	5 12 10 7 15 11 14 0 8 2 1 13 3 4 9 6	56
6	4 7 14 13 10 3 9 12 11 5 6 15 1 2 8 0	56
7	14 7 1 9 12 3 6 15 8 11 2 5 10 0 4 13	52
8	2 11 15 5 13 4 6 7 12 8 10 1 9 3 14 0	52
9	12 11 15 3 8 0 4 2 6 13 9 5 14 1 10 7	50
10	3 14 9 11 5 4 8 2 13 12 6 7 10 1 15 0	46
11	13 11 8 9 0 15 7 10 4 3 6 14 5 12 2 1	59
12	5 9 13 14 6 3 7 12 10 8 4 0 15 2 11 1	57
13	14 1 9 6 4 8 12 5 7 2 3 0 10 11 13 15	45
14	3 6 5 2 10 0 15 14 1 4 13 12 9 8 11 7	46
15	7 6 8 1 11 5 14 10 3 4 9 13 15 2 0 12	59
16	13 11 4 12 1 8 9 15 6 5 14 2 7 3 10 0	62
17	1 3 2 5 10 9 15 6 8 14 13 11 12 4 7 0	42
18	15 14 0 4 11 1 6 13 7 5 8 9 3 2 10 12	66
19	6 0 14 12 1 15 9 10 11 4 7 2 8 3 5 13	55
20	7 11 8 3 14 0 6 15 1 4 13 9 5 12 2 10	46
21	6 12 11 3 13 7 9 15 2 14 8 10 4 1 5 0	52
22	12 8 14 6 11 4 7 0 5 1 10 15 3 13 9 2	54
23	14 3 9 1 15 8 4 5 11 7 10 13 0 2 12 6	59
24	10 9 3 11 0 13 2 14 5 6 4 7 8 15 1 12	49
25	7 3 14 13 4 1 10 8 5 12 9 11 2 15 6 0	54
26	11 4 2 7 1 0 10 15 6 9 14 8 3 13 5 12	52
27	5 7 3 12 15 13 14 8 0 10 9 6 1 4 2 11	58
28	14 1 8 15 2 6 0 3 9 12 10 13 4 7 5 11	53
29	13 14 6 12 4 5 1 0 9 3 10 2 15 11 8 7	52
30	9 8 0 2 15 1 4 14 3 10 7 5 11 13 6 12	54
31	12 15 2 6 1 14 4 8 5 3 7 0 10 13 9 11	47
32	12 8 15 13 1 0 5 4 6 3 2 11 9 7 14 10	50
33	14 10 9 4 13 6 5 8 2 12 7 0 1 3 11 15	59
34	14 3 5 15 11 6 13 9 0 10 2 12 4 1 7 8	60
35	6 11 7 8 13 2 5 4 1 10 3 9 14 0 12 15	52
36	1 6 12 14 3 2 15 8 4 5 13 9 0 7 11 10	55
37	12 6 0 4 7 3 15 1 13 9 8 11 2 14 5 10	52
38	8 1 7 12 11 0 10 5 9 15 6 13 14 2 3 4	58
39	9 0 4 10 1 14 15 3 12 6 5 7 11 13 8 2	49
40	11 5 1 14 4 12 10 0 2 7 13 3 9 15 6 8	54
41	8 13 10 9 11 3 15 6 0 1 2 14 12 5 4 7	54
42	4 5 7 2 9 14 12 13 0 3 6 11 8 1 15 10	42

Nº Caso	Configuración del tablero	Prof. Solución
43	11 15 14 13 1 9 10 4 3 6 2 12 7 5 8 0	64
44	12 9 0 6 8 3 5 14 2 4 11 7 10 1 15 13	50
45	3 14 9 7 12 15 0 4 1 8 5 6 11 10 2 13	51
46	8 4 6 1 14 12 2 15 13 10 9 5 3 7 0 11	49
47	6 10 1 14 15 8 3 5 13 0 2 7 4 9 11 12	47
48	8 11 4 6 7 3 10 9 2 12 15 13 0 1 5 14	49
49	10 0 2 4 5 1 6 12 11 13 9 7 15 3 14 8	59
50	12 5 13 11 2 10 0 9 7 8 4 3 14 6 15 1	53
51	10 2 8 4 15 0 1 14 11 13 3 6 9 7 5 12	56
52	10 8 0 12 3 7 6 2 1 14 4 11 15 13 9 5	56
53	14 9 12 13 15 4 8 10 0 2 1 7 3 11 5 6	64
54	12 11 0 8 10 2 13 15 5 4 7 3 6 9 14 1	56
55	13 8 14 3 9 1 0 7 15 5 4 10 12 2 6 11	41
56	3 15 2 5 11 6 4 7 12 9 1 0 13 14 10 8	55
57	5 11 6 9 4 13 12 0 8 2 15 10 1 7 3 14	50
58	5 0 15 8 4 6 1 14 10 11 3 9 7 12 2 13	51
59	15 14 6 7 10 1 0 11 12 8 4 9 2 5 13 3	57
60	11 14 13 1 2 3 12 4 15 7 9 5 10 6 8 0	66
61	6 13 3 2 11 9 5 10 1 7 12 14 8 4 0 15	45
62	4 6 12 0 14 2 9 13 11 8 3 15 7 10 1 5	57
63	8 10 9 11 14 1 7 15 13 4 0 12 6 2 5 3	56
64	5 2 14 0 7 8 6 3 11 12 13 15 4 10 9 1	51
65	7 8 3 2 10 12 4 6 11 13 5 15 0 1 9 14	47
66	11 6 14 12 3 5 1 15 8 0 10 13 9 7 4 2	61
67	7 1 2 4 8 3 6 11 10 15 0 5 14 12 13 9	50
68	7 3 1 13 12 10 5 2 8 0 6 11 14 15 4 9	51
69	6 0 5 15 1 14 4 9 2 13 8 10 11 12 7 3	53
70	15 1 3 12 4 0 6 5 2 8 14 9 13 10 7 11	52
71	5 7 0 11 12 1 9 10 15 6 2 3 8 4 13 14	44
72	12 15 11 10 4 5 14 0 13 7 1 2 9 8 3 6	56
73	6 14 10 5 15 8 7 1 3 4 2 0 12 9 11 13	49
74	14 13 4 11 15 8 6 9 0 7 3 1 2 10 12 5	56
75	14 4 0 10 6 5 1 3 9 2 13 15 12 7 8 11	48
76	15 10 8 3 0 6 9 5 1 14 13 11 7 2 12 4	57
77	0 13 2 4 12 14 6 9 15 1 10 3 11 5 8 7	54
78	3 14 13 6 4 15 8 9 5 12 10 0 2 7 1 11	53
79	0 1 9 7 11 13 5 3 14 12 4 2 8 6 10 15	42
80	11 0 15 8 13 12 3 5 10 1 4 6 14 9 7 2	57
81	13 0 9 12 11 6 3 5 15 8 1 10 4 14 2 7	53
82	14 10 2 1 13 9 8 11 7 3 6 12 15 5 4 0	62
83	12 3 9 1 4 5 10 2 6 11 15 0 14 7 13 8	49
84	15 8 10 7 0 12 14 1 5 9 6 3 13 11 4 2	55
85	4 7 13 10 1 2 9 6 12 8 14 5 3 0 11 15	44
86	6 0 5 10 11 12 9 2 1 7 4 3 14 8 13 15	45
87	9 5 11 10 13 0 2 1 8 6 14 12 4 7 3 15	52
88	15 2 12 11 14 13 9 5 1 3 8 7 0 10 6 4	65
89	11 1 7 4 10 13 3 8 9 14 0 15 6 5 2 12	54
90	5 4 7 1 11 12 14 15 10 13 8 6 2 0 9 3	50
91	9 7 5 2 14 15 12 10 11 3 6 1 8 13 0 4	57

Nº Caso	Configuración del tablero	Prof. Solución
92	3 2 7 9 0 15 12 4 6 11 5 14 8 13 10 1	57
93	13 9 14 6 12 8 1 2 3 4 0 7 5 10 11 15	46
94	5 7 11 8 0 14 9 13 10 12 3 15 6 1 4 2	53
95	4 3 6 13 7 15 9 0 10 5 8 11 2 12 1 14	50
96	1 7 15 14 2 6 4 9 12 11 13 3 0 8 5 10	49
97	9 14 5 7 8 15 1 2 10 4 13 6 12 0 11 3	44
98	0 11 3 12 5 2 1 9 8 10 14 15 7 4 13 6	54
99	7 15 4 0 10 9 2 5 12 11 13 6 1 3 14 8	57
100	11 4 0 8 6 10 5 13 12 7 14 3 1 2 9 1	54

Cuadro 19: 100 instancias del *15-Puzzle*

Nº Caso	Configuración del tablero	Prof. Solución
1	10 3 24 12 0 7 8 11 14 21 22 23 2 1 9 17 18 6 20 4 13 15 5 19 16	96
2	2 17 4 13 7 12 10 3 0 16 21 24 8 5 18 20 15 19 14 9 22 11 6 1 23	82
3	3 17 9 8 24 1 11 12 14 0 5 4 22 13 16 21 15 6 7 10 20 23 2 18 19	81
4	1 12 18 13 17 15 3 7 20 0 19 24 6 5 21 11 2 8 9 16 22 10 4 23 14	97
5	8 12 18 3 2 11 10 22 24 17 1 13 23 4 20 16 6 15 9 21 19 5 14 0 7	93
6	23 22 5 3 9 6 18 15 10 2 21 13 19 12 20 7 0 1 16 24 17 4 14 8 11	100
7	8 19 7 16 12 2 13 22 14 9 11 5 6 3 18 24 0 15 10 23 1 20 4 17 21	92
8	21 24 8 1 19 22 12 9 7 18 4 0 23 14 10 6 3 11 16 5 15 2 20 13 17	101
9	14 5 9 2 18 8 23 19 12 17 15 0 10 20 4 6 11 21 1 7 24 3 16 22 13	95
10	17 15 7 12 8 3 4 9 21 5 16 6 19 20 1 22 24 18 11 14 23 10 2 13 0	98
11	2 10 1 7 16 9 0 6 12 11 3 18 22 4 13 24 20 15 8 14 21 23 17 19 5	90
12	17 1 20 9 16 2 22 19 14 5 15 21 0 3 24 23 18 13 12 7 10 8 6 4 11	100
13	7 6 3 22 15 19 21 2 13 0 8 10 9 4 18 16 11 24 5 12 17 1 23 14 20	95
14	18 24 17 11 12 10 19 15 6 1 5 21 22 9 7 3 2 16 14 4 20 23 0 8 13	96
15	18 14 0 9 8 3 7 19 2 15 5 12 1 13 24 23 4 21 10 20 16 22 11 6 17	98
16	22 21 15 3 14 13 9 19 24 23 16 0 7 10 18 4 11 20 8 2 1 6 5 17 12	105
17	6 0 24 14 8 5 21 19 9 17 16 20 10 13 2 15 11 22 1 3 7 23 4 18 12	97
18	19 20 12 21 7 0 16 10 5 9 14 23 3 11 4 2 6 1 8 15 17 13 22 24 18	99
19	13 19 9 10 14 15 23 21 24 16 12 11 0 5 22 20 4 18 3 1 6 2 7 17 8	106
20	21 11 10 4 16 6 13 24 7 14 1 20 9 17 0 15 2 5 8 22 3 12 18 19 23	92

Cuadro 20: 20 instancias del *24-Puzzle*

Apéndice C

Resultados Individuales

Caso		Sólo patrón				Regiones de blancos		
No.	Sol.	H ₀	Nodos	Tiempo-NL	Tiempo-L	H ₀	Nodos	Tiempo
1	53	47	4363	0.004s	0.004s	47	2739	0.004s
2	57	51	5515	0.0s	0.0s	51	3674	0.004s
3	55	47	3363	0.004s	0.004s	49	1984	0.0s
4	59	47	178885	0.100s	0.060s	49	101858	0.064s
5	56	50	3079	0.004s	0.0s	50	1539	0.0s
6	56	46	31659	0.016s	0.012s	46	19835	0.012s
7	52	48	9342	0.008s	0.004s	48	3441	0.004s
8	52	46	31879	0.016s	0.012s	46	18404	0.012s
9	50	40	27423	0.016s	0.008s	42	16941	0.008s
10	46	40	740	0.0s	0.0s	44	475	0.004s
11	59	51	97633	0.056s	0.036s	51	49137	0.028s
12	57	49	93990	0.052s	0.028s	49	55287	0.036s
13	45	41	1548	0.0s	0.004s	41	583	0.0s
14	46	40	2797	0.004s	0.0s	42	1417	0.0s
15	59	51	11185	0.004s	0.004s	51	7162	0.008s
16	62	52	200457	0.108s	0.064s	52	110509	0.064s
17	42	34	23006	0.012s	0.008s	34	10146	0.004s
18	66	56	18502	0.008s	0.008s	56	11433	0.008s
19	55	51	9878	0.008s	0.004s	51	6389	0.004s
20	46	44	336	0.0s	0.0s	44	297	0.0s
21	52	42	12193	0.008s	0.004s	44	6372	0.004s
22	54	46	75620	0.040s	0.024s	46	33690	0.024s
23	59	49	180667	0.100s	0.060s	49	102241	0.064s
24	49	41	4577	0.004s	0.004s	43	3011	0.0s
25	54	46	65270	0.036s	0.020s	46	25550	0.016s
26	52	44	7835	0.004s	0.004s	46	5188	0.004s
27	58	48	84262	0.048s	0.028s	48	48171	0.028s
28	53	43	67699	0.040s	0.024s	45	30921	0.020s
29	52	48	2162	0.0s	0.0s	48	1366	0.004s
30	54	48	7045	0.004s	0.004s	48	5089	0.0s
31	47	43	797	0.0s	0.0s	45	252	0.0s
32	50	48	368	0.0s	0.0s	48	254	0.004s
33	59	49	219458	0.124s	0.072s	51	132518	0.080s
34	60	52	32235	0.020s	0.012s	52	20039	0.012s
35	52	44	21119	0.012s	0.008s	44	13047	0.008s
36	55	47	7153	0.004s	0.004s	47	4640	0.004s
37	52	44	23217	0.012s	0.008s	44	8476	0.008s
38	58	50	19004	0.012s	0.008s	50	11992	0.008s
39	49	41	17803	0.012s	0.004s	41	9289	0.004s
40	54	44	33383	0.016s	0.012s	46	11390	0.008s
41	54	46	48627	0.028s	0.016s	46	23079	0.016s

Caso		Sólo patrón				Regiones de blancos		
No.	Sol.	H ₀	Nodos	Tiempo-NL	Tiempo-L	H ₀	Nodos	Tiempo
42	42	36	582	0.0s	0.0s	38	383	0.0s
43	64	56	35502	0.020s	0.012s	56	23093	0.016s
44	50	42	10856	0.008s	0.0s	44	3853	0.004s
45	51	47	4370	0.004s	0.004s	47	2811	0.0s
46	49	43	5300	0.0s	0.0s	43	3312	0.004s
47	47	43	458	0.004s	0.0s	45	250	0.0s
48	49	43	2300	0.0s	0.0s	45	1415	0.0s
49	59	49	32309	0.020s	0.012s	49	17877	0.012s
50	53	47	30167	0.016s	0.008s	47	17294	0.012s
51	56	48	10921	0.008s	0.004s	50	6619	0.004s
52	56	48	35653	0.020s	0.012s	50	12327	0.008s
53	64	58	42654	0.024s	0.012s	58	26891	0.016s
54	56	48	54900	0.032s	0.020s	48	23862	0.016s
55	41	35	2011	0.0s	0.0s	37	1702	0.004s
56	55	43	76437	0.040s	0.028s	47	42696	0.024s
57	50	46	1556	0.004s	0.0s	46	983	0.0s
58	51	45	4219	0.0s	0.0s	45	2567	0.004s
59	57	49	53842	0.032s	0.020s	51	24538	0.016s
60	66	54	383508	0.212s	0.128s	56	236073	0.144s
61	45	37	5750	0.004s	0.0s	37	3812	0.004s
62	57	49	17230	0.012s	0.008s	51	6564	0.004s
63	56	48	117159	0.064s	0.040s	48	72981	0.048s
64	51	41	47864	0.028s	0.016s	43	21912	0.012s
65	47	41	6115	0.004s	0.004s	41	3639	0.004s
66	61	53	85060	0.048s	0.028s	55	48210	0.032s
67	50	40	51543	0.028s	0.016s	40	27940	0.016s
68	51	43	12518	0.008s	0.004s	43	6127	0.004s
69	53	45	22656	0.012s	0.004s	45	12738	0.008s
70	52	42	29017	0.016s	0.012s	44	13003	0.008s
71	44	40	1060	0.0s	0.0s	40	999	0.0s
72	56	50	5788	0.004s	0.004s	50	4603	0.004s
73	49	43	2464	0.004s	0.0s	43	1065	0.0s
74	56	52	1482	0.0s	0.0s	52	876	0.0s
75	48	40	34180	0.016s	0.012s	42	12247	0.008s
76	57	49	28148	0.020s	0.004s	49	11507	0.008s
77	54	44	19843	0.012s	0.004s	46	9894	0.008s
78	53	45	22231	0.012s	0.008s	47	11440	0.004s
79	42	36	1870	0.0s	0.0s	36	1040	0.0s
80	57	49	27860	0.016s	0.012s	51	17595	0.008s
81	53	49	1644	0.0s	0.0s	49	1003	0.004s
82	62	52	175685	0.100s	0.060s	52	97004	0.060s
83	49	43	13438	0.008s	0.004s	45	7756	0.004s
84	55	47	25486	0.016s	0.008s	47	16520	0.008s
85	44	40	1123	0.0s	0.0s	40	807	0.0s
86	45	37	4471	0.004s	0.004s	39	2406	0.0s
87	52	44	6095	0.004s	0.0s	44	4016	0.004s
88	65	53	433966	0.236s	0.144s	55	229248	0.140s
89	54	46	48025	0.028s	0.016s	46	22153	0.016s

Caso		Sólo patrón				Regiones de blancos		
No.	Sol.	H ₀	Nodos	Tiempo-NL	Tiempo-L	H ₀	Nodos	Tiempo
90	50	44	6450	0.004s	0.004s	44	3181	0.0s
91	57	47	144976	0.076s	0.044s	49	80913	0.048s
92	57	45	39107	0.020s	0.016s	47	21886	0.012s
93	46	42	249	0.0s	0.0s	44	177	0.0s
94	53	49	2342	0.004s	0.0s	49	1514	0.0s
95	50	42	4802	0.004s	0.004s	44	2065	0.004s
96	49	41	5185	0.0s	0.0s	41	3188	0.0s
97	44	40	1029	0.004s	0.0s	40	686	0.0s
98	54	46	87716	0.048s	0.032s	46	39042	0.028s
99	57	49	13302	0.008s	0.004s	51	5660	0.004s
100	54	42	75688	0.044s	0.024s	46	38996	0.024s

Cuadro 21: Resultados individuales de las 100 instancias del 15-Puzzle

Caso		Sólo patrón				Regiones de blancos		
No.	Sol.	H ₀	Nodos	Tiempo-NL	Tiempo-L	H ₀	Nodos	Tiempo
1	96	82	39909459	18.569s	11.564s	82	30405109	17.797s
2	82	68	47776619	21.897s	13.516s	68	32592093	18.213s
3	81	65	283511182	2m11.46s	1m20.801s	65	226607069	2m7.78s
4	97	81	450735877	3m31.385s	2m10.084s	81	323844014	3m7.832s
5	93	81	892770469	7m0.13s	4m17.58s	81	437378546	4m15.9s
6	100	84	1424467822	11m27.979s	6m55.026s	84	777964095	7m33.044s
7	92	74	1459260731	11m17.27s	7m2.35s	74	1003979379	9m44.889s
8	101	83	2415703993	18m27.97s	11m45.68s	83	1934469869	18m40.54s
9	95	81	2056152603	15m51.563s	9m53.789s	81	715868830	6m56.102s
10	98	84	2257859226	17m19.18s	10m43.104s	84	1588885427	15m3.992s
11	90	70	3314354791	25m40.55s	15m50.867s	70	1574885980	14m46.411s
12	100	84	2238935189	17m25.58s	10m49.093s	84	1775034059	17m33.72s
13	95	79	3104591389	24m8.22s	15m2.18s	79	1241261972	12m6.809s
14	96	78	3527758861	26m43.2s	16m41.74s	78	2376162763	22m15.82s
15	98	78	5613901147	43m8.79s	26m58.51s	78	3512027726	33m52.01s
16	105	87	8313562913	1h3m44.3s	39m17.84s	87	4878106264	46m7.48s
17	97	77	12679929426	1h37m6.45s	1h0m38.65s	77	7521152074	1h12m19.08s
18	99	79	25893358275	3h21m40.6s	2h4m18.06s	79	19624687662	3h6m50.3s
19	106	90	26250496925	3h24m5.2s	2h6m12.45s	90	15505806483	2h29m43.17s
20	92	74	27755420584	3h28m27.4s	2h8m57.26s	76	13246442277	2h3m34.56s

Cuadro 22: Resultados individuales de las 20 instancias del 24-Puzzle