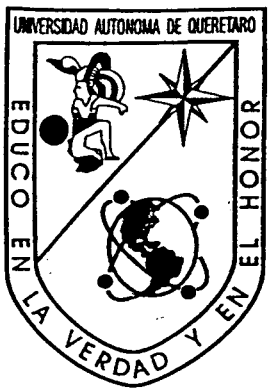


UNIVERSIDAD AUTONOMA DE QUERETARO

FACULTAD DE INFORMATICA

BUSQUEDA UNIAGENTE EN PRESENECIA DE BLOQUEOS

TESIS

QUE COMO PARTE DE LOS REQUISITOS PARA
OBTENER EL GRADO DE
MAESTRO EN CIENCIAS COMPUTACIONALES

PRESENTA:
GUSTAVO ALONSO HERNANDEZ GUZMAN

SANTIAGO DE QUERÉTARO, QRO., AGOSTO DE 2001.

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INFORMATICA
MAESTRÍA EN CIENCIAS COMPUTACIONALES

NOMBRE DE LA TESIS

BÚSQUEDA UNI AGENTE EN PRESENCIA DE BLOQUEOS

Que como parte de los requisitos para obtener el grado de

MAESTRO EN CIENCIAS COMPUTACIONALES
Especialidad Ingeniería de Software

Presenta:
Gustavo Alonso Hernández Guzmán

Dirigido por:
DR. Jaime Rangel Mondragón

SINODALES

Dr. Jaime Rangel Mondragón
Presidente

[redacted]

M.C. Arturo González Gutiérrez
Secretario

[redacted]
Firma

M.C. Ruth Angélica Rico Hernández
Vocal

[redacted]
Firma

Dr. Iván Terol Villalobos
Suplente

M.C. Alberto Lamadrid Alvarez
Suplente

[redacted]
Posgrado

Ing. Teresa Guzmán Flores
Nombre y Firma
Director de la Facultad

Centro Universitario
Querétaro, Qro.
Enero de 2001
México

RESUMEN

La búsqueda uni-agente tiene un amplio uso en una gran variedad de aplicaciones en la industria, bodegas, almacenes y demás áreas que requieren de la organización y clasificación de objetos. Dado un agente en un ambiente cerrado y con obstáculos, se desea desarrollar una estrategia que permita resolver problemas de reinstalación general de objetos. Esto involucra el análisis y diseño de algoritmos que permitan implementar un mecanismo sencillo en cuanto a su construcción y tecnología aplicada, que a base de un solo tipo de movimiento, en este caso, empujar, permita reubicar objetos. Esta investigación contribuye con la ingeniería, al poder aplicar en trabajos posteriores, los resultados obtenidos y por medio de ellos controlar los procedimientos de reubicación al utilizar un mecanismo que mueva los objetos hacia destinos establecidos, abatiendo costos generados por la administración requerida. En el capítulo I se estudia el juego del 15 el cual proporciona una introducción muy interesante para poder ingresar al análisis y estudio del BoxWorld y Sokoban, los cuales constituyen nuestro objetivo final. El juego del 15 es resuelto aplicando el algoritmo Backtrack, el cual proporciona conocimientos básicos para el análisis del problema a fondo. En el capítulo II se analizan otros algoritmos incluyendo el algoritmo A* mediante el cual se puede apreciar la gran diferencia existente en tiempos y resultados obtenidos entre el algoritmo Backtrack y A*. En el Capítulo III se resuelven los juegos BoxWorld y Sokoban utilizando el algoritmo A*; estos juegos aportan una imagen muy apegada a la realidad del problema en la industria, presentándolo en una forma muy amena e interesante. El capítulo IV presenta los resultados obtenidos en la solución de solo algunos de los niveles del juego. Este trabajo únicamente se concreta a la investigación y aplicación de dichos algoritmos computacionales y no a la implementación mecánica o externa a la computadora, la cual requiere una interface de comunicación entre la computadora y los elementos mecánicos.

(**Palabras clave**: Sokoban, BoxWorld, Algoritmos, análisis)

SUMMARY

The one-agent search is of widespread use in a variety of industrial applications. Given an agent positioned in a closed environment with obstacles, our aim in this work is to find a suitable strategy to solve the general object-repositioning problem. This involves the analysis and design of algorithms that facilitate the implementation of an atomic mechanism based on one type of movement - namely the pushing of an object, is developed in industries, stores, warehouses and other areas that require object reorganization and classification. This work contributes to the engineering aspects in its application in future works controlling the repositioning procedures and lowering the costs implicit in the underlying administration. We begin with the study of the 15-puzzle, which provides a very interesting introduction to enter the analysis of BoxWorld and Sokoban, our main goals. The 15-puzzle is solved by applying a Backtrack algorithm, which precedes a deeper analysis of the problem. We also analyse other algorithms including the A* algorithm. We also compare the differences in time and results obtained between Backtrack and A* algorithms. We attempt to solve the BoxWorld and Sokoban models using the A* algorithm. These games introduce an image very close to the real-life aspects of the problem, representing it in such an interesting and pleasant form whose one-level at a time solutions result appealing to the solver. Finally, the results obtained in the solution of some levels of the game are presented. This work concentrates mainly on the investigation and application of such computational algorithms and not on the mechanic implementation or hardware requiring an interface of communication between the computer and the mechanic components.

(**Key Words**: Sokoban, BoxWorld, algorithms, analysis)

Dedicada a

Irma e Irma Ireri

Agradezco eternamente a mi esposa Irma y a mi hija Irma Ireri, por el apoyo incondicional y muestras de cariño que de ellas siempre he recibido.

A mis Padres

Por el ejemplo de tenacidad y perseverancia que siempre me han enseñado.

AGRADECIMIENTOS

Agradezco muy especialmente a todos aquellos que, por su apoyo y comprensión estoy aquí, en esta vida, en este lugar y en este momento.

Agradezco al Dr. Jaime Rangel Mondragón, por sus enseñanzas y por su gran espíritu de superación académica e investigación.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Tabla **Página**

INTRODUCCIÓN

La búsqueda uni-agente es una herramienta poderosa en la solución de una gran variedad de problemas que requieren encontrar rutas óptimas.

La mayor parte de las técnicas que son usadas en la exploración y búsqueda uni-agente tienen la propiedad de que si se comienza en un estado soluble, en ningún momento de la búsqueda se puede llegar a un estado que sea insoluble.

En este trabajo se estudiarán las implicaciones que ocurren cuando las transiciones de estados dan lugar a estados insolubles (bloqueos). También se introduce lo que se conoce como *búsqueda de patrones*, que es un algoritmo que identifica las condiciones mínimas (patrones) necesarios para un bloqueo y aplica ese conocimiento a la eliminación de partes probablemente irrelevantes del árbol de búsqueda

Se escogió este tema de investigación ya que muchos problemas, tales como los puzzles[1] de corrimiento de fichas, generan arboles de búsqueda en los que diferentes nodos tienen diferentes números de hijos, en este tipo de problemas el número de hijos depende de la posición del espacio en blanco, la solución de este tipo de problemas pueden ser llevados a la práctica aplicándolos en la solución de problemas cotidianos en almacenes.

Este trabajo fue desarrollado con varios objetivos en mente. El primero y más obvio es el de encontrar la solución al juego del SOKOBAN programando el algoritmo de solución en Pascal proporcionando una configuración inicial y arrojando este los movimientos necesarios para solucionar el problema.

[1] Se utilizará su nombre en inglés (Puzzle) y no su significado en español (rompecabezas) ya que es así como se conoce y se hace referencia en la mayoría de la literatura que se refiere al tema.

Otro de los objetivos es el de encontrar una solución óptima, esto se logra aplicando diferentes algoritmos y seleccionando el que mejores resultados proporcione.

Lo anterior involucra la investigación y comparación de algoritmos que garanticen encontrar rutas o soluciones óptimas, una vez encontrado el algoritmo y las soluciones óptimas, es posible aplicarlos en la solución de problemas reales que se presentan en la industria, talleres, almacenes, áreas de logística etc.

La búsqueda es un proceso importante en la mayoría de los sistemas de Inteligencia Artificial, principalmente en los sistemas de producción que consisten de una base de datos global, un conjunto de reglas y un sistema de control. El uso de información heurística es necesario en la mayoría de los procesos de búsqueda de estos sistemas, el concepto de información heurística es muy importante ya que es el tema principal dentro del desarrollo de los algoritmos.

Los puzzles y juegos proporcionan una gran cantidad de problemas a manera de ejemplo para mostrar y probar métodos de solución. Muchos programas son diseñados y escritos para solucionar algunos tipos de puzzles que para los humanos son difíciles de resolver.

El gran valor educacional que tiene el SOKOBAN y en general los puzzles es que tanto niños como adultos disfrutan pensando; uno de los grandes beneficios de las computadoras personales es que ayudan tanto a niños como adultos a desarrollar una afición por los juegos de lógica que ejercitan el cerebro.

Existe un interés especial en jugar juegos de lógica en donde cada nivel es ligeramente más difícil que el anterior, en el libro "The art of human-computer interface design" [www.10 The educational Value of Sokoban], el diseñador del software hace la siguiente observación: "A los niños les gustan los retos y juegos excitantes, cuando un juego es atractivo y proporciona constantes incrementos de niveles y de dificultad, estos se pueden interesar por largo tiempo".

Una de las razones por las que el juego del SOKOBAN es especialmente interesante es debido a que puede ayudar al desarrollo de la visualización y al razonamiento y a adquirir habilidades analíticas.

La solución de este tipo de problemas ha tenido especial énfasis en el National Council of Teachers of Mathematics (NCTM). La NCTM en 1989 dijo que los estudiantes deben de practicar tratando de solucionar este tipo de problemas, para obtener una capacidad de desarrollo y aplicación de estrategias que ayuden a la solución de una gran variedad de problemas.

En Internet se puede encontrar una gran cantidad de trabajos y reportes sobre este juego, incluyendo una gran variedad de presentaciones que hacen atractivo jugarlo; muy pocos trabajos presentan algún modelo matemático y aún menos trabajos presentan estudios completos sobre la forma de solucionarlo.

La presente tesis está formada por 4 capítulos. En el capítulo I se analiza el juego del 15 o 15-Puzzle como es conocido en gran cantidad de libros y artículos así como en direcciones de Internet, éste juego es uno de los más comunes y al alcance de todos. Es un tablero con 15 fichas, numeradas del 1 al 15 y un espacio en blanco que sirve para deslizar las fichas una a una hasta lograr ordenarlas de menor a mayor de izquierda a derecha y de arriba hacia abajo. Esto permite dar el primer paso en el razonamiento lógico que requieren los puzzles y preparar al lector para la investigación de un juego más complejo como el Sokoban.

El juego del 15 se resuelve utilizando dos algoritmos: el algoritmo BACKTRACK y el algoritmo A*, este último será analizado en el capítulo II, en este análisis es posible observar las grandes bondades que proporciona el algoritmo A* y aunque su programación es más compleja al final arroja los resultados esperados.

Se utilizó lenguaje Pascal Versión 7.0, ocupando únicamente listas y apuntadores, lo que redujo mucho el tiempo de ejecución, pero rápidamente consumió el máximo de memoria que utiliza Pascal.

Finalmente se programó en Delphi 5, aprovechando el código Pascal elaborado y gracias a la optimización y mejor paginación de memoria de este lenguaje se logró obtener resultados satisfactorios.

El equipo utilizado par el desarrollo de la programación es una computadora Pentium III 500 Mhz y 64 MB de memoria principal.

El capítulo II se dedica al estudio del algoritmo A* como una especie de preparación para resolver el Sokoban utilizando este algoritmo en el capítulo III, después de haber realizado un detallado análisis del entorno y comportamiento del Sokoban.

En el capítulo IV se presenta el código de los programas PASCAL que se desarrollaron en los capítulos I y III para resolver configuraciones reales de este juego y como demostración de los resultados que se obtuvieron al resolver los niveles del Juego. Se proporcionan también listados con los movimientos que resuelven algunos de los niveles del Sokoban resuelto.

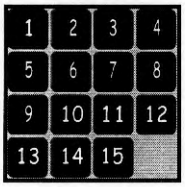
La solución de problemas puede verse como un tópico extremadamente vago, pero muchas de las investigaciones de inteligencia artificial están concentradas en este tipo de soluciones; en este sentido la solución de problemas abarca todas las ciencias computacionales, porque cualquier tarea computacional puede ser considerada como un problema a ser resuelto. Para nuestro propósito se acotará el problema, excluyendo aquellos que usen matrices de orden superior a 20X20.

Cuando se examinan varios métodos de solución de problemas como el algoritmo, se encuentra que muchos de ellos usan el método de búsqueda de

prueba y error, este método resuelve el problema pero dentro de un espacio de posibles soluciones demasiado grande. El propósito de este trabajo es el de utilizar uno de los mejores métodos de solución de problemas basados en búsquedas.

"No es que los juegos y los problemas matemáticos sean escogidos por ser simples y claros; al contrario, ellos causan, dentro de la más pequeña estructura inicial, la más grande complejidad, de tal forma que uno puede encontrar situaciones realmente difíciles seguidas de una diversión relativamente mínima en el momento de programarla".

M. Minsky

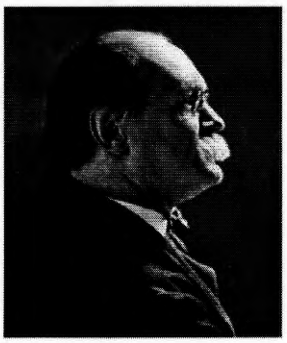


CAPÍTULO I

JUEGO DEL 15

1.1 HISTORIA DEL JUEGO DEL 15.

En la década de los 70's del siglo XIX, surgió en los Estados Unidos el juego del 15, éste, rápidamente obtuvo una incontable cantidad de adeptos que querían resolver el misterio; aquí fue donde comenzó la "fiebre".



Samuel Loyd
(1841-1911)

Lo mismo sucedió en Europa, en donde se podían ver a los jinetes montados en sus caballos con el juego en sus manos; en oficinas y tiendas fue verdaderamente impresionante ver como los empleados se distraían de sus ocupaciones en horas laborales. Esto también sucedió en los solemnes salones del Reichtag en Alemania. Hay que mencionar que El gran geógrafo y matemático Sigmund Gunter era diputado y adicto a este juego durante esta epidemia.

El juego del 15 (15-Puzzle) es un juego creado por Samuel Loyd hace más de 100 años, que llegó a ser un gran suceso en su época, tal como lo fue el cubo de Rubik 100 años después en la década de los 80's, éste último sólo es una variación, como muchas otras que hay del juego del 15.

Portada de una publicación de Samuel Loyd.

Samuel Loyd fue ampliamente conocido en su época como un gran autor de problemas divertidos y multitud de rompecabezas. Nació el 31 de Enero de 1841 en Pennsylvania, USA y murió el 10 de Abril de 1911 en Nueva York, USA. Estudió Ingeniería pero dedicó gran parte de su vida a los problemas relacionados con el ajedrez y los puzzles. En 1860 fue editor de la revista Chess Montly y en 1878 publicó un libro de problemas cuyo título es Chess Strategy[2]. Curiosamente, no pudo patentar su juego del 15 en los Estados Unidos, de acuerdo a las reglas, él

[2] Biography in Encyclopaedia Britannica, M Gardner, Mathematical Puzzles of Sam Loyd. Selected and Edited by Martin Gardner (New York, 1959). http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Loyd.html

tenía que presentar un "modelo de trabajo", de tal forma que el prototipo pudiera ser manufacturado a partir de éste.

Portada de una publicación de
Samuel Loyd.

Él presentó el problema a The Patent Office Oficial, pero cuando le preguntaron si tenía solución, su respuesta fue: "No, es matemáticamente imposible", a lo que el oficial razonó: "En este caso no puede tener un modelo de trabajo y sin un modelo de trabajo no puede ser patentada", Loyd aceptó esta decisión.

1.2.- DEFINICION DEL JUEGO DEL 15 (15 PUZZLE).

El juego del 15, consiste de un tablero dividido en forma de matriz de tamaño 4X4, con 16 cuadros iguales, cada uno ocupado por una ficha identificada con un número consecutivo del 1 al 15. El decimosexto cuadro está vacío. El orden de las fichas en el tablero es de manera ascendente, de izquierda a derecha y de arriba hacia abajo.

Teniendo las fichas dispuestas en desorden tal como se muestra en la Figura 1.1 a), y utilizando únicamente el movimiento de deslizar sucesivamente los cuadros adyacentes al espacio en blanco (las fichas no se pueden despegar o separar del tablero solo deslizar sobre el tablero), uno a la vez, hacia dicho espacio, en el caso de la Figura 1.1 a) solo se pueden deslizar los cuadros marcados con los números 7 y 15 ya que son los únicos adyacentes al espacio, si se mueve el cuadro marcado con el 7, enseguida se pueden mover los cuadros marcados con los números 11, 3 y el mismo 7 como se indica en la Figura 1.1 c), solo que este, el movimiento del 7, regresaría el juego a la configuración inicial, pero si en lugar del cuadro con el número 7, el que se desliza es el marcado con el número 15, enseguida los cuadros que se pueden deslizar son los marcados con los números 3, 2 y el mismo 15 tal como se muestra en la Figura 1.1 d), el objetivo del juego es ordenarlos de manera ascendente siguiendo esta metodología hasta lograr alcanzar el estado que se muestra en la Figura 1.1 b).

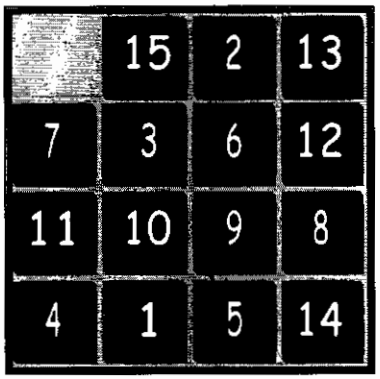


Figura 1.1 a) Estado inicial, cuadros dispuestos aleatoriamente.

Figura 1.1 b) Estado objetivo, cuadros ordenados ascendentemente.

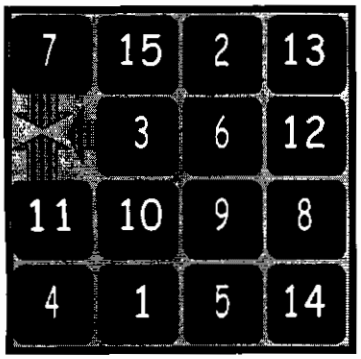


Figura 1.1 c) Opciones de movimientos después de deslizar la ficha con el 7.

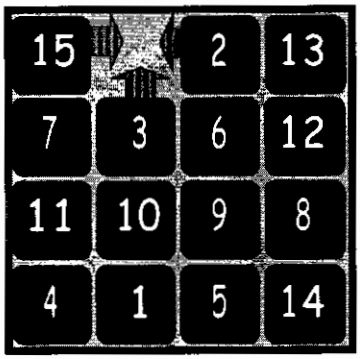


Figura 1.1 d) Opciones de movimientos después de deslizar la ficha del 15.

El juego termina una vez que todas las fichas están en orden y el espacio en blanco se encuentra en la decimosexta posición.

Lo anterior se indica de la siguiente manera en notación matemática:

La matriz formada por las fichas del tablero puede ser representada en un arreglo uni-dimensional si se utiliza una función llamada *h*, la cual al aplicarla convierte el arreglo bi-dimensional en un arreglo uni-dimensional tal como se muestra en la Figura 1.2.

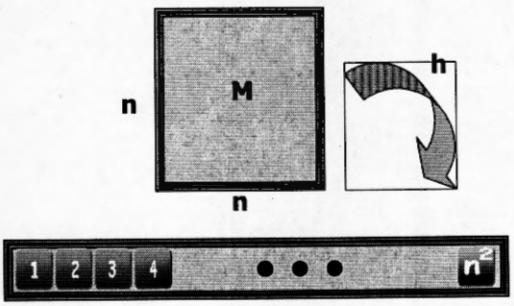


Figura 1.2 Convertir un arreglo bi-dimensional en uni-dimensional.

$(1,1) \rightarrow 1$

$(1,2) \rightarrow 2$

$(1,n) \rightarrow n$

$(2,1) \rightarrow n+1$

...

$(2,n) \rightarrow n+n \rightarrow 2n$

$(3,1) \rightarrow 2n+1$

$(3,n) \rightarrow 3n$

...

$(n,n) \rightarrow (n-1)n+n \rightarrow n^2$

M denota el contenido de la matriz o arreglo bi-dimensional y *n* el tamaño de dicho arreglo.

Finalmente $h:[1,n]^2 \rightarrow [1,n^2]$, es una función biyectiva

M se dice ordenada si ***h[M]*** esta ordenado.

De manera que ***h(i,j)=n(i-1)+j*** es biyectiva ya que

$h^{-1}(k)=(i,j)$

j=k-n(i-1)

$i=\lceil k/n \rceil$

Por ejemplo, si se considera la configuración del juego del 15 mostrada en la Figura 1.3, y se aplica la función *h* para convertirla en arreglo uni-dimensional:

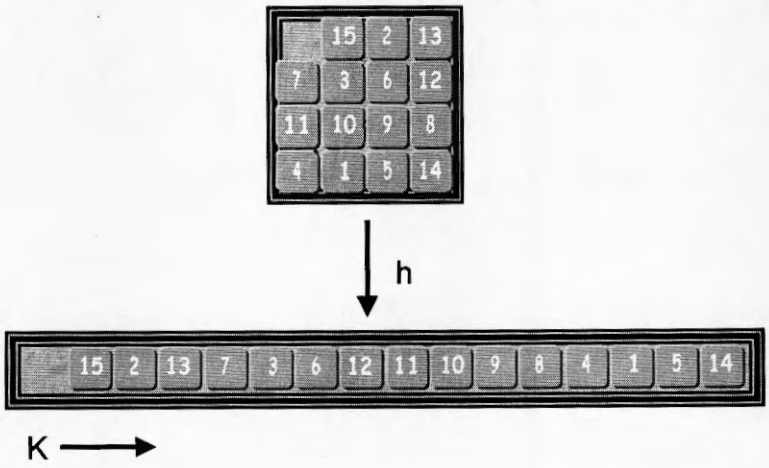


Figura 1.3 Aplicación de la función h a la configuración mostrada.

Si se elige el casillero 7, al aplicar la función *h* se obtienen las coordenadas (2,3), las cuales indican el casillero de la ficha marcada con el número 6, que es la misma ficha del casillero 7 del vector.

$$h^{-1}(7)=(2,3)$$

Algunos de los arreglos iniciales que se intentan resolver no tienen solución, esto se estudiará en la siguiente sección. Analizando el juego del 15, se puede encontrar que existen 16! diferentes configuraciones de los 15 cuadros y el espacio en blanco, esto es 20,922,789,888,000 posibles configuraciones diferentes.

1.3. INVERSIONES.

Considere el arreglo de la Figura 1.4.

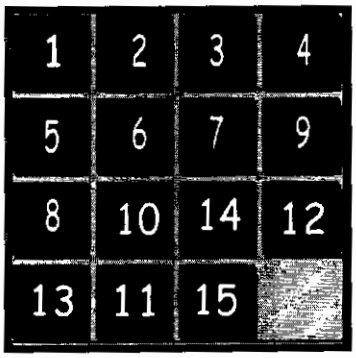


Figura 1.4 Tablero con 6 inversiones.

El primer renglón está en el orden deseado, el segundo renglón no está en orden por la posición de la ficha 9 situada antes del 8. Esta precedencia de la ficha 9 ante la ficha 8, se llama *inversión*. Además la ficha 14 precede a tres fichas con números menores que él: la 12, 13 y 11, así el 14 tiene tres inversiones. La ficha 12 y 13 precede a la 11. Esto da un total de 6 inversiones.

De manera general, sea N el número total de inversiones en un arreglo dado y a_i un elemento del tablero, si existen n elementos menores que a_i, en cuadros delante de a_i, se llama a esto una *inversión de orden n*, denotada por n_i. Entonces :

$$N = \sum_{i=1}^{15} n_i$$

En el caso del ejemplo de la Figura 1.2, si el número total de inversiones es par, esto indica que el arreglo puede ser ordenado ascendentemente, en otras

palabras dicho arreglo si tiene solución, de lo contrario si el número total de inversiones es impar, el arreglo no tiene solución.

Teorema 1.1. [www.9]

Para una configuración dada, supóngase que *N* denota la suma del número total de inversiones, entonces (*N* mod 2) no varía ante cualquier movimiento válido, en otras palabras después de un movimiento válido la variación de una *N* impar permanece impar y una *N* par permanece par.

Demostración:

En la Figura 1.5 se puede apreciar la ubicación de las fichas *a, b, c* y *d* en una configuración dada del tablero, la ***x*** representa cuadros ocupados por fichas cuyo valor, para el caso de este ejemplo, puede ser cualquier número.

Deslizar un cuadro horizontalmente (Figura 1.5), no cambia ni el número total de inversiones ni el número de renglón donde se encuentra el cuadro vacío debido a que el orden de las fichas es de izquierda a derecha y de arriba hacia abajo dentro del tablero.

Considere ahora, el deslizar una ficha verticalmente ya sea la situada arriba o la situada abajo del espacio.

Seleccione la ficha *a* que está situada directamente sobre el espacio vacío tal como se muestra en la Figura 1.5, si se desliza hacia abajo cambia la paridad del número de renglón del cuadro vacío.

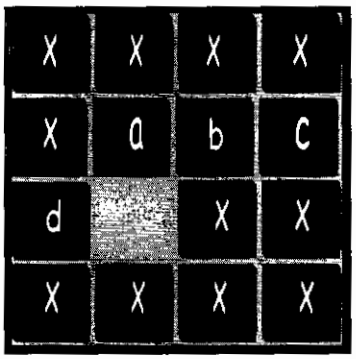


Figura 1.5. Ubicación de las fichas *a, b, c* y *d* en una configuración dada.

Ahora considérese el número total de inversiones. El movimiento solo afecta las posiciones relativas de las fichas *a, b, c, y d*, si ninguna de las fichas *b, c,* y *d* causan una inversión relativa a *a* (si los tres son mayores que *a*), entonces después de realizar este deslizamiento, se obtienen 3 inversiones adicionales. Si uno de los tres es más pequeño que *a*, entonces, antes del movimiento, *b, c* y *d* forman una inversión simple con relación a *a*, y después, de mover la ficha *a*, solo se forman dos inversiones, lo que proporciona un cambio en el número de inversiones con una diferencia de 1, lo que es también un número impar. Dos casos adicionales obviamente conducen al mismo resultado. Por lo que el cambio en la suma *N* es siempre par, ver Figura 1.6.

a<b, a<c, a<d } Variación de N de 0 a 3

a<b, a>c, a>d
a>b, a<c, a>d } Variación de N de 2 a 1
a>b, a>c, a<d

a<b, a<c, a>d
a<b, a>c, a<d } Variación de N de 1 a 2
a>b, a<c, a<d

Figura 1.6. Variación de n después del movimiento de ***a***.

El teorema anterior aplica también para el juego del 8 de Sam Loyd (3X3) o el juego del 24 (5X5).[www 9].

El tablero del juego del 15 puede ser representado en forma abstracta, por un grafo 4X4, Figura 1.5 [Stefan Edelkamp, Korf Richard E.], en donde cualquier posición corresponde a un nodo. Las líneas representan los movimientos de los cuadros que componen el tablero [www 3].

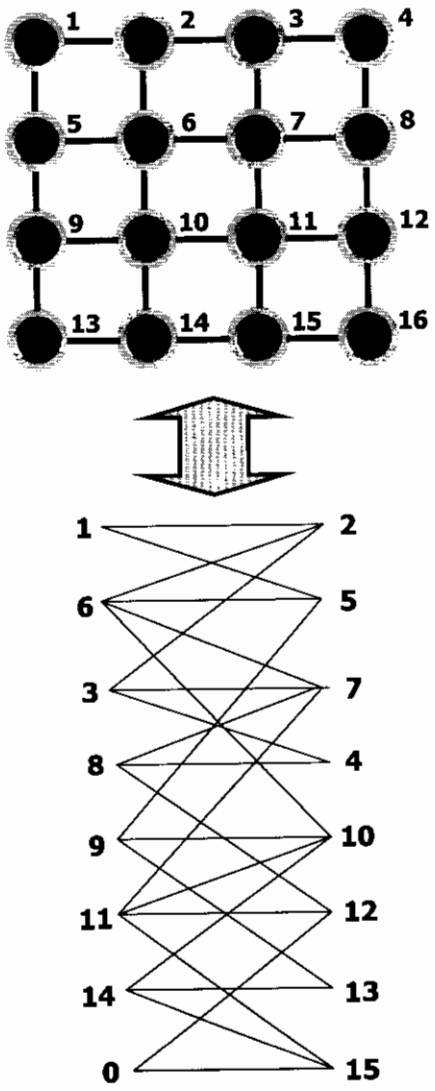


Figura 1.7 Representación del juego del 15 con grafos.

Etiquetar el grafo, significa asignar un número o letra único a cada uno de sus lados, el 0 corresponde a un nodo vacío.

1. Para un grafo G, puz(G), es un grafo cuyos nodos están todos etiquetados.
2. Dos nodos de puz(G), están conectados por una arista si y solo sí hay un movimiento válido que mueve una etiqueta dentro de otra.

Hay que tener en cuenta que, como un movimiento cambia las etiquetas, una etiqueta no puede ser transformada a sí misma realizando un movimiento válido. Lo anterior indica que puz(G) representa un grafo simple.

Un componente conectado de puz(G), consiste de aquellas etiquetas que pueden ser transformadas una dentro de la otra por medio de una secuencia de movimientos válidos. Si se prueba que el puz(G) es un grafo conexo, entonces el juego en G puede ser resuelto a partir de cualquier configuración inicial. Por el contrario, para mover una etiqueta a otra es necesario estar dentro de un componente adyacente a aquel a donde se quiere mover en el grafo puz(G).

Teorema 1.2:

Sea *G* un grafo conexo simple representado como un polígono tal como se muestra en la Figura 1.6

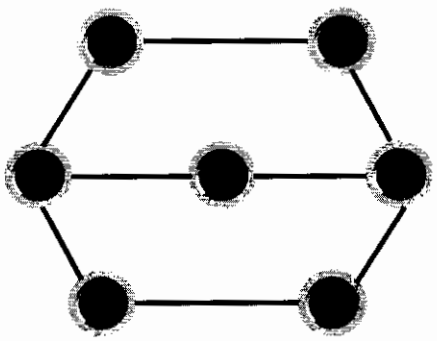


Figura 1.8 Grafo G, representado como un polígono.

Entonces *puz(G)* es conexo a menos que G sea bipartito, en tal caso *puz(G)* tiene exactamente dos componentes, etiquetas y conectores. Las etiquetas *f* y *g* en *G* tienen vértices vacíos en distancias pares en *G* están en el mismo componente de *puz(G)* si y solo si fg^{-1} es una permutación par de *V(G)*.

Demostración.

La *distancia* entre dos nodos es la longitud de la ruta más corta de un nodo a otro. De manera que, a no ser por la noción de permutación, la cual se estudiará en la siguiente sección, se podría interpretar el resultado. El teorema resuelve completamente el problema del juego del 15, las gráficas son siempre bipartitas y *puz(G)* tiene siempre dos componentes. Intercambiar dos fichas hace incompatibles a las etiquetas, perteneciendo a diferentes componentes conexos de *puz(G)*.

El intercalar dos posiciones significa aplicar una transposición a las etiquetas. Mover el cuadro vacío es equivalente a aplicar una secuencia de transposiciones; el número de transposiciones es igual al número de movimientos o a la distancia entre el lugar inicial y el lugar final del cuadro vacío.

El teorema entonces establece que si dos etiquetas tienen la misma paridad ellas pertenecen al mismo *puz(G)* $\Leftrightarrow$ la distancia entre dos cuadros vacíos en las dos etiquetas es par. Si dos etiquetas tienen diferentes paridades pertenecen al mismo componente de *puz(G)* $\Leftrightarrow$ la distancia entre los cuadros vacíos en las dos etiquetas es par.

Lema 1.1.

Sea *n=4;* si se seleccionan dos cuadros adyacentes, existe una secuencia de movimientos que al final deja todas las fichas en su posición inicial. Únicamente las dos fichas seleccionadas sufren un cambio con relación a su posición inicial; como resultado de esta secuencia de movimientos dichas fichas solo intercambian sus posiciones.

Demostración.

Para intercambiar dos fichas *a* y *b* (Figura 1.9) sin que al final de los movimientos se vea afectada la posición del resto de ellas, es necesario un movimiento horizontal para colocar *a* en la posición de *d;* con un segundo movimiento pero en dirección vertical se coloca *b* en la posición de *c,* ahora si se intercambian los cuadros *c* y *d,* enseguida se efectúan movimientos contrarios a los originales, primero vertical y después horizontal.

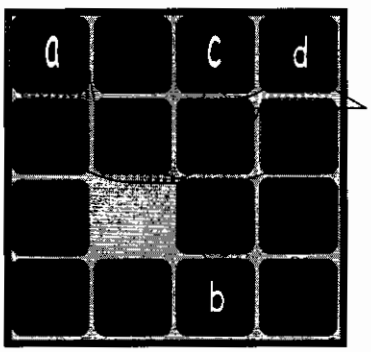


Figura 1.9 Intercambiar c y d.

Debido a que solo se puede hacer un intercambio entre dos cuadros adyacentes, se puede también cambiar dos cuadros los cuales no necesariamente tienen que ser adyacentes y acomodarlos como nos convenga para aplicar este lema. Por lo tanto el lema conduce a una proposición obvia. Primero se

intercambia el cuadro #1 con cualquier otro que ocupe una posición a la derecha a la esquina superior izquierda, encuentre el cuadro #2 e intercámbielo a esta posición, enseguida al # 1, etc.

Debido a que la prueba es constructiva, se sigue la secuencia de movimientos que demuestra la veracidad del lema.

Una de las dificultades es la de inventar las notaciones concisas que describen la secuencia. La secuencia que consiste de dos movimientos A y B ejecutados consecutivamente se representan como AB.

1.4 PERMUTACIONES

Una de las reglas de la multiplicación de arreglos es llamada permutación cuando los objetos de una configuración resultante tienen diferente orden a la configuración original.

Frecuentemente se tiene un conjunto de objetos *G* para los cuales hay una forma de definir la multiplicación, de tal manera que las siguientes propiedades se satisfacen [Grimaldi Ralph P., 1994].

1. El producto de dos elementos de *G* es siempre un elemento de G.
2. Para *a, b* y *c* en *G* se tiene que *a(bc)=(ab)c.*
3. Hay un elemento identidad *I* en *G* tal que para cualquier *x* en *G,*

$$Ix = xI = x$$

4. Para cualquier *x* en *G,* hay un elemento inverso x^{-1} en *G* tal que

$$xx^{-1} = x^{-1}x = I$$

Por las propiedades anteriores *G* es un grupo.
Si adicionalmente a*b=b*a para toda a,b$\in$ G, entonces G es un grupo Abeliano [Grimaldi Ralph P., 1994].

Ejemplo 1 [Grimaldi Ralph P., 1994**].**
El grupo S_4 esta formado por las 24 permutaciones de los elementos {1,2,3,4}; La identidad es $I=\begin{pmatrix}1234\\1234\end{pmatrix}$; sí $\alpha=\begin{pmatrix}1234\\2143\end{pmatrix}, \beta=\begin{pmatrix}1234\\3124\end{pmatrix}$, entonces $\alpha\beta=\begin{pmatrix}1234\\1342\end{pmatrix}$; pero $\beta\alpha=\begin{pmatrix}1234\\4213\end{pmatrix}$, de tal forma que S_4 es NO Abeliano.

Ejemplo 2.
Sea G={1,3,7,9}, si se define el producto de *x* y *y* como el dígito unidad del entero resultante de *xy,* se tiene un grupo. Por ejemplo 3 * 7=21 cuyo dígito unidad es 1,

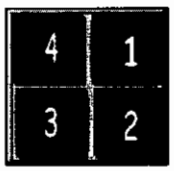
7 * 9=63, en G se dice que 7*9=3. Examinando todos los casos posibles, se determina que G con este producto es un grupo, la tabla de multiplicación para este grupo es:

*	1	3	7	9
1	1	3	7	9
3	3	9	1	7
7	7	1	9	3
9	9	7	3	1

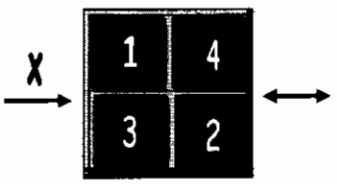
Las permutaciones se pueden apreciar en un tablero de un puzzle, cada configuración diferente obtenida al realizar movimientos a sus fichas representan un estado de la permutación.

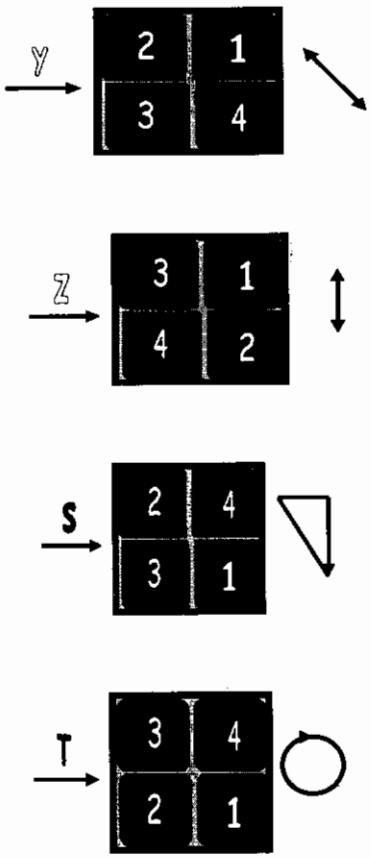
El número de permutaciones de las fichas con que cuenta el tablero es la totalidad de las configuraciones que se pueden generar en dicho tablero.

Considérese un juego de 2X2, cuyo estado inicial es:



Y sus movimientos son:





Las etiquetas de las flechas a la izquierda representan la variable con la que se identifica el movimiento y las flechas a la derecha de los tableros representan el movimiento que se realizó.

En donde:

$$XY = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix}$$

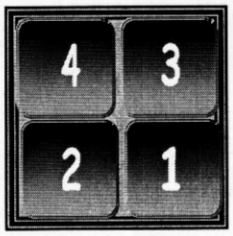
Representa el producto de dos permutaciones, esto es en el caso de X, el 1 se intercambia con la posición del 4, el 2 y 3 no tienen modificación y el 4 se intercambia con el 1, de manera similar se interpretan las posiciones para Y.

Si se efectúa la operación XX y se analizan los resultados de YY, ZZ, SSS y TTTT se observa lo siguiente:

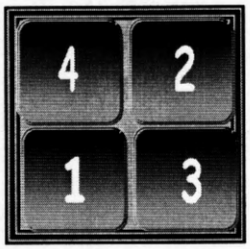
$$X^2=Y^2=Z^2=S^3=T^4=I$$

de la misma manera $(XT)^3=I$, como se ilustra en la Figura 1.10.

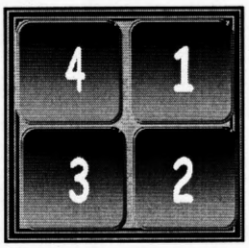
XT=



$(XT)^2$=



$(XT)^3$=



$=T^4 = I$

Figura 1.10 operaciones que conducen a la matriz identidad.

De lo anterior se puede establecer el siguiente algoritmo:

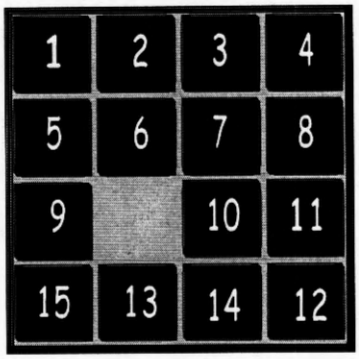
1) Usando T ubicar el 1
2) Usando XT o $(XT)^2$ ubicar el 2
3) Usando TXT^3 resolver

Lo cual lleva a establecer el siguiente teorema.

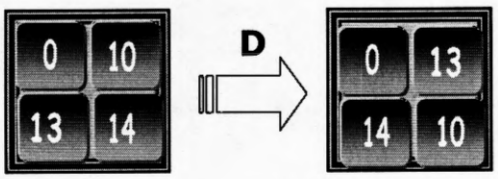
Teorema 1.3.

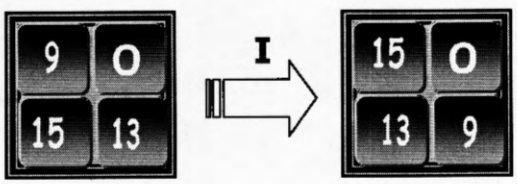
Usando la nomenclatura S y T, máximo se necesitan 5 movimientos para resolver el juego.

Si se tiene la siguiente configuración:

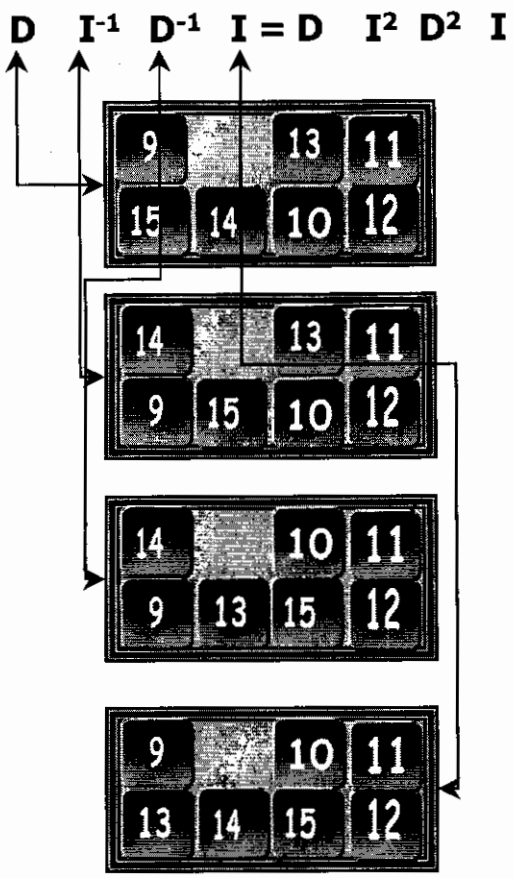


Los movimientos que lo solucionan se pueden resumir en los siguientes:





Los movimientos que lo resuelven son los siguientes:



Estos movimientos son para el caso en que el espacio vacío se encuentre en la posición indicada en el tablero, solo se consideran los últimos dos renglones del mismo lo que se reduce a encontrar la solución de un juego con configuración de n^2-1.

Para la solución del problema, existen dos funciones de evaluación que estiman el número de movimientos que se deben realizar con el fin de obtener un estado objetivo, conocidas como aquel que toma en cuenta el *Número de fichas fuera de lugar* y *la Distancia Manhattan [*Chung-Hung Tzeng*].* La función de Número de cuadros fuera de lugar es la suma del número de elementos que no están ubicados en su posición objetivo respectiva, lo que se requiere es que dicha suma sea nula.

La *distancia Manhattan* es la suma de los cuadros (distancia) existentes entre la ubicación actual de cada una de las piezas y su ubicación destino, lo ideal es que esta suma sea = 0.

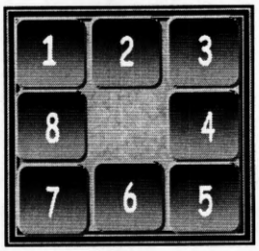


Figura 1.11 a) Configuración objetivo en un tablero 3X3.

Figura 1.11 b) Número de fichas fuera de lugar =4 y distancia Manhattan =5.

Si se considera un tablero 3X3 en el que la configuración objetivo es la indicada en la Figura 1.11 a) y la configuración inicial dada es la presentada en la Figura 1.11 b), se observa que el número de *fichas fuera de lugar* es igual a 4 y la *distancia Manhattan* es igual a 5.

Algoritmo.[Chung-Hung Tzeng]

1) Elegir todos los posibles movimientos, evitando ciclos, esto es no repetir un movimiento por el que se acaba de llegar.
2) Elegir el movimiento cuya suma sea de valor mínimo e ir a 1.

1.5 BACKTRACK

El algoritmo de Backtrack, es un algoritmo de búsqueda exhaustivo, el cual hace posible encontrar todas las configuraciones posibles del juego.

El programa termina después de encontrar una solución, en caso de no encontrar una solución adecuada, el control del programa finaliza debido a que termina de ejecutarse el ciclo principal del algoritmo.

Para verificar si se ha encontrado la solución de este juego, se debe indicar la condición siguiente:

Si se desea mover una ficha que se encuentre a la izquierda o hacia arriba del cuadro vació y todos los cuadros hacia la izquierda y hacia el primer renglón, se encuentran en su posición destino, entonces no se realiza movimiento alguno, debiendo intentarlo con otra ficha contigua al cuadro vacío.

El algoritmo inicia a partir de una configuración dada, la cual se considera como la raíz del árbol, se realiza un movimiento y este es considerado un hijo de la configuración inicial. Esta operación se realiza sucesivamente hasta que no es posible generar más hijos de la n-ésima configuración encontrada; es en este momento cuando se realiza un Backtrack colocando al padre de la n-ésima configuración y es a éste al que se le genera otro sucesor.

Lo anterior se puede observar gráficamente en la Figura 1.12

A continuación se presenta el algoritmo de BACKTRACK.

Algoritmo BACKTRACK.

- Mientras se pueda realizar un corrimiento de fichas hacia el cuadro en blanco sin que se genere una configuración ya encontrada, hacer lo siguiente:
    - Si alguno de los cuadros menores o iguales al cuadro que se desea afectar, no contiene fichas en su posición objetivo.
        - Realiza corrimiento y archivar la configuración encontrada, se establece dicha configuración como la actual.
- Regresa a la configuración padre del estado actual (BACKTRACK).

La búsqueda de la solución en el juego del 15 para una configuración dada se puede representar con el árbol que se muestra en la Figura 1.12. en el cual se siguen todas las posibilidades de movimientos hasta encontrar la configuración objetivo.

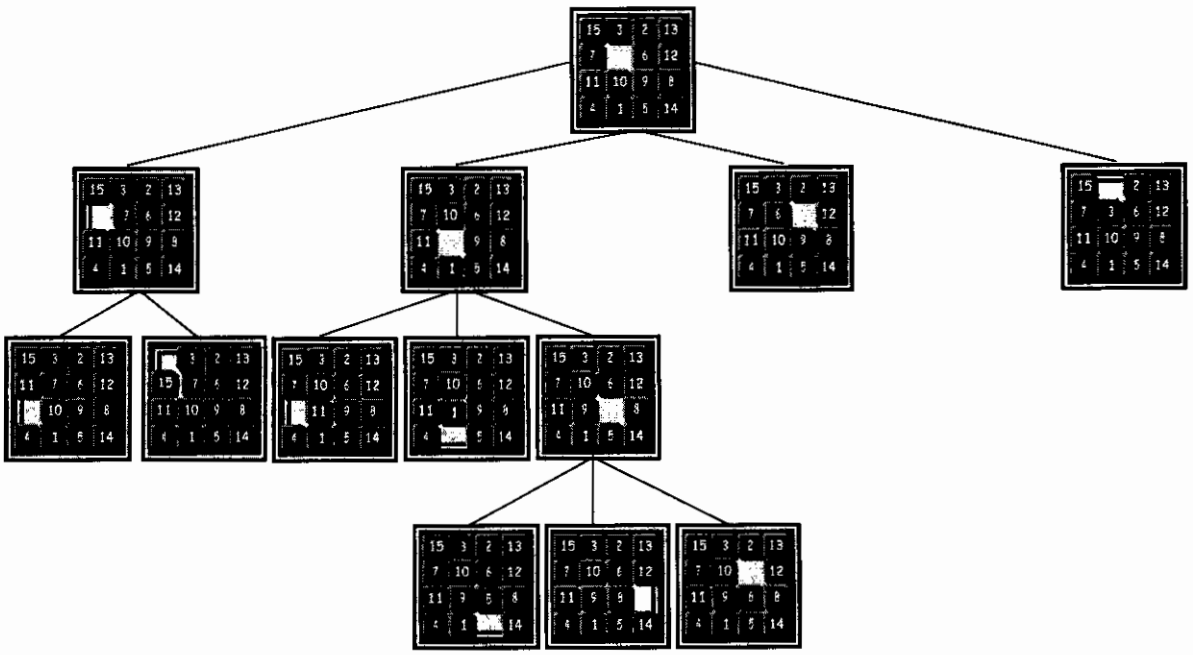


Figura 1.12 Representación del árbol de búsqueda de la solución para una configuración dada.

La programación de este algoritmo, tiene como base fundamental, la selección del movimiento de las fichas que estén alrededor del cuadro vacío, el universo de posibilidades para el movimiento de las fichas es 4 y el análisis de cada uno de ellos se inicia en orden de: izquierda, abajo, derecha, arriba. Esto se puede apreciar en la figura 1.13.

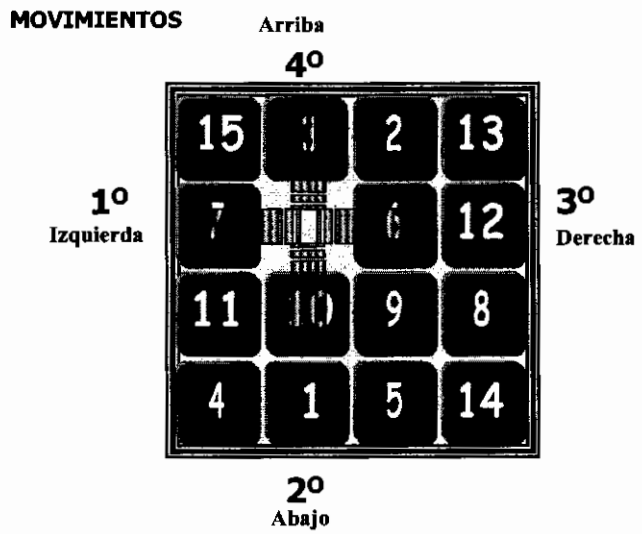


Figura 1.13 Movimientos del cuadro vacío y el orden en el que se analizan.

Al insertar en algoritmo la siguiente condición:

Si tanto la ficha a mover, como todas las fichas anteriores a esta se encuentren en su posición objetivo entonces no efectuar movimiento de esta ficha.

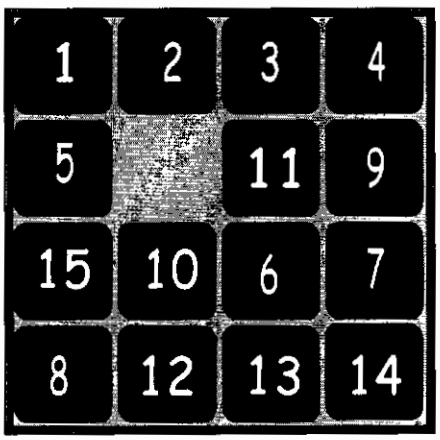


Figura 1.14. Configuración inicial del juego del 15, que al ser resuelto con el algoritmo Backtrack tardó 6 minutos con 28 segundos.

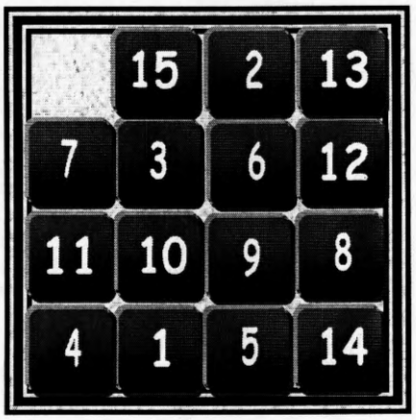


Figura 1.15. Configuración inicial, que al ser ejecutado con el algoritmo Backtrack tardó 27 horas y no se encontró la solución.

La configuración inicial mostrada en la Figura 1.16 logró alcanzar la configuración objetivo realizando tan solo 42 movimientos en 3 segundos.

Figura 1.16 Configuración inicial cuyo resultado fue alcanzado en 3 segundos y generó 42 movimientos.

Al programar este algoritmo en lenguaje Pascal, se obtienen mejores resultados ya que el utilizar solo la memoria principal elimina el tiempo de búsqueda y escritura en memoria secundaria.

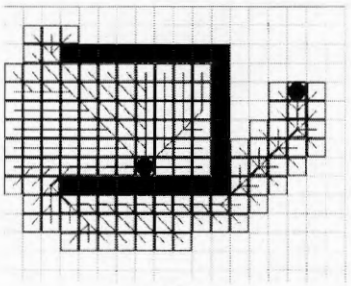
La misma configuración inicial mostrada en la Figura 1.14 duró 2 minutos con 52 segundos y generó 181,818 configuraciones diferentes.

En las primeras 6 horas, el programa solamente logró ordenar el primer renglón de la configuración de la figura 1.15.

La configuración inicial mostrada en la Figura 1.16 logró encontrar el objetivo en menos de 1 segundo.

No debe haber barreras para la libertad de preguntar. No hay sitio para el dogma en la ciencia. El científico es libre y debe ser libre para hacer cualquier pregunta, para dudar de cualquier aseveración, para buscar cualquier evidencia, para corregir cualquier error.

Julius Robert Oppenheimer (1904-1967) Físico estadounidense



VG

CAPÍTULO II

ALGORITMO A*

2.1 ALGORITMO A*

El algoritmo A*, es muy usado para llevar a cabo búsquedas del tipo BEST FIRST en un grafo dirigido, en el cual, cada nodo representa a un punto en el espacio del problema [Russel Stuart, Norving Peter, 1995]

Cada nodo contendrá además de una descripción del estado del problema que indica lo prometedor que es, un enlace a su padre y una lista de los nodos generados a partir de él.

El enlace al padre hará posible recuperar la ruta desde el objetivo una vez que el nodo objetivo fue encontrado. La lista de sucesores hará posible el que si se encuentra una ruta más óptima hacia un nodo que ya existe, se pueda propagar ésta mejora encontrada hacia sus sucesores.

Para éste fin se usaran dos listas las cuales se llamarán OPEN y CLOSED, la lista OPEN contendrá todos los nodos generados y que se les está aplicando la función heurística, pero que aún no han sido examinados, esto es, aún no se les han generado sus sucesores. Ésta lista es una cola de prioridad, en la cual los elementos con mayor peso son aquellos cuyo valor es el mejor de acuerdo a la función heurística. Así mismo la lista CLOSED contendrá los nodos que están siendo examinados.

Se dice que un algoritmo es *completo*, si éste logra encontrar una solución, siempre y cuando exista [Pear Judea, 1984]. Un algoritmo es a*dmisible* si garantiza encontrar una solución óptima, siempre y cuando exista. Es necesario definir el término *dominio*: un algoritmo A_1 se dice que *domina* A_2, si cualquier nodo expandido por A_1, es también expandido por A_2. De manera similar *domina* estrictamente a A_2 si A_1 *domina* a A_2 y A_2 no *domina* a A_1. Un algoritmo es *óptimo* dentro de una clase de algoritmos, si éste domina todos los miembros de la clase

Las búsquedas demasiado austeras minimizan el costo estimado hacia el objetivo *h(n)* y acortan considerablemente el costo de la búsqueda. Desgraciadamente éstas no son ni óptimas ni completas. La búsqueda de costos uniformes, por el contrario, minimiza el costo de la ruta, *g(n)*, la cual es óptima y completa, pero puede ser muy ineficiente. Un verdadero avance se tiene cuando se combinan esas dos estrategias ya que se obtienen las ventajas de ambos [Pear Judea, 1984]. Afortunadamente, se puede realizar, combinando las dos funciones de evaluación.

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

F(n) es el costo estimado de la solución más económica que pasa a través de *n*;

g(n) proporciona el costo de la ruta del nodo inicial al nodo *n;* y *h(n)* es el costo estimado de la ruta más económica del nodo *n* al nodo objetivo.

Es necesaria una función heurística que estime el valor de cada uno de los nodos que se generan, esto hará que el algoritmo busque la mejor de las rutas. A ésta función se le llamará *f'* para indicar que es una aproximación a la función *f* que genera la verdadera evaluación del nodo. Es conveniente definir ésta función como la suma de dos componentes llamados *g* y *h'*. La función *g* es una medida del costo de ir del nodo inicial al nodo actual. No es un estimado, es la suma del costo de aplicar cada una de las reglas a lo largo de la ruta hacia éste nodo, y la función *h'* es un estimado del costo adicional para llegar del nodo actual al nodo objetivo.

La función de combinación *f'* representa un estimado del estado inicial al estado objetivo a lo largo de la ruta que generó el nodo actual. Si el nodo generó más de una ruta, el algoritmo registrará la mejor de ellas. Debido a que *g* y *h'* deben ser sumadas, es importante que *h'* sea una medida del costo de llegar a la solución (esto es, que los nodos considerados como mejores sean los de menor valor y los nodos que no interesan sean los de mayor valor). La función *g* debe ser no negativa, de lo contrario, las rutas que contiene ciclos en el grafo aparecerán como las mejores entre más largas sean.

La operación actual del algoritmo es muy simple, se procede por pasos, expandiendo un nodo en cada paso hasta que éste genera un nodo que corresponde a un estado objetivo. En cada paso se elige el mejor de los nodos que esta siendo generado pero no expandido, éste genera los sucesores de los nodos escogidos, les aplica la función heurística y los adiciona a la lista de nodos en OPEN, después de verificar si cualquiera de ellos ya fue generado con anterioridad. Realizando ésta revisión, se garantiza que cada uno de los nodos aparece solo una vez en el grafo aunque muchos nodos pueden apuntar a éste como un sucesor.

2.2 COMPLEJIDAD DE A*.

A* es el más completo, óptimo y eficiente de todos los algoritmos de su tipo. Desafortunadamente, esto no significa que A* sea la respuesta a todas las necesidades de búsqueda, en muchos de los problemas, el número de nodos dentro del espacio de búsqueda del objetivo, es exponencial en cuanto al tamaño de su solución. El crecimiento exponencial ocurrirá a menos que el error en la función heurística no crezca más rápido que el algoritmo del actual costo de la ruta [Pear Judea, 1984]. En notación matemática la condición para crecimiento sub exponencial es

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

Donde *h*(n)* es el costo verdadero de obtener el objetivo a partir de *n*, para casi todas las heurísticas en uso práctico, el error es por lo menos proporcional al costo de la ruta y el resultado exponencial crece eventualmente sobrepasando la capacidad de cualquier computadora

En el estudio de algoritmos, es conveniente distinguir entre procedimientos no determinísticos y determinísticos. Un algoritmo pasa de un estado del problema a otro durante su ejecución. Un algoritmo determinístico puede moverse únicamente a un nuevo estado a la vez, mientras que un algoritmo no determinístico puede moverse a varios nuevos estados a la vez. Esto es, un algoritmo no determinístico puede explorar varias posibilidades simultáneamente.

La clase de problemas para los cuales existe un algoritmo determinístico cuya complejidad es polinomial, es llamado P. La clase de problemas para los cuales existe un algoritmo no determinístico cuya complejidad es polinomial es llamado NP [Roberts Fred S. 1984].

Claramente se puede apreciar que cualquier problema en P se encuentra en NP. Hasta la fecha, nadie a descubierto un problema en NP que pueda demostrarse que no está en P. Por lo tanto hay muchos problemas que se saben que están en NP los cuales pueden o no estar en P.

2.3 ALGORITMO A* (1ª versión)

Para atraer la atención del lector hacia un algoritmo la mejor forma es hacer que a simple vista sea el más sencillo y claro, esto es, que el lector no lo deteste por la cantidad de información y pasos que se le presentan, en el transcurso de la investigación se encontraron versiones de éste algoritmo, de las cuales y únicamente para fines didácticos se presentan únicamente tres.

Cabe señalar que en esencia todos los algoritmos llevan a los mismos resultados y todos los algoritmos son A*, lo que se quiere decir es que cada autor lo presenta más o menos condensado y lo trata de explicar a su manera.

La versión del algoritmo A* que se presenta en el siguiente capítulo es la que se considera más conveniente para fines didácticos.

Algoritmo A* [Rich Elaine, 1983]

1.- Inicia con la variable OPEN conteniendo únicamente el nodo inicial. Hacer g=0, y h al que corresponda y f'=h'+0 o h', inicializar CLOSED como una lista vacía.

2.- Hasta que el nodo objetivo sea encontrado, repetir el siguiente procedimiento:

- Si no hay nodos en OPEN, reportar una falla.
- Escoger el nodo de OPEN que tenga el valor de f' más bajo.
- Llamar a éste nodo BESTNODE.

- Eliminar éste nodo de OPEN. Adicionar en CLOSED. Verificar si BESTNODE es un nodo objetivo.
- Si es así salir y reportar una solución.
- En caso contrario, generar los sucesores de BESTNODE, pero no hacer que BESTNODE apunte a ellos.
- Para cada uno de estos sucesores, hacer los siguiente:

2.1.- Hacer que SUCCESSOR apunte hacia atrás, esto es, hacer BESNODE, ese enlace, hará posible el recuperar la ruta una vez que se encuentre la solución.

2.2.- Hacer g(SUCCESSOR)=g(BESTNODE)+costo de llegar de BESTNODE a SUCCESSOR.

2.3.- Verificar si SUCCESSOR se encuentra en la lista OPEN (esto es, si ya fue generado aunque aún no halla sido procesado). Si es así llamar a éste nodo OLD. Como éste nodo existe en el grafo, se puede desechar a SUCCESSOR y adicionar OLD a la lista de sucesores de BESTNODE. Ahora, es necesario decidir cuales enlaces padres de OLD deben ser cambiados para que apunten a BESTNODE. Esto debe de hacerse, si la ruta recién encontrada hacia SUCCESSOR, es más pequeña en cuanto a costo que la actual mejor ruta a OLD (ya que SUCCESSOR y OLD son realmente el mismo nodo). De ésta manera se determina si es más económico llegar a OLD vía su actual padre o a SUCCESSOR vía BESTNODE, por medio de la comparación de sus valores de *g*. Si OLD es de menor costo, entonces no se necesita hacer nada. Si SUCCESSOR es de menor costo, entonces se inicializa los enlaces padres de OLD, para que

ahora apunten a BESTNODE, registrando la nueva ruta más económica en g(OLD), y actualizar f'(OLD).

2.4.- Si SUCCESSOR no está en OPEN, verificar si se encuentra en CLOSED, si es así, llamar al nodo en CLOSED, OLD y adicionar OLD a la lista de sucesores de BESTNODE, verificar si la nueva ruta o la anterior ruta es mejor como en el paso 2.3 y direccionar los valores del enlace padre, g y f' apropiadamente. Si se ha encontrado una mejor ruta hacia OLD, se debe propagar ésta mejora a los sucesores de OLD. OLD apunta a sus sucesores, cada sucesor en turno, apunta a sus sucesores y así sucesivamente hasta que cada ramificación termina con un nodo que se encuentra en OPEN o que no tiene sucesores, así que, para propagar el nuevo costo hacia abajo, hacer un depth-first, transversal del árbol comenzando en OLD, cambiando cada uno de los valores de los nodos g (y también sus valores de f'), terminando cada ramificación cuando se alcance un nodo sin sucesores o un nodo para el cual a sido encontrado una mejor ruta o su equivalente, ésta condición es fácil de verificar. Cada enlace padre del nodo, apunta hacia atrás, a su mejor padre conocido. Como se propaga hacia abajo a un nodo, verificar si su padre apunta al nodo del cual se viene. Si es así, se continúa con la propagación. Si no es así, entonces sus actuales valores de g reflejan la mejor ruta, de la cual ésta forma parte. La propagación puede detenerse aquí, pero es posible que con el nuevo valor de g, comience a propagarse hacia abajo, la ruta que se ésta siguiendo puede llegar a ser mejor que la ruta a través del padre actual. Así que se debe de comparar las dos. Si la ruta a través del padre actual sigue siendo mejor, detener la propagación. Si la ruta que se

esta propagando es ahora la mejor, inicializar el padre y continúe con la propagación.

2.5.- Si SUCCESSOR no esta ni en OPEN ni en CLOSED, entonces ponerlo en OPEN y adicionarlo a la lista de sucesores de BESTNODE. Realizar la operación:

$$f'(SUCCESSOR)=g(SUCCESSOR)+h'(SUCCESSOR).$$

Se puede apreciar que ésta descripción del algoritmo es muy confusa debido a la gran cantidad de información y detalles que se tratan de transmitir, por tal motivo se presenta otra versión del algoritmo la cual es más condensada.

2.4 ALGORITMO A* (2ª VERSION)

1.- Colocar el nodo inicial *s* en *OPEN*.

2.- Si *OPEN* esta vacío salir marcando ERROR.

3.- Eliminar de *OPEN* y poner en *CLOSED* el nodo *n* para el cual *f* es mínimo.

4.- Si *n* es un nodo objetivo, terminar, reportando la solución obtenida y desplegando como ruta los nodos recorridos desde *n* hasta s.

5.- En caso contrario, expandir *n*, generar todos sus sucesores y relacionar a ellos los apuntadores hacia *n*

para cualquier sucesor *n'* de *n*, hacer lo siguiente:

a) Si *n'* no existe en *OPEN* o *CLOSED* calcule *h(n')* (estimado del costo de la mejor ruta de *n'* a alguno de los nodos objetivos), calcule f(n')=g(n')+h(n') donde g(n')=g(n)+c(n, n') y g(s)=0.
b) Si *n'* existe en *OPEN* o *CLOSED*, direccionar sus apuntadores a lo largo de la ruta uniendo los *g(n')* más pequeños.
c) Si *n'* Requiere ajustar sus apuntadores y fue encontrado en *CLOSED*, volver a ponerlo en *OPEN*.

6.- Ir al paso 2.

El algoritmo A* es fácil en primera instancia, pero involucra gran dificultad y un concienzudo análisis, de los elementos que en él intervienen. Su heurística es muy importante ya que de ella depende que el algoritmo sea muy eficiente o sumamente lento, una de sus grandes ventajas es que permite analizar varias rutas de solución a la vez e ir eliminando rutas en cuanto se detecta alguna más optima, sin dejar de tener en cuenta las otras rutas ya que en cualquier momento

la ruta actual o que en ese momento es la mejor, se pude convertir en una mala opción.

El algoritmo del Backtrack encuentra la solución a un problema dado solo que debido a la gran cantidad de rutas que genera la solución puede ser encontrada en las primeras de ellas que se seleccione o bien en las últimas de ellas lo que sería el peor de los casos, pero si se tiene en cuenta que la solución puede estar después de recorrer millones de rutas, no es conveniente tener que esperar tanto tiempo.

Se pueden hacer varias observaciones al algoritmo.

Con respecto a la función *g*, se procede a escoger el siguiente nodo a expandir, basados no solo en que tan prometedor se ve el nodo (como medido por h'), sino también basados en que tan aceptable es la ruta.

Debido a que se incorpora *g* dentro de *f'*, no siempre se escogerá como el siguiente nodo a expandir el nodo que aparece como más cercano al objetivo, esto es muy usual si se considera la ruta que encontró.

Si, de otra manera, solo se desea obtener una solución, cualquiera que ésta sea, se puede definir a g como igual a cero y así siempre escoger al nodo que se vea más próximo al objetivo.

Si se desea encontrar una ruta con menor número de pasos, entonces se mantiene el costo de ir de un nodo a su sucesor como una constante, usualmente 1.

Si lo que se busca es encontrar la ruta de menor costo y algunos de los operadores tienen más valor que otros, entonces se inicializa el costo de ir de un nodo a otro para que reflejen esos costos.

El algoritmo A* puede ser usado en la forma que más interese, ya sea encontrar una ruta de costo mínimo o simplemente una ruta que sea lo más rápida posible.

El valor de h', que es el estimado de h, es la distancia del nodo al objetivo, si h' es un estimado perfecto de h, entonces A* converge inmediatamente al objetivo sin búsqueda, la mejor de las h' es la más cercana que se obtenga. Si de otra manera h' siempre es 0, la búsqueda será controlada por g. Si g siempre es 0 la estrategia de búsqueda será aleatoria, si siempre es 1 la estrategia será BREADTH-FIRST, todos los nodos en el mismo nivel, tendrán los valores de g más pequeños y por lo tanto los valores de f más pequeños que los nodos del siguiente nivel. Es importante preguntarse ¿Qué sucede si h' no es perfecta ni cero?, ¿Se puede decir algo interesante acerca del comportamiento de la búsqueda?, la respuesta es si, si se desea garantizar que h' nunca sobre estime a h, en ese caso el algoritmo A* garantiza encontrar una ruta optima al objetivo, si es que existe.

2.5 EJEMPLOS.

Si se considera la situación presentada en la figura 2.1 a), y se asume que el costo de todos los arcos es 1, inicialmente todos los nodos excepto A están en OPEN. Para cada nodo f' se indica como la suma de h' y g. En éste ejemplo el nodo B tiene el valor más pequeño de f' que es 4, de tal forma que éste es el que se expande primero, supóngase que éste solo tiene el sucesor E, el cual también parece encontrarse a tres movimientos del objetivo. Ahora f'(E) es 5 igual que f'(C), suponer que se decide por seguir la misma ruta que hasta ahora, entonces se expande E a sus respectivos hijos, suponga que éste también tiene un único sucesor F. También tiene 3 movimientos para el objetivo, claramente se están realizando movimientos pero sin ningún progreso. Pero f'(F) =6, el cual es mayor que f'(C) así que se expande C al siguiente. De ésta manera, se observa que por subestimar a h(D) se ha realizado esfuerzo en vano, pero finalmente se descubre

que B fue más lejos de lo que se esperaba, es necesario regresar y tomar otra ruta.

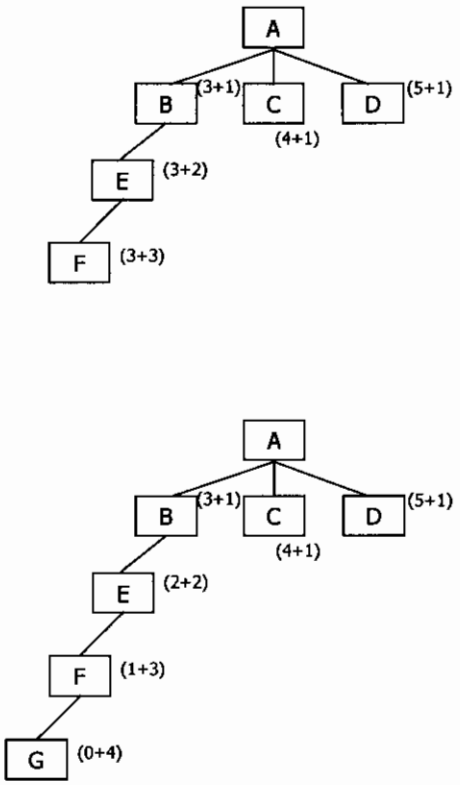


Fig. 2.1 a) y b) Búsqueda de una solución asumiendo que esta se encuentra por la ruta del nodo D

Ahora se considera la situación presentada en la figura 2.1 b), nuevamente se expande B en el primer paso, en el segundo paso E, en el siguiente paso F y finalmente se genera G para tener una ruta solución de longitud 4, pero suponga que hay una ruta directa de D a una solución, dando una ruta de longitud 2, nunca se encontrará. Por sobrestimar h'(D) se provoca que D se vea tan mal que se tenga que buscar otra ruta, que será equivocada por no expandir D, en general, si h' puede sobrestimar a h, no se puede garantizar el encontrar la ruta

solución de menor costo a menos que se expanda el grafo entero, hasta que todas las rutas sean más grandes que el de la mejor solución.

El algoritmo en su forma más general se aplica a grafos, éste puede ser simplificado para que se aplique a arboles, haciendo que no verifique si un nuevo nodo está en OPEN o en CLOSED, lo que hace más rápido el generar nodos, pero puede llevar a que la búsqueda empiece muchas veces si los nodos se duplican.

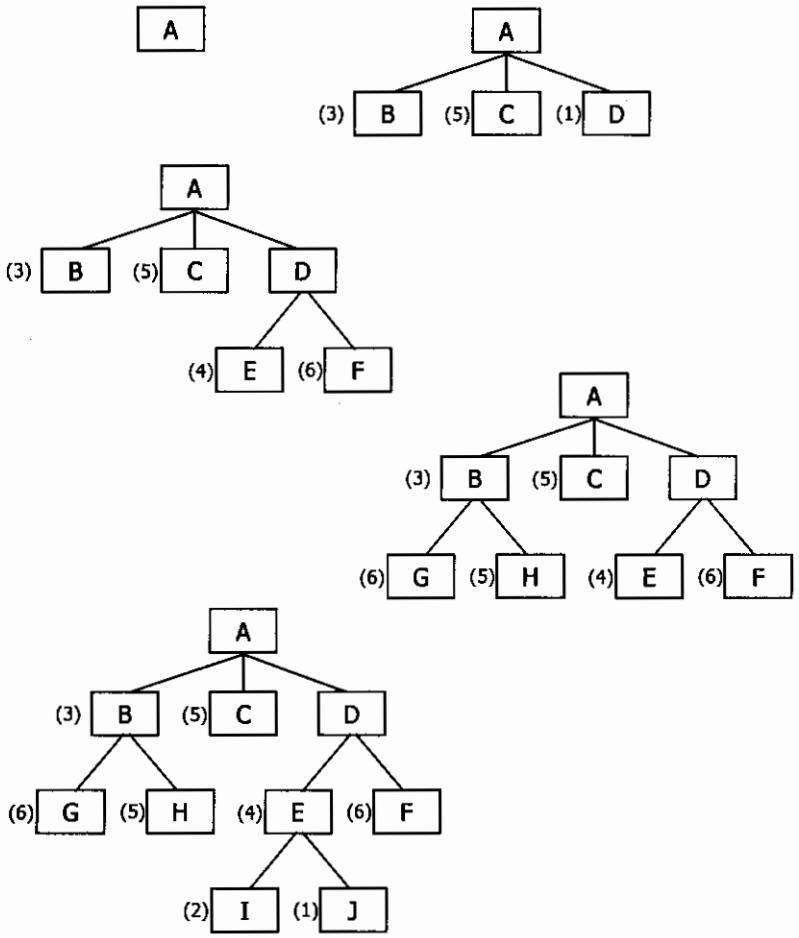


Fig. 2.2 Representación del desarrollo de la búsqueda de un objetivo.

En la figura 2.3 se presenta el árbol de búsqueda que se genera para resolver el juego del 8 que es una derivación del juego del 15, solo que éste esta formado por 8 fichas numeradas y un espacio en blanco, para éste caso la configuración objetivo es ordenar los números ascendentemente de manera circular con el vacío en el centro. Los valores sombreados representan el costo de cada uno de los nodos.

La configuración inicial tiene un valor de *f=4*, se debe considerar que el nodo que tiene el valor de *f* más bajo es uno de los candidatos a estar en la ruta óptima. El valor de *g* es constante y se considera como 1 debido a que el costo de ir de un nodo a otro solo implica mover una ficha una posición. El valor de f para cada nodo, en la Figura 2.3 se representa sombreado y a un lado de cada uno de ellos.

El valor de h se calcula, para el caso de éste ejemplo, con la función de *Número de fichas fuera de lugar*, observe que la configuración inicial tiene fuera de su posición objetivo 4 fichas y éstas son las marcadas con los números 1, 2, 6 y 8, como no se ha recorrido ninguna distancia previa el valor de *g=0*, por lo que

$$f = g + h = 0 + 4$$

para primer movimiento se tienen 3 posibilidades, las cuales son mover el 7 a la derecha, lo que genera una configuración con 5 fichas fuera de su posición objetivo (1, 2, 6, 8 y 7) y como ya se realizó el primer movimiento g=1, por lo que

$$f = 1 + 5$$

El segundo posible movimiento es deslizar el 6 hacia abajo, lo que produce una configuración con 3 fichas fuera de su posición objetivo (1, 2 y 8) y como se ha producido un movimiento el valor de f es:

$$f = 1 + 3 = 4$$

El tercer movimiento es deslizar el 5 a la izquierda de su posición inicial, lo que genera 5 fichas fuera de su posición objetivo (1, 2, 6, 8 y 5) más la distancia recorrida, por lo que:

$$f = 5 + 1 = 6$$

Y así sucesivamente hasta encontrar el nodo objetivo

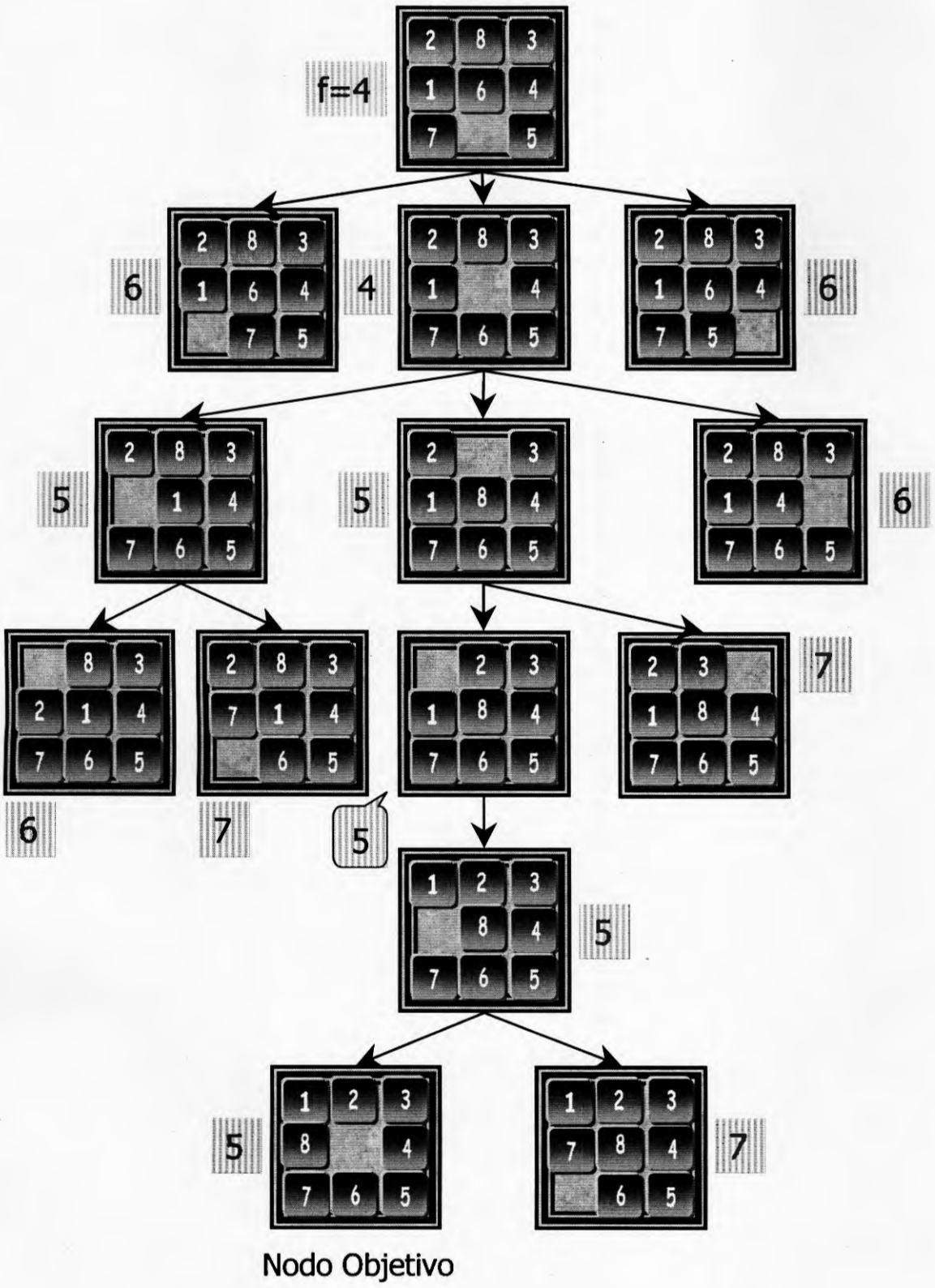


Figura 2.3 Arbol de búsqueda generado para resolver el juego del 8

2.7 OTROS ALGORITMOS.

Los algoritmos de búsqueda Depth First y breadth First [Shoham Yoav. 1994] caen dentro de la categoría de las búsquedas blind[1], ya que el orden de búsqueda en ambos es determinado únicamente por la estructura del grafo, algunas veces existe información adicional que auxilia en el proceso de búsqueda.

El algoritmo de búsqueda Best First, está diseñado para usar esa información y entonces buscar en el grafo. La búsqueda estima la "mejor" de las rutas recientemente exploradas, y entonces expande la mejor de esas rutas, ésta medida de "mejor" es llamada el valor de la heurística de la ruta; se toman esos valores numéricos, el valor más bajo es el considerado como el mejor. Pero estos algoritmos no son considerados como los mejores para la obtención de la ruta más corta.

El algoritmo Best First no encuentra la mejor solución en casos como el que se estudia en éste trabajo. Después de todo la idea principal del algoritmo de búsqueda Best First, es encontrar alguna solución tan rápido como sea posible encontrando cualquier nodo cuya distancia al nodo objetivo sea 0. De ninguna manera intenta encontrar el nodo objetivo que éste a la mínima profundidad del árbol de búsqueda. Si se desea hacer esto, se necesita modificar el algoritmo de búsqueda de tal forma que se refleje el interés en encontrar un objetivo no importa lo profundo que se encuentre en el árbol, esto si lo contempla el algoritmo A*

Algoritmo BEST-FIRST.

1. *L* es la lista de el o los nodos iniciales en el problema.

[1] Éste nombre se le da a los algoritmos que realizan búsquedas exhaustivas y se prefiere dejar éste nombre y no la respectiva traducción ya que su significado es "ciego", por lo que se les llamaría "búsquedas ciegas".

2. Seleccionar el nodo *n* de la lista *L*, el cual se supone que es el más cercano al objetivo. Si *L* está vacío, entonces se encontró un error.

3. Si n es el nodo objetivo, detener la ejecución del programa y presentar a *n* y la ruta correspondiente del nodo inicial a *n*.

4. En otro caso, eliminar *n* de *L* y adicionar a *L* todos los hijos de *n*, etiquetando cada uno con la ruta correspondiente a partir del nodo inicial. Regresar al paso 2.

2.8 BUSQUEDA $\alpha - \beta$.

Éste tipo de búsqueda es para problemas donde interactuan dos adversarios [Gingberg Matt, 1993]

Algunos algoritmos de búsqueda visitan todos los nodos del árbol del problema, ésta búsqueda exhaustiva es a veces innecesaria ya que ciertos nodos pueden ser ignorados, específicamente ciertos sub árboles que pueden ser "podados" sin que esto afecte el resultado. Considere el árbol de la Figura 2.4, tan pronto como el algoritmo descubre que el valor del nodo j es 9, pueden ser ignorados los nodos k y l por la siguiente razón: Debido a que b es un nodo min, su valor se encuentra al menos entre 7 (que es el valor del nodo d) y el valor del nodo e. Como el nodo e es un nodo max, su valor será al menos el valor del nodo j, esto es 9, por lo tanto el valor del nodo b será 7, aún sin tener en cuenta los valores de los nodos k y l, en una forma similar puede obtenerse los valores de los nodos g, o, p, q y r sin que sean operados.

Los nombres de min y max que se encuentran a la derecha del árbol representan el turno de tirada de cada uno de los jugadores [Gingberg Matt. 1993].

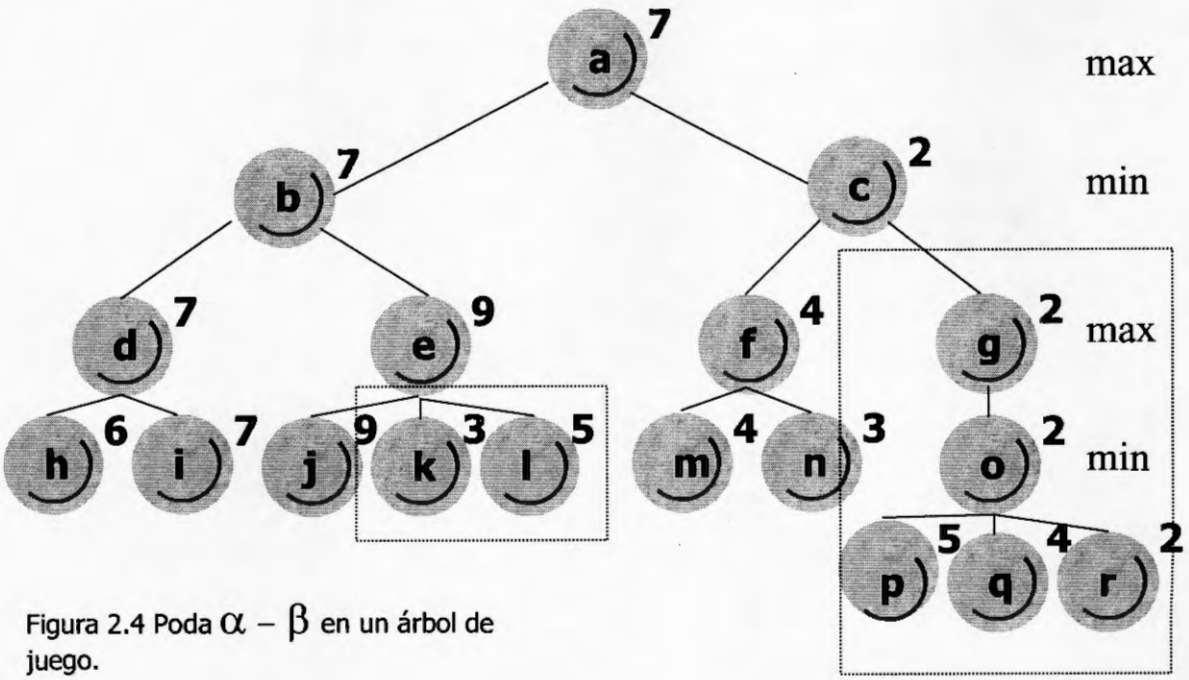
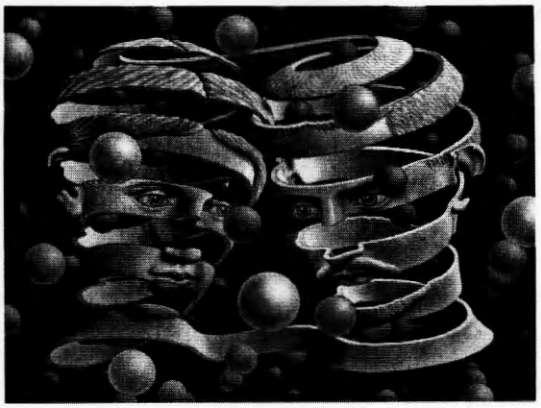


Figura 2.4 Poda $\alpha - \beta$ en un árbol de juego.

Ésta poda es realizada por el algoritmo de búsqueda $\alpha - \beta$, el nombre de $\alpha - \beta$ se refiere a los números que son asignados a cada uno de los nodos durante la búsqueda, esos números de una manera afectan la búsqueda por medio de la ayuda que proporciona la poda de ciertos sub árboles y también por la actualización de sí mismos como resultado de la búsqueda.

-Él navegante italiano ha llegado al nuevo mundo
-¿Y que le parecieron los aborígenes?
-¡Muy hospitalarios!

Conversación en clave entre Enrique Fermi y James Conant (1942)



CAPÍTULO III

SOKOBAN

3.1 REGLAS DEL JUEGO.

Sokoban es un juego de estrategia inventado en Japón en el cual el jugador es un almacenista que debe empujar los objetos a sus destinos correctos en un ambiente cerrado y con obstáculos.

Existe un juego que en esencia es lo mismo sólo que inventado por otro autor y con el nombre del Boxworld (Figura 3.1 y 3.2); en éste trabajo de investigación se desarrollarán algunos de sus niveles.

La regla principal en el Sokoban es mover los objetos evitando que el almacenista sea acorralado, es decir, que los corredores del "almacén" sean

bloqueados. Debido a que los objetos sólo pueden ser empujados y en ningún momento jalados además de no poder empujar dos o más objetos juntos, el almacenista o Sokoban en Japonés debe evitar realizar movimientos que lleven a bloquear los corredores antes de que coloque los objetos en sus destinos.

Imagínese que está en medio de un laberinto tridimensional compuesto por celdas cuadradas las cuales pueden o no estar llenas de cajas u objetos simulando mercancía.

El jugador o participante es representado por una figura que se puede mover hacia arriba, abajo izquierda o derecha una celda cada vez, estos movimientos son llamados desplazamientos.

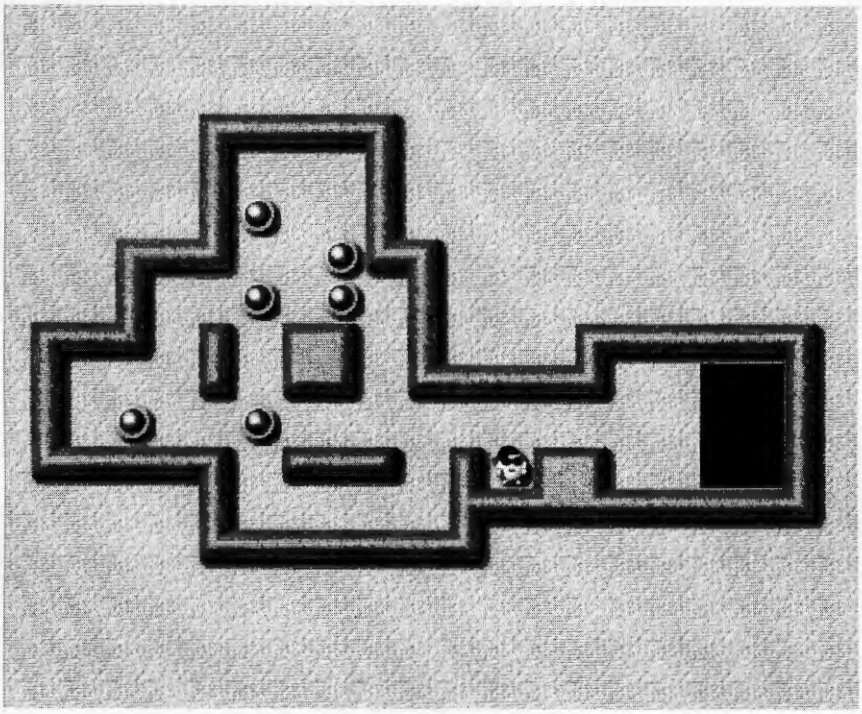


Figura 3.1 Presentación de un nivel del Sokoban.

Una o alguna de las celdas vacías (sin paredes) contiene una caja que puede moverse a una celda adyacente, colocándose al lado de la caja y luego moviéndose en la dirección en la que se encuentra la caja, esto es empujando la

caja, no hay otra forma de mover la caja si no es empujandola, esto quiere decir que si se empuja la caja hacia una esquina, no habrá manera de sacarla de ahí.

Una o algunas de las celdas vacías están marcadas como objetivo (el mismo número de cajas de mercancías existentes en nuestro laberinto). La tarea a realizar es llevar las cajas a las celdas objetivos mediante una serie de empujes a cada una de ellas, buscando obtener el menor número de empujes y desplazamientos.

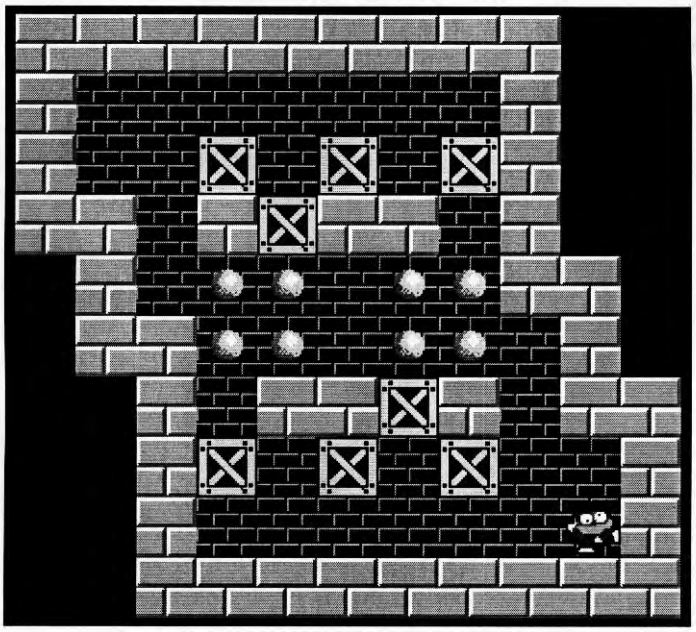


Figura 3.2. Versión del Sokoban llamada BoxWorld.

La solución puede ser realizada por un programa que lee la descripción de un laberinto desde un archivo, dicha descripción contiene dos enteros i, j que representan el número de filas y columnas del laberinto, enseguida n cadenas, donde cada carácter de la cadena, describe el contenido de cada celda del laberinto.

3.2 Nomenclatura.

El caractér "*#*" representa una pared del laberinto y una celda vacía es representada por un " ". El punto de arranque se indica con una "*A*", esto es la posición inicial del Sokoban. La posición inicial de cada caja de mercancía se representa por el número 1 y las celdas objetivo por la letra "*X*"; si un objeto se encuentra ubicado en una celda objetivo, éste será representado por "*Y*".

La solución puede ser representada por la secuencia de salida en forma de una cadena de caracteres A, B, I, D, a, b, i, d donde las letras mayúsculas indican empujes y las minúsculas desplazamientos hacia arriba, abajo, izquierda y derecha correspondientemente.

Se debe tener en cuenta las siguientes observaciones para la solución de estos laberintos:

- No se permiten movimientos de cajas u objetos que coloquen dos de ellos adyacentes, a menos que uno de los objetos queden con tres lados libres.

- Ningún objeto podrá colocarse en una esquina a menos que esta sea un objetivo.

- Buscar el objeto en turno y el Sokoban, implica buscar la ruta adecuada moviendo los objetos que sean necesarios para llegar a la posición correcta del objeto deseado y empujarlo.

- Ruta para la llegada.

- Durante el retroceso para acomodarse alrededor de un objeto tratar de mover el menor número de objetos posibles.

- Marcar las posiciones no válidas, esto es todas aquellas que jamás deberán ser ocupadas por un objeto.

- Una vez encontrada una ruta que lleva al objetivo tratar de utilizarla con los demás objetos a menos que exista un camino más corto.

- Nunca dejar espacios de un cuadro cerrados o bloqueados por dos o más objetos que no puedan recorrerse.

- Para mover un objeto que esté pegado a paredes recorrerlo si es necesario en sentido opuesto a su objetivo hasta un punto donde pueda ser empujado en la dirección requerida.

- Buscar rutas alternas para llegar al lado conveniente del objeto con el fin de poder empujarlo.

Para la solución de éste problema se puede hacer uso de algoritmos tales como Backtrack y A*.

Como ya se vio en el capítulo I titulado "Juego del 15", Backtrack es un algoritmo que hace una búsqueda exhaustiva dentro del universo de movimientos de los objetos, lo cual representa el mismo problema de solución en el Sokoban.

Las heurísticas que se contemplan dentro del algoritmo A* representan una gran ventaja para la disminución del universo de movimientos que debe realizar el Sokoban en la búsqueda de la solución.

En la Figura 3.3 se puede apreciar un nivel del Sokoban y su representación computacional.

Los caracteres de dicha representación configuración se introducen en un string en el algoritmo.

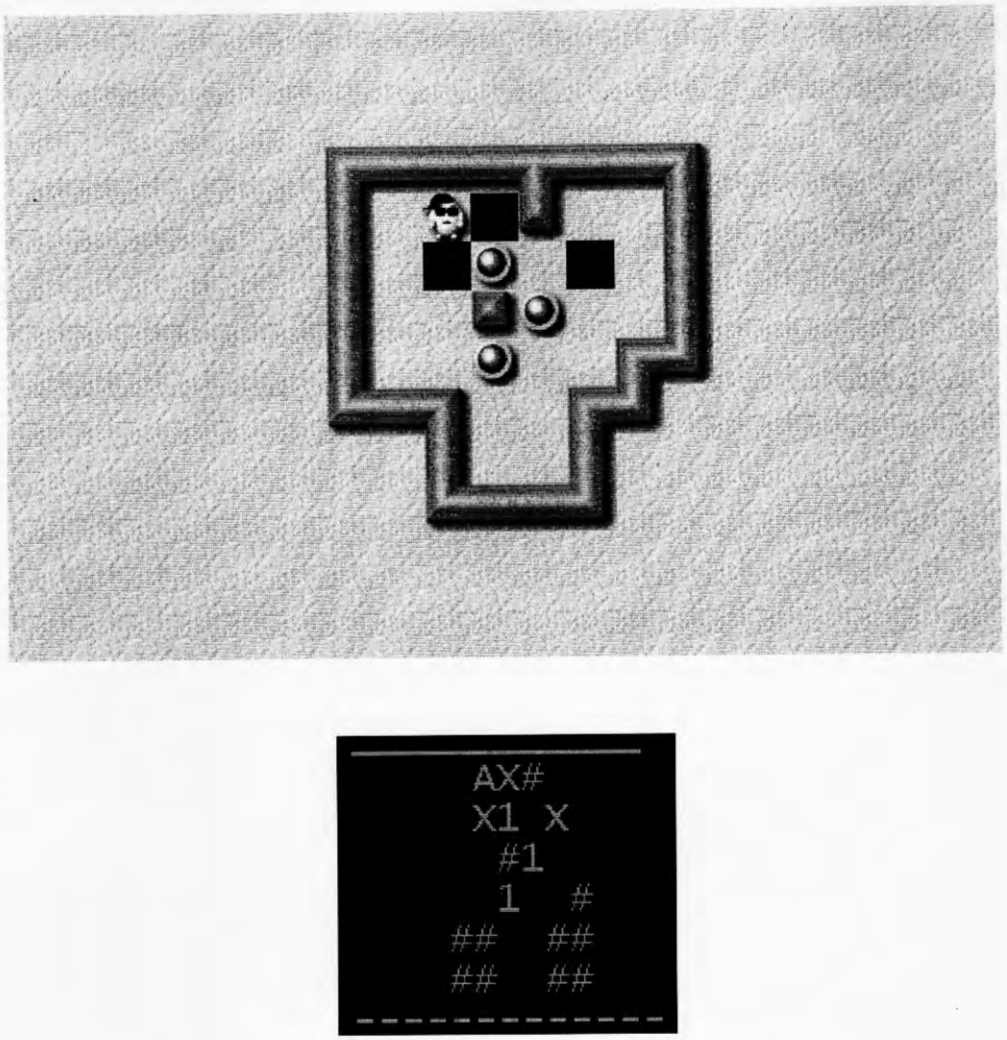
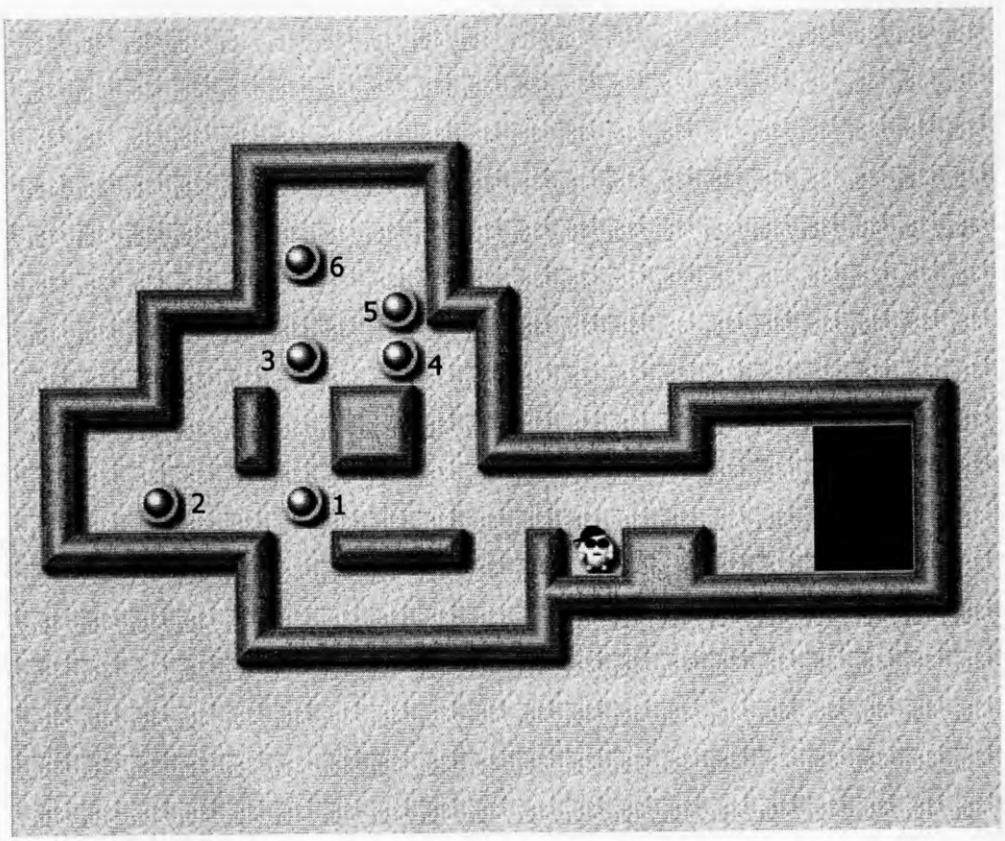


Figura. 3.3 Representación computacional del Sokoban Nivel 14.

Considérese el siguiente nivel del Sokoban, el cual tiene sus objetos numerados del 1 al 6:



Colocar el objeto 1 en una posición objetivo se logra mediante la siguiente secuencia de salida:

AiiiaaaIAiiBiibbbdDDDDDDDDDDDD

Lo que llevará al resultado presentado en la Figura 3.4

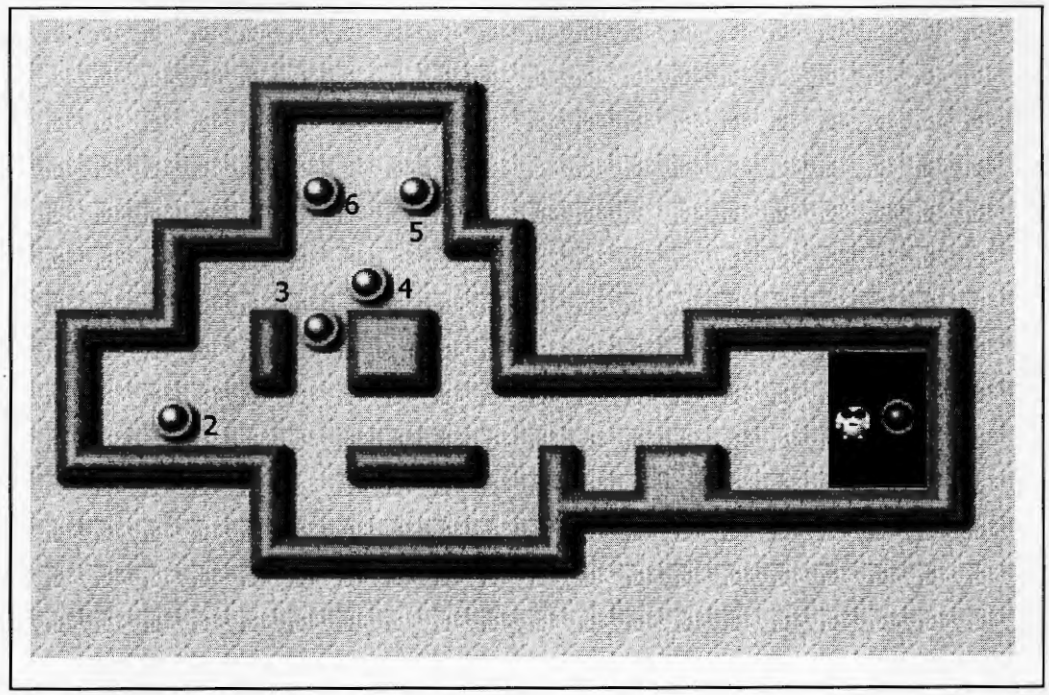


Figura. 3.4 Nivel 1 del Sokoban después de haber acomodado el objeto marcado con el número 1.

Se debe tener en cuenta que el objeto 1 pudo haber sido colocado en la columna anterior a la que se presenta como estado final de la Figura 3.4 y también se considera como posición objetivo.

Movimiento del objeto 2.

iiiiiiiiiiiiiaiibDDDDDDDDDDDDDDDbdAiaD

Movimiento del objeto 3

iiiiiiiaaaiaiibBBaaiibbbdDDDDDDDDDDDbdA

Movimiento del objeto 4

iiiiiiiiaaaaiIaiBBBaaaiibbbdDDDDDDDDDDDD

Quedando en éste punto en una situación como la que se presenta en la Figura 3.5

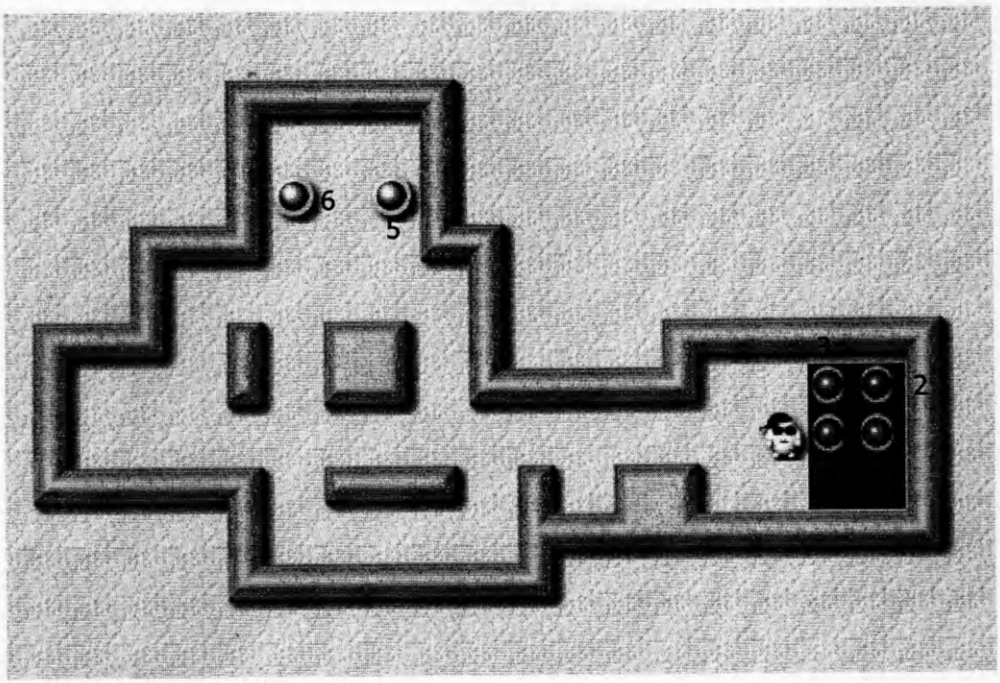


Figura 3.5 Vista del Sokoban después de haber llevado hasta su objetivo los 4 primeros objetos.

Los movimientos que se acaban de presentar, representan una forma ideal, esto es, sin considerar que alguno de los objetos, en su trayectoria hasta el objetivo puede ser obstaculizado o bien ser obstáculo de algún otro objeto.

Por lo anterior se debe de considerar que los movimientos de cualquiera de ellos implican mover más de un objeto incluyéndolo a él mismo.

En la Figura 3.6 (a), (b), (c) y (d), se puede observar que es imposible llevar directamente hasta los objetivos, cualquiera de los objetos, sin tener que mover dos de ellos durante el trayecto.

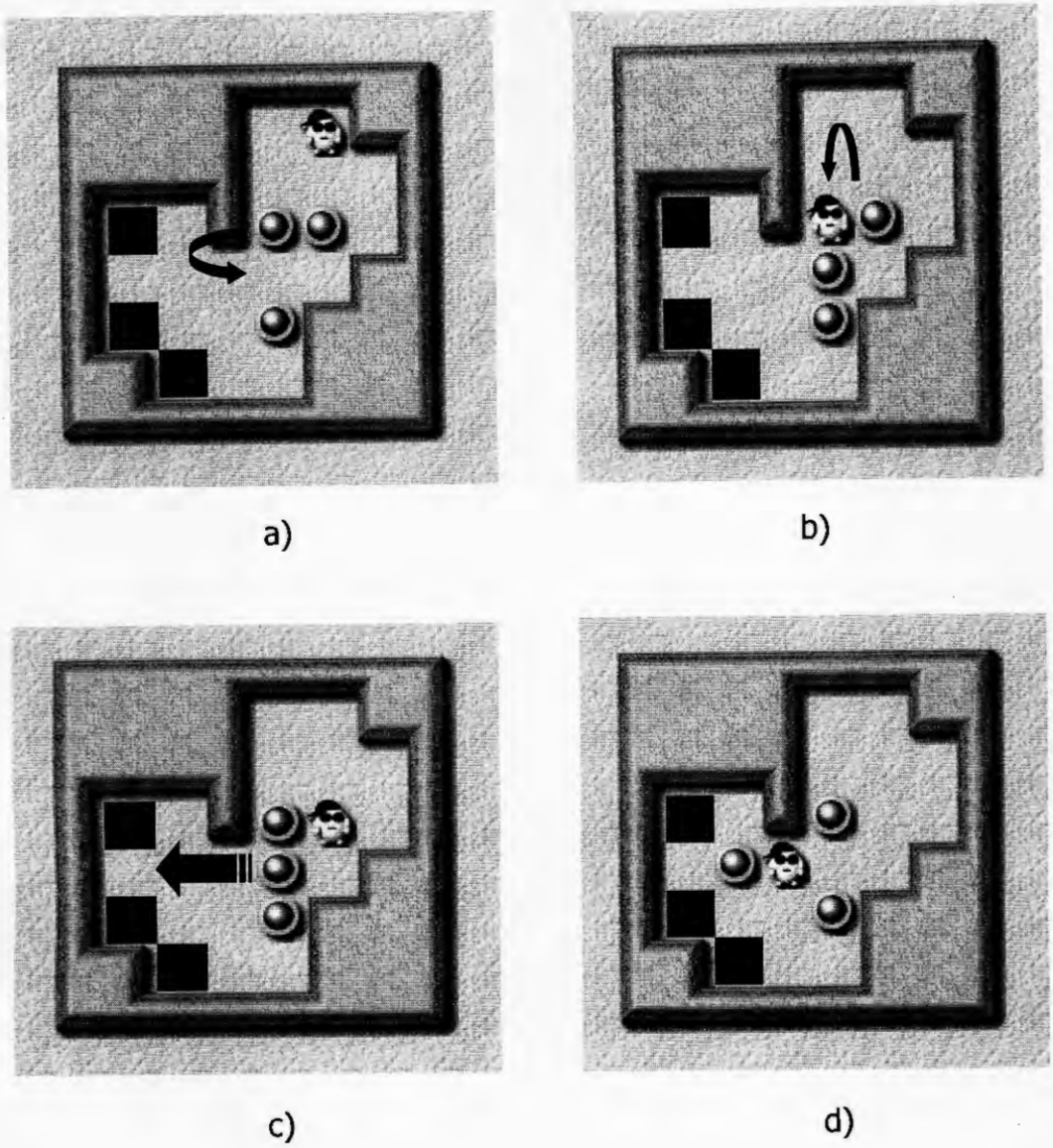


Figura 3.6 Movimiento de más de un objeto para poder llevar a su objetivo a cualquiera de ellos.

3.3 APLICACION DEL ALGORITMO A*

El algoritmo de solución y que garantiza encontrar la ruta óptima es el algoritmo A*, el cual se presenta a continuación.

Declarar la lista *Open* como una cola de prioridad
Declarar la lista *Closed* como una lista

INICIO

```
s.g = 0                    // s es el nodo inicial
s.h = Distancia_Estimada_al_Objetivo(s)
s.f = s.g+s.h
s.padre = null
PUSH s en Open
MIENTRAS Open no esté vacío
        pop nodo n de Open   // n tiene el valor más pequeño de la lista
        SI n es el nodo objetivo
                construir la ruta
                Terminar
        PARA cada sucesor n' de n
                Newg = n.g + costo(n,n')
                Si n' esta en Open o Closed y n'.g < = newg
                        Brinca al siguiente
                n'.padre = n
                n'.g = newg
                n'.h = Distancia_Estimada_al_Objetivo(n')
                n'.f = n'.g + n'.h
                SI n' está en closed
                        Borrarla de Closed
```

Si n' no está en Open

Push n' en Open

PUSH n en Closed

Regresa un mensaje de Error // Cuando no se encontró una ruta de solución

FIN DEL ALGORITMO

La versión del SOKOBAN que se utilizará en lo sucesivo, fue creada y distribuida como shareware por Björn Källmark, Källkod AB.; este shareware puede ser encontrado en la dirección http://www.sourcecode.se/sokoban de Internet y contiene 375 niveles en modo de dificultad medio y difícil.

De esta versión del juego en particular, se seleccionaron los niveles por número de objetos a acomodar, algunos de estos niveles se presentan en la tabla 3.1.

OBJETOS	NIVELES
3	7,9,10,14,25
4	8, 11,12,16,21,27
6	1,4,15
8	6,23
9	19,20,22,24
10	2,5,13,17
11	3

Tabla 3.1 Número de objetos de SOKOBAN y los niveles que los contienen

Ahora se aplicará a éste juego el algoritmo A*.

El algoritmo A* involucra dos datos muy importantes que deben ser contemplados para su desarrollo, estos son los que contienen la distancia de la trayectoria de un nodo a otro o del padre al hijo al que se llamará *g* y la variable *h* que contiene una heurística *h*, cuya función es la de guiar hacia la solución del problema. La suma de estos dos valores genera el valor de *f(n)*.

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

El segundo sumando de la formula anterior particulariza al nodo *n* en cuestión.

Del valor de h depende la velocidad de solución, si se escoge una heurística errónea el programa puede llegar a ser demasiado lento.

En el juego del 15 cualquier movimiento que se realice produce una configuración diferente de alguno de sus objetos, pero en el Sokoban puede haber movimientos que no afecten dichos objetos produciendo un estado que no debe de contemplarse, tal como se muestra en la Figura 3.7.

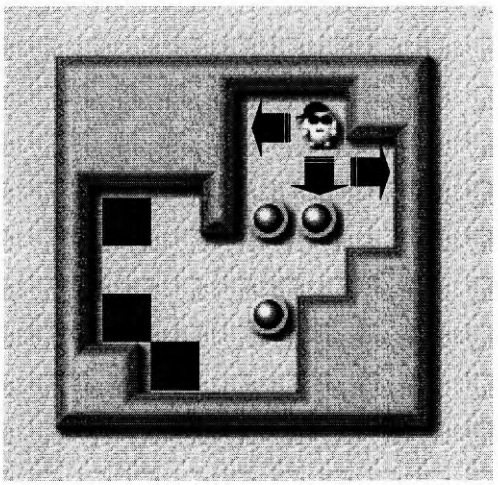


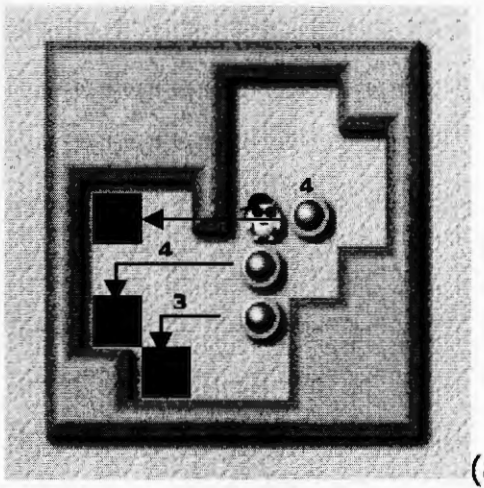
Figura. 3.7 Movimientos del Sokoban que no afectan los objetos.

Al análizar los movimientos del juego para encontrar la ruta de solución de una configuración dada, se podría pensar en únicamente introducir en las listas OPEN y CLOSED, aquellos movimientos en los que se refleje un empuje de objetos y omitir los demás movimientos, cualquiera que éstos sean, logrando de esta manera ahorrar tiempo de búsqueda ya que solo se contemplarían los movimientos que afectan de manera substancial la posición de los objetos hacia los objetivo, esto conduciría a que una vez resuelta dicha configuración, no se tendría la ruta de solución completa, ya que se omitirían los desplazamientos que el Sokoban tuvo que realizar, sin empujar, para poder seleccionar el objeto correspondiente y colocarse en la posición adecuada junto a él para empezar a empujarlo, lo que también evitaría que se pudiera apreciar las diferentes causas por las que no se pudo resolver dicha configuración en caso de que asi sucediera.

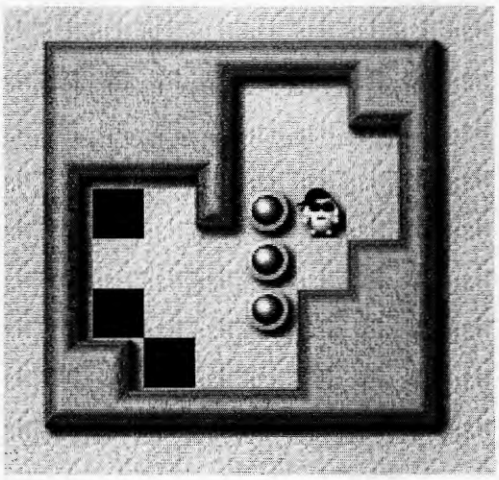
La heurística *h* de cada movimiento que se seleccionó para el juego, es la suma de las distancias más cortas de cada objeto a cada objetivo. Sea *n* el número de objetos a ordenar en una configuración dada y d la función que encuentra la distancia más corta entre un objeto x y el objetivo o, entonces la heurística *h* se puede encontrar como sigue:

$$h = \sum_{i=1}^{n} d(x,o)$$

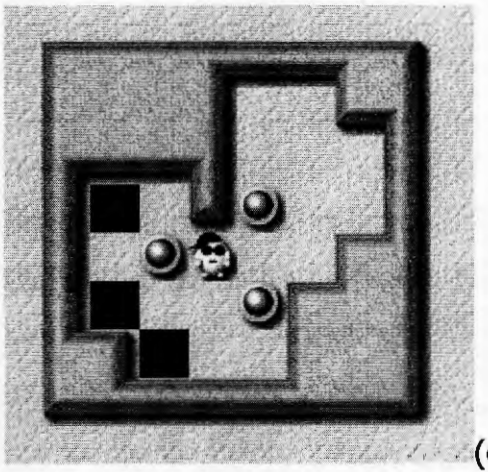
En la Figura 3.8 se puede apreciar la heurística que produce cada movimiento a partir de su configuración inmediata anterior.



(a)



(b)



(c)

Figura 3.8. (a) h = 11.
(b) h = 10.
(c) h = 8.

El Sokoban en cualquiera de los casilleros, sólo puede realizar un máximo de 4 movimientos de la misma manera que en el juego del 15, estos son, arriba, abajo, izquierda o derecha.

Esto provoca un problema, ya que dentro de la cola de prioridad OPEN se generan 4 movimientos con el mismo valor de la heurística debido a que los objetos continúan en sus mismas posiciones que en la configuración anterior, y el algoritmo no es capaz de saber cual de los 4 movimientos que se generaron es el mejor para estar más cerca del objetivo final, éste caso se puede apreciar en la Figura 3.7.

El único movimiento del Sokoban que produciría una heurística diferente al resto de sus movimientos a partir del mismo casillero es cuando se ejecuta el empuje de un objeto, éste ejemplo se puede observar con más claridad en la Figura 3.7 y comparar los posibles movimientos de la Figura 3.8

Lo anterior provoca asignar una heurística a los movimientos del Sokoban cuando no se realiza ningún empuje de objetos.

Para éste fin se calcula una distancia Manhattan del Sokoban a cada objeto y se selecciona el de menor valor, éste valor se asigna a la variable *g* la cual hasta éste momento siempre ha tenido un valor constante de 1 ya que representa la distancia recorrida de casillero a casillero; sea *e* la función que encuentra la distancia menor del Sokoban representado por *A* a cada uno de los objetos que serán ordenados, representado cada objeto por la variable *o'*, la formula quedaría de la siguiente manera:

$$g = e(A, o') \qquad \textbf{Ec. 1}$$

En donde g es la heurística de la posición actual del Sokoban a los objetos.

Considere una configuración sencilla del juego la cual se muestra en la Figura 3.9 a); de acuerdo a la heurística del Sokoban, el objeto que se encuentra a la derecha de él, esto es, en el casillero (3,5) si lo vemos como una matriz de 6X6, es el que se moverá primeramente a la posición objetivo que se encuentra en el casillero (3,6), ya que la distancia del Sokoban al objeto del casillero (3,5) es 1 y la distancia del objeto al objetivo es 1 por lo que la heurística es igual a 2, mientras que para el objeto del casillero (3,3), la distancia al Sokoban es 1 y la distancia del objeto al objetivo mas cercano es 2 por lo que la heurística es mayor ya que la suma de estos dos valores encontrados es igual a 3.

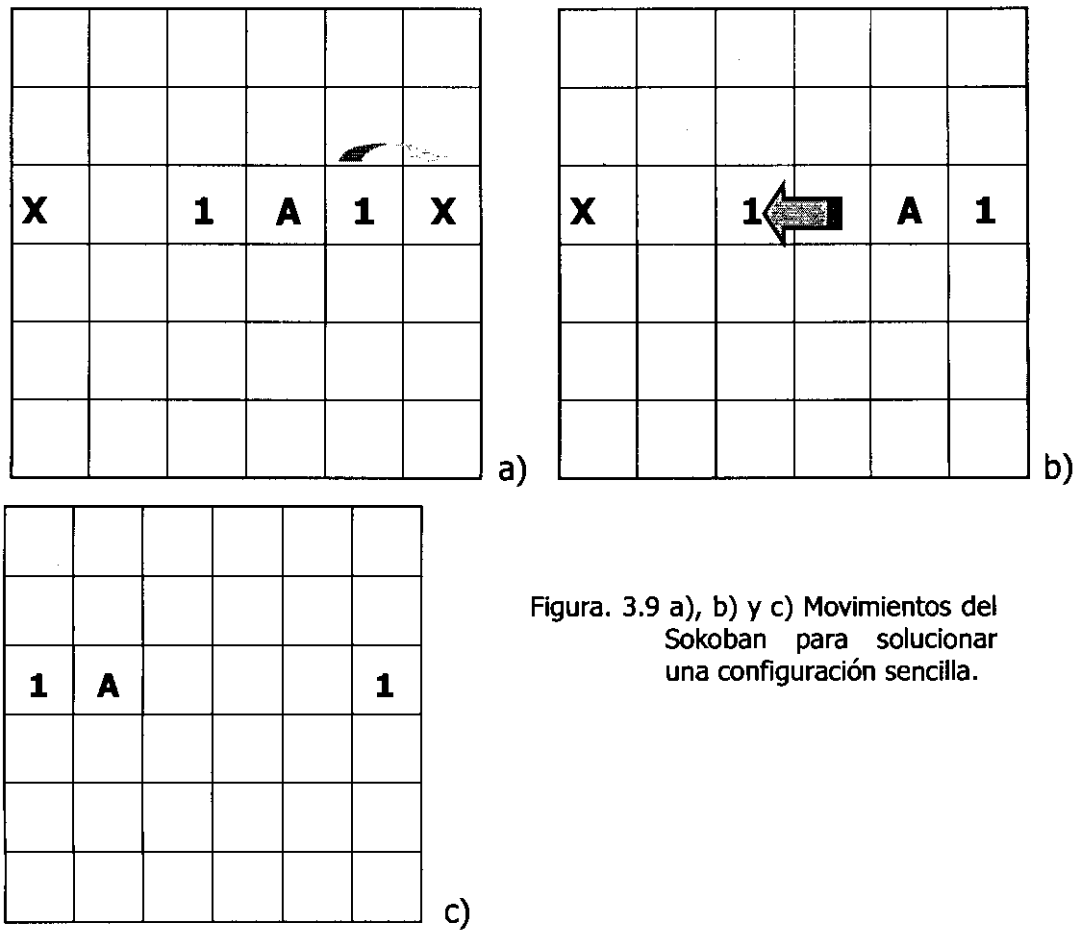


Figura. 3.9 a), b) y c) Movimientos del Sokoban para solucionar una configuración sencilla.

Después de desplazarse el Sokoban hasta el casillero (3,4) contiguo al objeto del casillero (3,3) Figura 3.9 b), éste será empujado hasta su posición objetivo localizada en el casillero (3,1) Figura 3.9 c), terminando así el juego. La posición de los objetos y del Sokoban es la que se muestra en la Figura 3.9 c).

Cabe mencionar que debido a que el valor de la distancia del Sokoban a los objetos es una heurística, ésta se debe sumar al valor de h y no omitir el valor de g el cual se incrementa en 1 en la generación de cada hijo, ya que ésta g proporcionará la distancia del inicio al objetivo una vez que éste sea alcanzado, de manera que el valor de h', será:

$$h = \sum_{i=1}^{n} d(x,o) + e(A,o') \qquad \textbf{Ec. 2}$$

Una vez que un objeto es llevado hasta una posición objetivo, si el algoritmo lo sigue teniendo en cuenta para el cálculo de sus heurísticas, éste objeto interferirá en el acomodo de los demás objetos que aún se encuentran pendientes por re-ubicar, ya que debido a que se encuentra en un casillero objetivo su heurística siempre será menor que la de cualquier otro objeto.

Para resolver éste problema, una vez que el objeto se encuentre en alguna posición objetivo, se ignorará y no se calculará heurística, considerando para éste caso sólo aquellos objetos que no se encuentren en casilleros objetivos.

Es importante analizar que sucede en un caso como el que se muestra en la Figura 3.10, en el que para poder colocar los objetos en los objetivos, algunos de ellos tienen que ser empujados a través de casilleros marcados como objetivos y en ese momento no serán considerados dentro del calculo de la heurística.

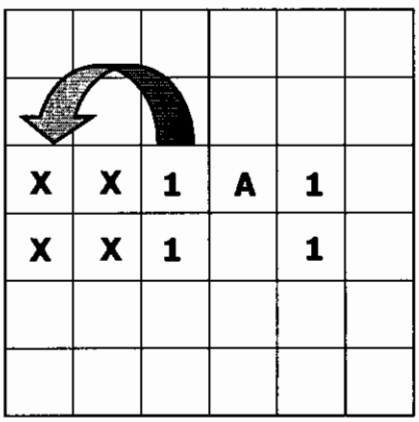


Figura. 3.10 Movimiento de los objetos a través de los casilleros.

Para solucionar este problema, basta con tener en cuenta la siguiente consideración:

La heurística sobre los objetos está dada por la siguiente ecuación:

$$h = \sum_{i=1}^{n} d(x,o)$$

En donde es posible darse cuenta de que el o los objetos que se encuentren sobre casilleros objetivos siempre tendrán la distancia más corta y que es:

$$d(x,o) = 0$$

Por lo que sí se considera o no su valor no afectará el resultado.

Debe considerarse la heurística obtenida por la distancia del Sokoban a los objetos, se encuentren o no en posiciones objetivo, para evitar que los objetos bloqueen el avance de otros objetos.

Otra consideración importante es detectar, en primer lugar, cuales son los casilleros en los que no puede ser colocado un objeto ya que si llegara a colocarse en él, el objeto quedaría bloqueado y por lo tanto el juego termina sin solución.

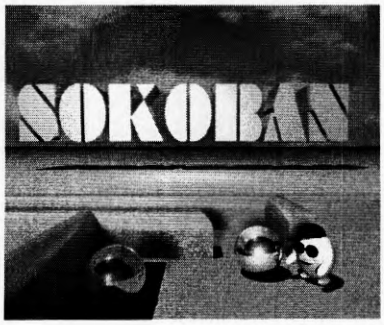
Para este efecto se realiza una búsqueda y marcación de los casilleros que están en una esquina, cuya característica es la de tener dos lados pared adyacentes, siempre y cuando estos lugares no estén señalados como objetivos. Con esto se logra eliminar una gran cantidad de casilleros del universo de posibilidades donde pueden colocarse objetos y por lo tanto de movimientos a realizar.

Con la consideración anterior se logran eliminar de manera automática muchas posibles configuraciones que no serán analizadas y reduce de cierta manera el universo total de búsqueda.

Cuando el algoritmo encuentra la solución despliega una secuencia de matrices que indican los movimientos que se tuvieron que realizar para llegar a la solución tal como se muestra en la Figura 3.9 c). ésto con la finalidad de dar una solución con una representación en forma de matriz.

¿Por qué las cosas son como son y no de otra manera?

Johannes Kepler (1571-1630) Astrónomo alemán



CAPÍTULO IV

RESULTADOS Y CONCLUSIONES

El algoritmo Backtrack es un algoritmo de búsquedas exhaustivas que dependiendo de la configuración inicial insertará más o menos configuraciones resultantes en el árbol, las cuales pueden o no llevar a la solución.

Esta naturaleza de búsquedas exhaustivas es el punto débil del algoritmo BACKTRACK, ya que realiza movimientos en algunos casos innecesarios y provoca que el tiempo de búsqueda de la solución en ocasiones sea muy grande y consuma demasiados recursos del sistema, pudiendo, en el peor de los casos, saturar la capacidad del mismo.

En el capítulo II, se analizó y se aplicó el algoritmo A*, los resultados obtenidos con este algoritmo fueron muy satisfactorios ya que se logró encontrar rutas de solución óptimas en cuanto a su longitud o valor que las generadas con el algoritmo Backtrack, por ejemplo, la configuración mostrada en la Figura 4.1 al ser

resuelta mediante la aplicación del algoritmo Backtrack, la ruta de solución encontrada está formada por 30,659 pasos o movimientos de fichas; la ruta de solución obtenida para ésta misma configuración mediante el algoritmo A* consta de 39 pasos.

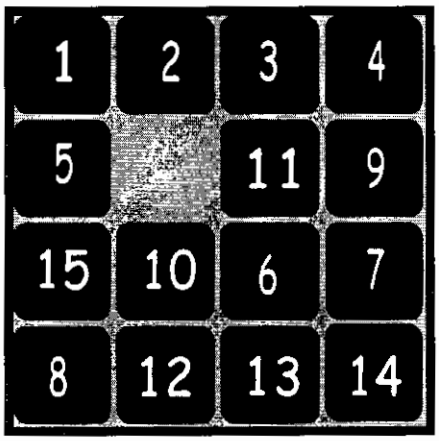


Figura 4.1 configuración del juego del 15 resuelta mediante Backtrack y A*.

En el Apéndice A se pueden observar algunos ejemplos de rutas de solución encontradas mediante los algoritmos Backtrack y A*.

Ésta es una de las grandes desventaja del algoritmo Backtrack ya que la ruta encontrada no siempre es la óptima ya que dicha ruta puede contener movimientos innecesarios, en aquellos casos en los que la ruta encontrada por medio de la aplicación de este algoritmo, sea óptima, esto puede ser considerado como una coincidencia ya que no siempre será así.

En la Figura 4.2 se presenta una gráfica que muestra la diferencia de longitudes de las rutas encontradas para algunos ejemplos de configuraciones aplicando el algoritmo Backtrack y A*, estas configuraciones y partes de sus rutas se localizan en el apéndice A.

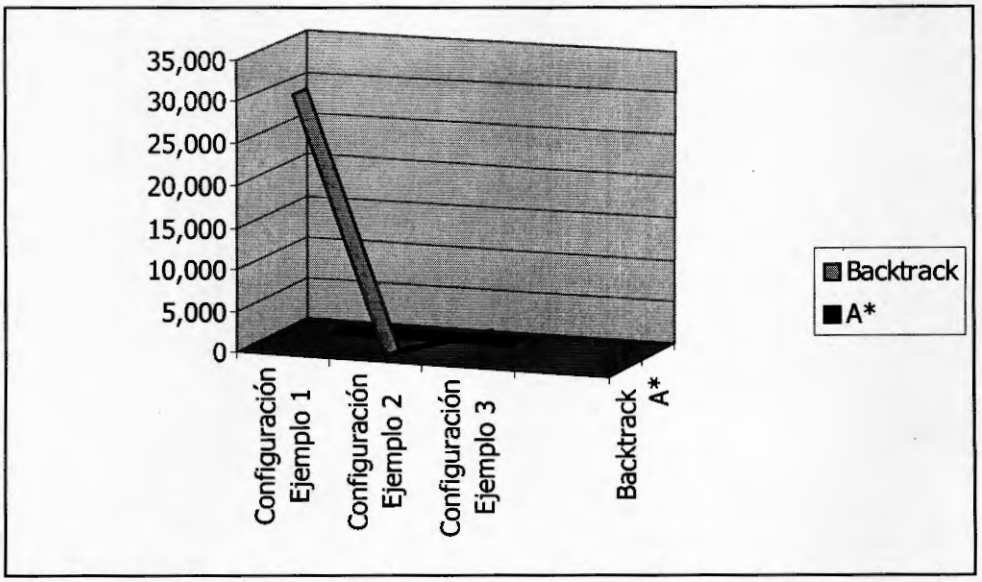


Figura 4.2 Gráfica de las longitudes de las rutas óptimas encontradas mediante los algoritmos Backtrack y A*.

Se debe tener en cuenta que la diferencia en las longitudes de las rutas encontradas para cada configuración se debe a la eficiencia que tienen dichos algoritmos para encontrar una solución, como se vio en el Capítulo II la complejidad del algoritmo A* es logarítmica y la complejidad del algoritmo Backtrack no lo es, lo que pone en ventaja al algoritmo A*.

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

Frigura 4.3 Complejidad del algoritmo A*.

Los libros de los autores consultados fueron de mucha utilidad en el trabajo realizado. Investigando sobre como resolver el juego del 15 algunos de ellos solo mencionan este juego como un gran ejemplo de búsquedas exhaustivas pero nunca mencionan un algoritmo que ellos aseguren soluciona el problema, siempre muestran como ejemplo un puzzle de 2X2 o de 3X3 e indican que así como se resuelven éstos, se resuelve el del 15, esto no es de gran utilidad ya que no se especifica como es que se resuelve de manera tan fácil como lo dicen, la configuración del juego del 15 la mencionan como muy popular y que se resuelve siguiendo los pasos de las configuraciones pequeñas.

En ellos se mencionan que se puede aplicar el algoritmo de Backtrack, pero que tiene sus desventajas ya que efectúa demasiados movimientos y puede llegar a ser demasiado tardado, lo que se comprobó al estar resolviendo configuraciones distintas del juego del 15 en el capitulo I.

Judea Pearl menciona el juego del 15 y afirma que Nilsson creó un algoritmo que lo puede resolver, encontrando al final la ruta óptima, hace referencia a su libro[Nilsson Nils J. 1971], encontrándose la descripción y análisis del algoritmo A*, pero en ningún lugar se proporciona la respuesta o bien un ejemplo práctico, aquí es donde se empieza la investigación sobre el buen desempeño de este algoritmo y se realiza la programación del mismo, pudiendo hacer al final del capítulo II la comparación con el desempeño del algoritmo Backtrack.

En la gran mayoría de las direcciones de Internet solo mencionan este juego como un gran tema de estudio pero sin mostrar algún modelo matemático que facilite el encontrar una solución, incluso muchos mencionan que es más interesante la variante de este juego llamada el cubo de Rubick, lo cual depende del gusto de cada persona. El cubo de Rubick a sido objeto de más estudio, al menos es lo que aparenta en Internet, debido a su gran popularidad que tuvo en los años 80's, pero sus movimientos tridimensionales facilitan mucho la elaboración de algoritmos que lo solucionen. El juego del 15 es más interesante debido a que solo se tiene una estructura plana y esto dificulta su solución, aunque con menos cuadros o espacios por acomodar.

El juego del 15, en Internet, es popular en cuanto a tratar de resolver una configuración dada y ganar algún premio y es muy común que sea presentado con dibujos y figuras atractivas que no vienen al caso mencionar en esta tesis ya que

su aportación en ese sentido no es importante para el conocimiento y la investigación.

La información más recomendable es la que se encuentra en algunas páginas de clubes o asociaciones de matemáticos (ver sección de direcciones de Internet), que tampoco, al menos en lo que se investigó en Internet, dan una solución contundente.

La gran parte la información y las pistas que llevan a la solución y a aplicar el algoritmo más eficaz se encuentra en libros de inteligencia artificial, algunos de ellos que datan de los años 60's o anteriores, esto no debe de sorprender ya que el principio de la computación se encuentra en las investigaciones matemáticas de hace varias décadas e incluso siglos.

El trabajo teórico que se presenta es fundamental para el entendimiento y comprensión de la importancia de encontrar una solución, la teoría lleva a escudriñar el porqué de las cosas, se puede afirmar que no puede haber una investigación con resultados prácticos, si no existe previamente una labor de investigación teórica.

Es importante señalar que la teoría de la computación data, como ya se mencionó con anterioridad, de hace muchos años, representado en trabajos teóricos que hicieron personas que no sabían que fueran a existir computadoras, pero con el paso del tiempo esa teoría desarrollada por ellos fue fundamental para la creación de una nueva ciencia.

El juego del 15 y el Sokoban son juegos de destreza y agilidad mental que para muchas personas resultan aburridos, pero es necesario que la gente se involucre en ellos para darse cuenta de la importancia que tiene si se logra aplicar

a la vida diaria, representan modelos matemáticos que dan solución a problemas de acomodamientos, de ahí el nombre del Sokoban, acomodador de cajas o almacenista.

Es importante señalar que para la aplicación de ambos algoritmos se deben de establecer y tener en cuenta las reglas de movimientos y criterios que nos conduzcan a la solución de la configuración a resolver, uno de ellos es el orden en el que se realizarán los movimientos; hay que recordar que en este trabajo se siguió el orden de búsqueda de movimientos válidos a partir del cuadro en blanco para el caso del juego del 15 o bien del Sokoban para el caso del juego del mismo nombre, de izquierda, abajo, derecha y arriba; si al realizar el movimiento a la izquierda, éste no podía llevarse a cabo o no llevaba a la solución, se procede a efectuar el movimiento siguiente, hacia abajo, inmediatamente después el de la derecha y así sucesivamente.

Si se decidiera comenzar los movimientos teniendo en cuenta otro criterio, como empezar a hacer la búsqueda hacia abajo o a la derecha o arriba, en el caso del algoritmo Backtrack, se pueden encontrar rutas de solución diferentes a las presentadas en el Apéndice A para cada configuración dada, ya que este algoritmo realiza búsquedas exhaustivas sin garantizar que sea la óptima; en el caso del algoritmo A* no importa la dirección donde se empiecen a realizar los movimientos, ya que éste siempre encontrará la ruta óptima a menos que existan dos rutas óptimas diferentes en cuanto a configuraciones generadas pero con el mismo valor o costo de solución.

La generación de configuraciones crece exponencialmente con cada configuración padre examinada por lo que se genera una gran cantidad de elementos que son almacenados en listas ligadas, ningún elemento o configuración encontrada es desechada a menos que se halla llegado a ésta con anterioridad;

como el algoritmo necesita comprobar esto último, debe de recorrerse dichas listas comparando cada elemento, tarea que consume mucho tiempo y produce lentitud durante la ejecución del algoritmo, esto implica implementar árboles y búsquedas binarias que logran una eficiencia mayor en la ejecución del algoritmo.

Los algoritmos de solución mencionados son muy relevantes, el hecho de que el Backtrack no resulte el mejor algoritmo para la solución de éste, no le resta importancia en cuanto a su aportación a la ciencia. Su método de búsqueda exhaustiva lleva a una solución un poco lenta en este juego y que consume muchos recursos computacionales.

Para la solución del Sokoban es muy importante tener en cuenta dos heurísticas, la de los objetos y la del Sokoban o acomodador de los objetos, ya que la primera indica que objetos son los que se deben mover y la segunda la ruta que debe seguir el Sokoban hacía ese objeto, esto hace la diferencia con el juego del 15 ya que éste sólo tiene en cuenta la heurística de la ficha en turno, criterio que aumenta la complejidad del Sokoban.

El tiempo de ejecución del algoritmo aumenta en razón del tamaño de la matriz formada por la configuración que se desea resolver, si no existen limitaciones en cuanto a memoria y del equipo que se utiliza para la ejecución del algoritmo pueden ser resueltas configuraciones de matrices de más de 20X20.

Este trabajo de investigación tiene mucho futuro, ya que puede ser continuado por investigadores interesados en la automatización de búsqueda de uni-agentes en espacios cerrados con obstáculos de manera física, como el caso de un almacén en el que los productos empacados deben ser acomodados en lugares establecidos, el hecho de que el mecanismo que realiza este acomodo sea tan sencillo como el de solo empujar puede disminuir algunos costos de fabricación

pues el brazo que lo realiza solo tiene un movimiento y de la misma manera costos de administración.

Un trabajo posterior, podría ser el de crear la interface que comunique el programa computacional con un robot o mecanismo electromecánico.

BIBLIOGRAFIA

A.J. Jones, B.Sc., Ph.D., 1980, "Game Theory: Mathematical Models of Conflict", Department of Mathematics Royal Holloway College, University of London. ELLIS HORWOOD LIMITED Publishers Chichester.

Chung-Hung Tzeng "A Theory of Heuristic Information in Game Tree Search"

Gingberg Matt. 1993. "Essentials of Artificial Intelligence", Morgan Kauffmann

Kaplan Edward L., 1982, "Mathematical Programming and Games". Department of Mathematics, Oregon State University, 1982 ISBN 0-47103632-3 AACR2, Jhon Wiley & Sons, Inc.

Minieka Edward, 1992, Volumen I, "Optimization Algorithms For Networks and Graphs", Department of Quantitative Methods, University of Illinois at Chicago Circle. Chicago Illinois, Marcel Dekker, Inc.

Nilsson Nils J. 1971, "Problems-Solving Methods in Artificial Intelligence", McGrawn Hills.

Pear Judea, 1984, "Heuristics", Addison-Wesley.

Reingold Edward M., Nievergelt Jurg, Deo Narsingh, 1977, "Combinatorial Algorithms: Theory and Practice", Prentice Hall.

Rich Elaine, 1983, "Artificial Intelligence", Mc Graw Hills.

Roberts Fred S. 1984 "Applied Combinatorics" Prentice Hall.

Russel Stuart, Norving Peter, 1995, "Artificial Intelligence, A Modern Approach", Prentice Hall.

Shoham Yoav. 1994. "Artificial Intelligence Techniques in Prolog".Morgan Kauffmann.

Grimaldi Ralph P., 1994, "Discrete and Combinatorial Mathematics, An Applied Introduction", 3rd. Edition, Adidison-Wesley.

Stefan Edelkamp, Korf Richard E. "The branching Factor of Regular Search Spaces". Institut für Informatik, Am Flughafen 17, 79110 Freiburg.

Richard E. Korf, Computer Science Department, University of California, Los Angeles, Ca. 90095.

American Association for Artificial Intelligence.

Stefan edelkamp, Jürgen Eckerle, "New Strategies in Learning Real Time Heuristic Search". Institut für Informatik, Albert-Ludwigs.Universität, Am Flughafen 17,D-79110 Freiburg.

Stefan Edelkamp, "Suffix Tree Automata in State Space Search", Institut für Informatik, Albert-Ludwigs.Universität, Am Flughafen 17,D-79110 Freiburg.

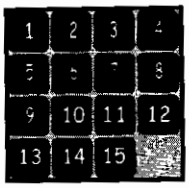
DIRECCIONES DE INTERNET

[www 1]http://www.gamasutra.com/features/programming/19990212/sm_04.htm
[www 2]http://www.informs.org/conf/dallas97/talks/ta32.html
[www 3]http://www.cut-the-knot.com/do-you-know/sgroups/html
[www 4]http://www.mscs.mu.edu/~globsol/user_guide/13whatis/node4.html
[www 5]http://www.sleepless.com/pocoman/index.html
[www 6]http://www.home.newsfactory.net/~sokoban/sokoengl.htm
[www 7]http://www.astro.virginia.edu/~eww6n/math/15puzzle.html
[www 8]http://www.ex.ac.uk/cimt/puzzles/slideblk/sbint.htm
[www 9]http://www.cut-the-knot.com/pythagoras/slider_proof.html "Sam Loyd's Fifteen"
[www 10]http://www.his.com/~pshapiro/about.ss.html "The education value of Sokoban".
[www 11]http://www-cs-students.stanford.edu/~amitp/Articles/Astar3.html
[www 12] http://www.math.tu-berlin.de/~combi/peaple/former_members_pages/schaeftter/

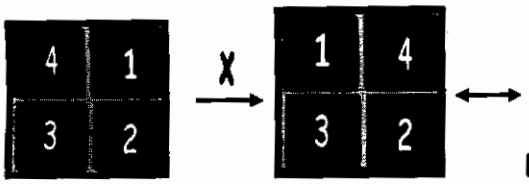
GLOSARIO DE TÉRMINOS

Heurística Deriva del griego "Heurisken" que significa descubrir, el significado técnico a cambiado a lo largo de la historia. Newell, Shaw and Simon, establecieron en 1963 es: "Un proceso que puede solucionar un problema dado pero no ofrece garantía alguna de que lo resolverá, esto es llamado heurística del problema"

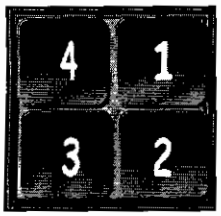
Puzzle nombre en ingles que se le da a los rompecabezas, el juego del 15 y el Sokoban son puzzles, en este trabajo se les menciona con este nombre ya que en la mayoría de la literatura existente se les conoce con este nombre.



Tablero del juego del 15 (15 puzzle), consta del tablero de 16 cuadros o posiciones un espacio en blanco y 15 fichas numeradas, la configuración presentada en este glosario es la configuración objetivo.



Movimiento representado por una variable, para este caso la variable *X* representa el intercambio de las fichas 1 y 4.
Existen y se utilizan en este trabajo más variables, las cuales indican por medio de una flecha en el frente de la figura, el tipo de movimiento que realizan.

$(XT)^3 =$ 

$= T^4 = I$

Operaciones con las variables *X* y *T*.

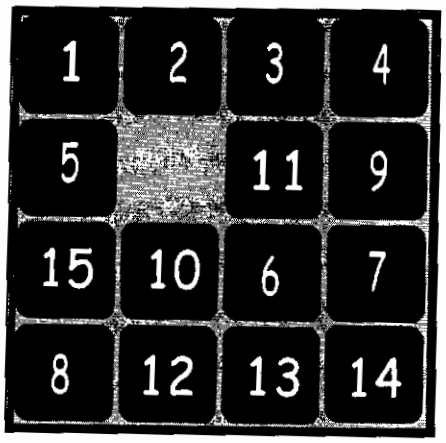
Distancia Manhattan es la suma de los cuadros o fichas (distancia) existentes entre la ubicación actual de cada una de las piezas y la ubicación destino.

APÉNDICE A

En este apéndice se presentan las rutas de solucion de algunas configuraciones del juego del 15, Boxworld y Sokoban, los pasos que llevan a la solución se encuentran en modo texto y la secuencia de los listados son en sentido inverso, esto es, del estado final (ya resuelto) al estado inicial del problema (configuración inicial), lo anterior debido al recorrido en el árbol de solución una vez encontrado el objetivo.

RUTA DE SOLUCIÓN APLICANDO EL ALGORITMO BACKTRACK

Ejemplo 1.- A continuación se presenta un fragmento del listado de la ruta de solución encontrada para resolver la configuración siguiente del juego del 15, utilizando el algoritmo BACKTRACK.



La ruta se presenta en las gráficas siguientes a partir de la configuración final (objetivo) a la configuración inicial; sígase esta ruta de izquierda a derecha y de arriba abajo, en caso de preferir seguir la ruta en el orden en el que se encuentra la solución léase de manera inversa.

Configuración objetivo

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

El cuadro en blanco es el que permite deslizar las fichas para poder llevarlas hasta su posición objetivo.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

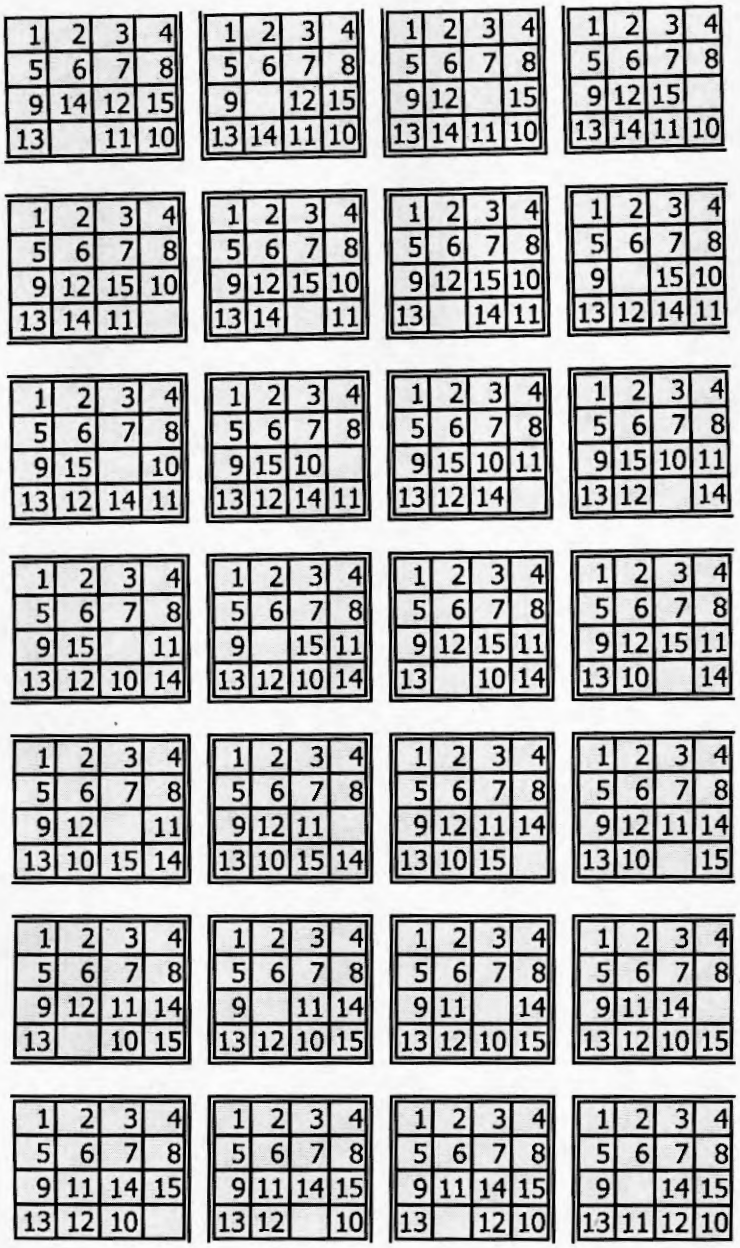
1	2	3	4
5	6	7	8
9		14	12
13	11	10	15

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

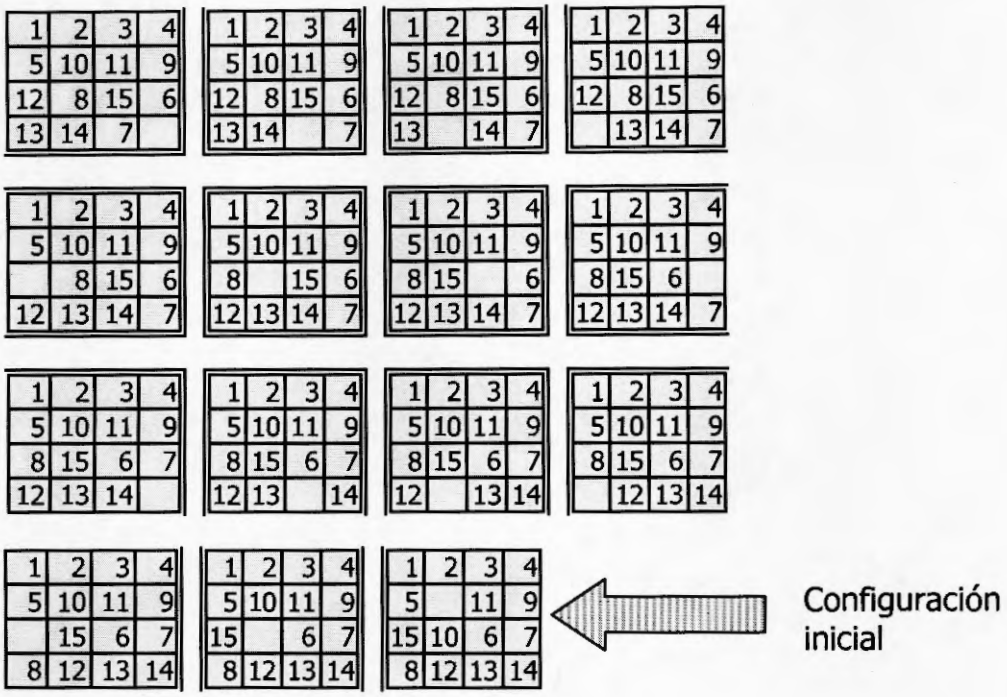
1	2	3	4
5	6	7	8
9	14	12	
13	11	10	15

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

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



La ruta total consta de 30,659 configuraciones diferentes por lo que solamente se muestran algunas de ellas, por tal motivo en esta parte, se trunca el listado y solo se muestran los primeras 15 pasos que se generaron a partir de la configuración inicial.



Ejemplo 2.- Las gráficas siguientes representan la ruta de solución encontrada para resolver la configuración siguiente del juego del 15 utilizando el algoritmo BACKTRACK.

Objetivo

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

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

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

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

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

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

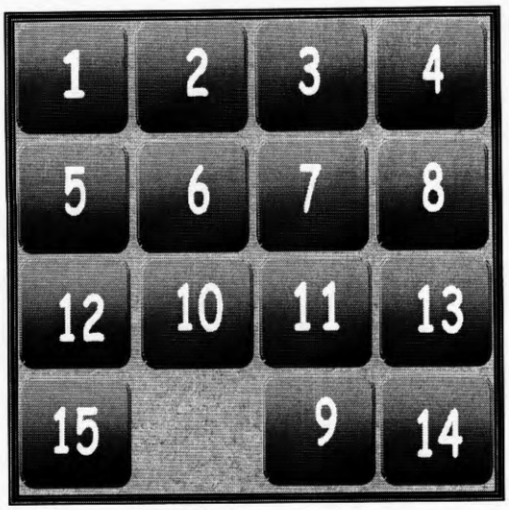
1	2	3	4
5	6	7	8
9		10	15
13	14	12	11

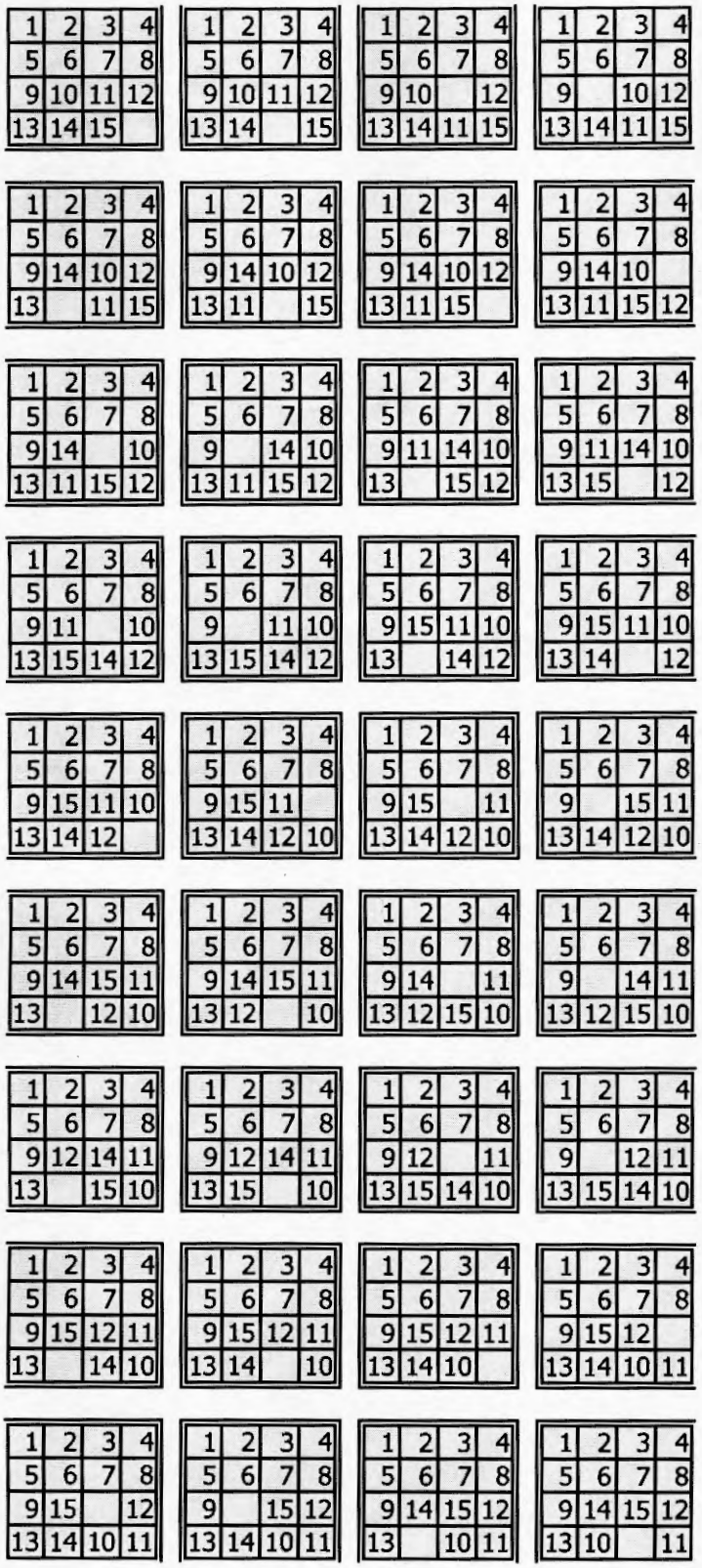
1	2	3	4
5	6	7	8
9	14	10	15
13		12	11

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

Configuración Inicial

Ejemplo 3.- Enseguida se presenta un fragmento del listado de la ruta de solución encontrada para resolver la configuración dada del juego del 15 utilizando el algoritmo BACKTRACK.




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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

En esta parte se trunca el listado de la ruta de solución para presentar solo los primeros 12 pasos a partir de la configuración inicial.

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

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

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

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

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

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

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

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

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

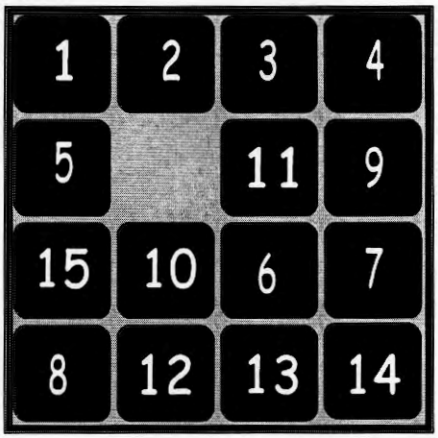
1	2	3	4
5	6	7	8
10	15	11	13
	12	9	14

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

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

RUTA DE SOLUCION DEL JUEGO DEL 15 CON ALGORITMO A*.

Ejemplo 1.- A continuación se presenta el listado de la ruta de solución encontrada para resolver la configuración siguiente del juego del 15 utilizando el algoritmo A*



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	
13	14	15	12

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

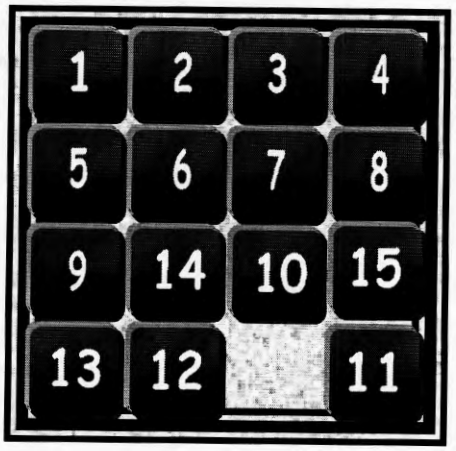
1	2	3	4
5	10	11	9
15	12	6	7
8	13	14	

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

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

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

Ejemplo 2.- Listado de la ruta de solución encontrada para resolver la configuración dada del juego del 15 utilizando el algoritmo A*.



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	
13	14	15	12

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

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

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

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

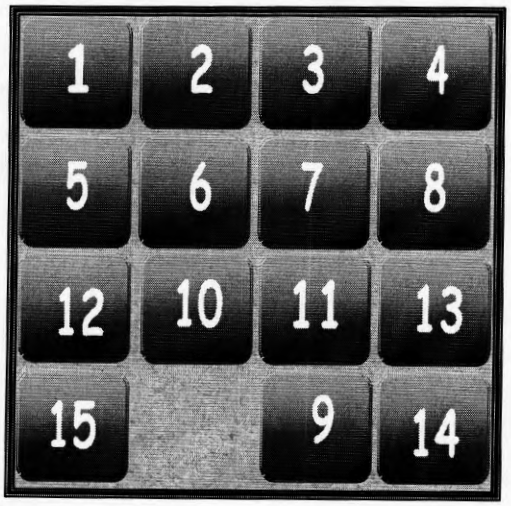
1	2	3	4
5	6	7	8
9	10		15
13	14	12	11

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

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

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

Ejemplo 3.- Listado de la ruta de solución encontrada para resolver la configuración dada del juego del 15 utilizando el algoritmo A*.



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	
13	14	15	12

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

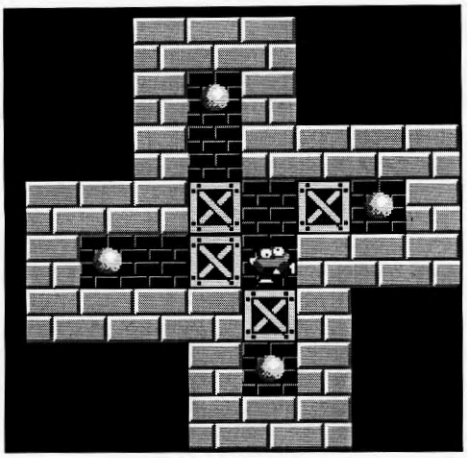
1	2	3	4
5	6	7	8
12	10	11	13
15	9		14

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

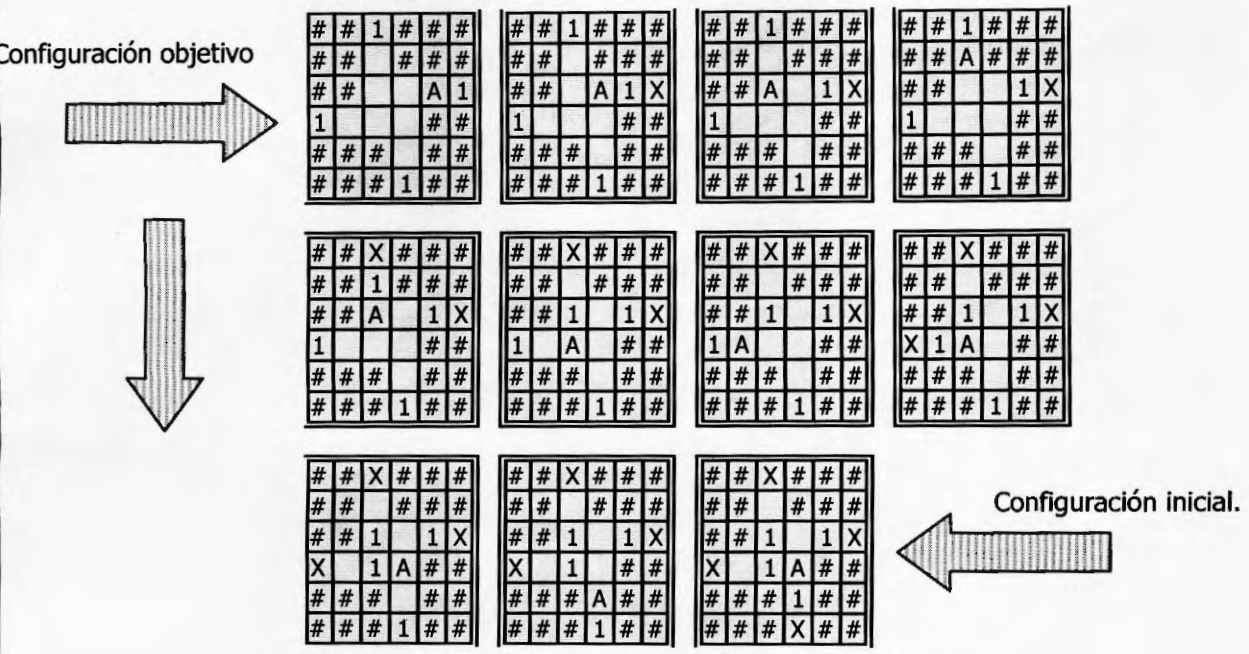
LISTADOS DE RUTA DE SOLUCION DEL BOXWORLD APLICANDO EL ALGORITMO A*.

Las gráficas que se presentan a continuación siguen la nomenclatura establecida en el capítulo II para representar el espacio de búsqueda y el medio ambiente en el que se desarrolla el juego del Sokoban o Boxworld.

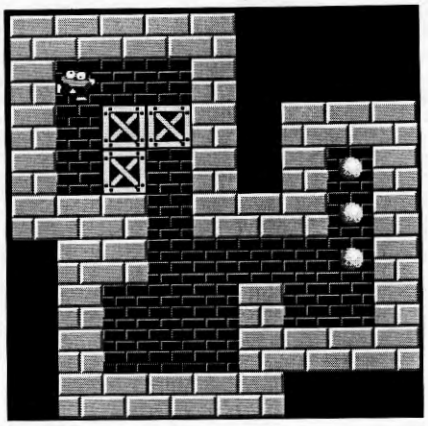
Ejemplo 1 Nivel 1 de BoxWorld la ruta óptima para encontrar la solución del 1 nivel de Boxworld se muestra a continuación, esta es una ruta que consta de solo 9 pasos ya que el décimo cuadro presentado es la configuración original.



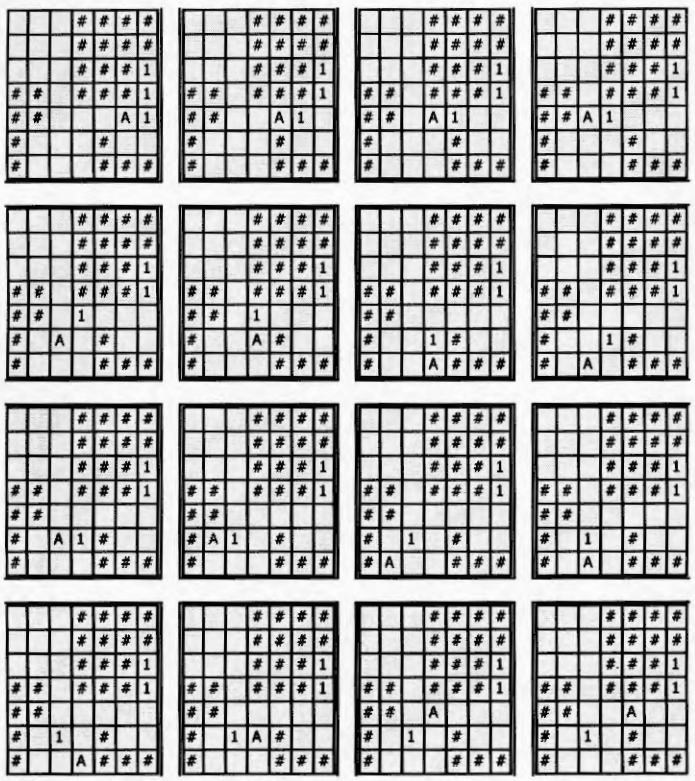
Se debe tener en cuenta que en este juego, el Sokoban, en este caso representado por un muñequito, tiene que alcanzar en primera instancia el objeto más cercano a él y hacer un empuje hacia algún objetivo, aunque no en todas las configuraciones puede ser movido el objeto más cercano a el ya que en determinado momento puede causa un bloqueo definitivo en el juego.

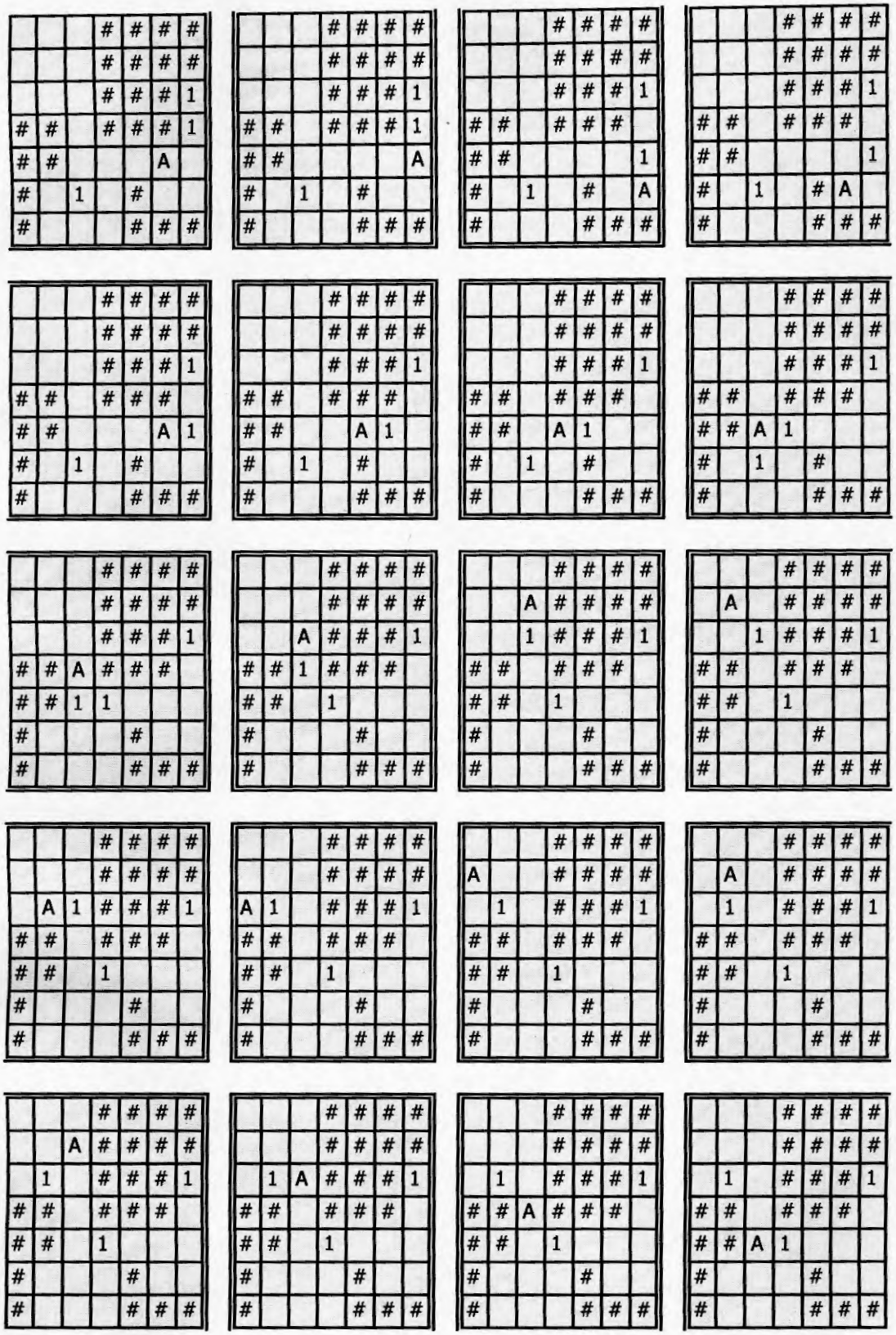

Configuración objetivo
Configuración inicial.

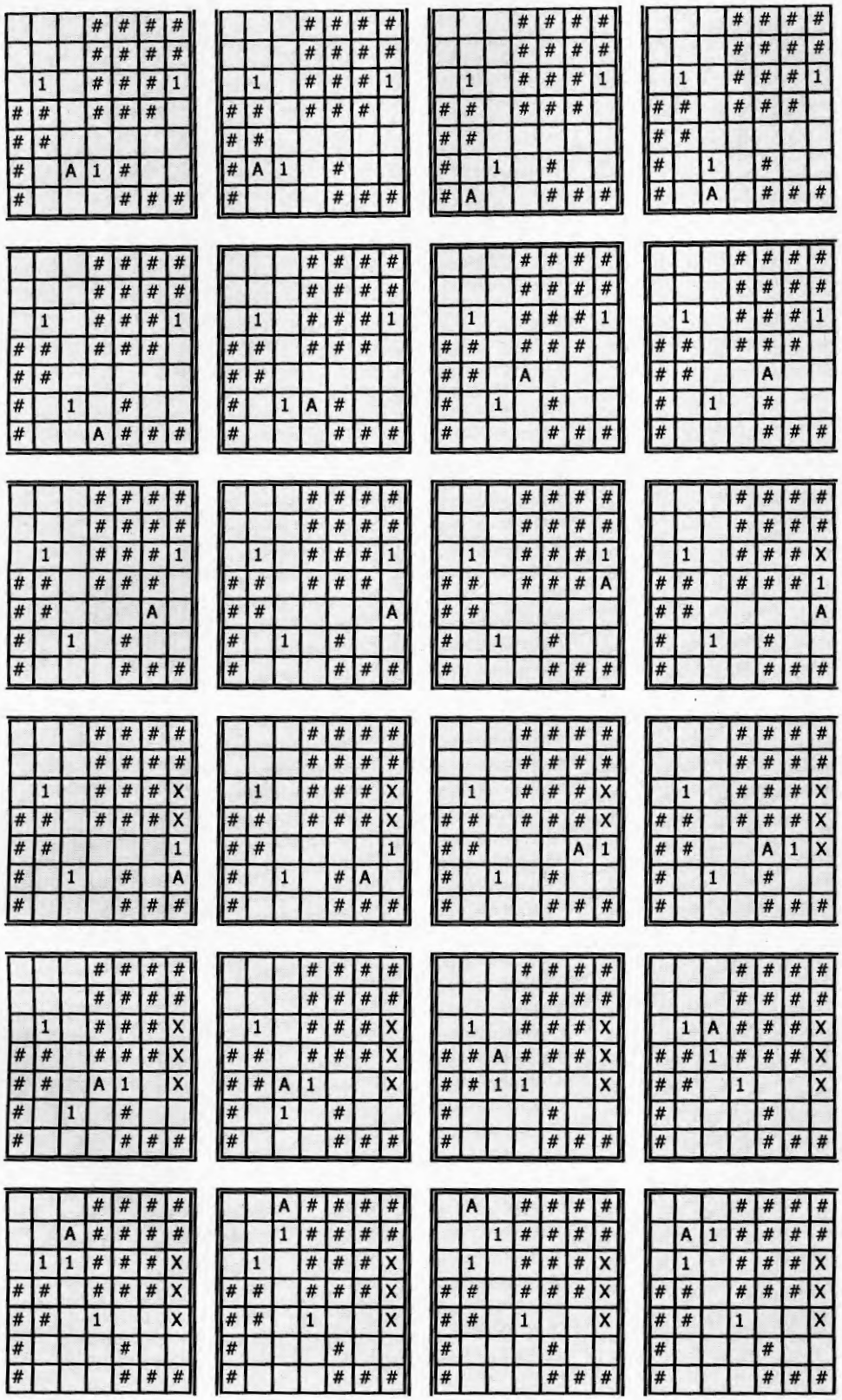
Ejemplo 2,. Listado de configuraciones generadas durante el recorrido de la ruta que soluciona el nivel 2 del Boxworld.

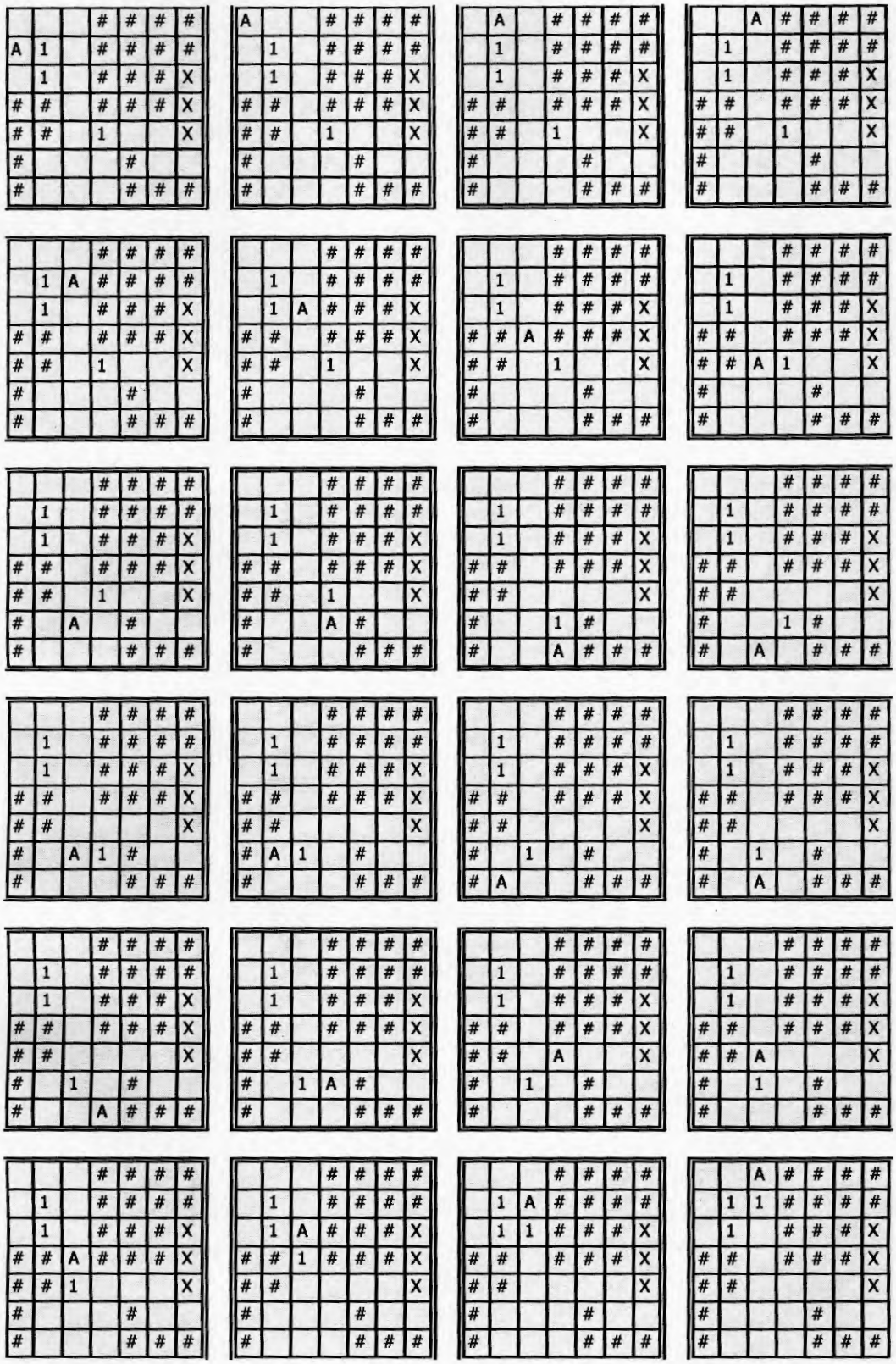


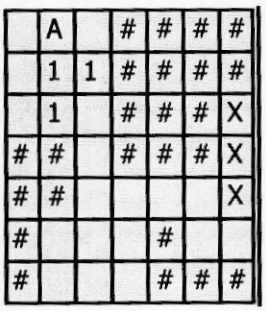
En este nivel el primer objeto que se debe de mover es el de más a la derecha del Sokoban ya que de otra manera se bloquearía el juego, para poder apreciar estos movimientos siga la ruta de solución leyendo del ultimo cuadro del listado hacia el principio, esto es de abajo hacia arriba y de izquierda a derecha. Hay que recordar que el listado de solución se presenta a partir de la solución a la configuración inicial.



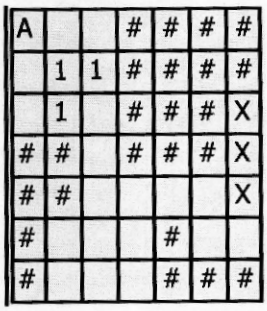






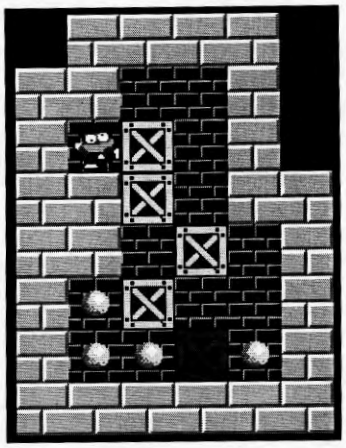


	A		#	#	#	#
	1	1	#	#	#	#
	1		#	#	#	X
#	#		#	#	#	X
#	#					X
#				#		
#				#	#	#



A			#	#	#	#
	1	1	#	#	#	#
	1		#	#	#	X
#	#		#	#	#	X
#	#					X
#				#		
#				#	#	#

Ejemplo 3.- Listado de configuraciones que se obtuvieron para encontrar la solución del nivel 4 del Boxworld



```
#    #    #    #
     #    #    #
#    #    #    #
#    #    #    #
1    A    #    #
1 1  1 1  1 #  1 #
```

```
#    #    #    #
     #    #    #
#    #    #    #
#    A    #    #
1    1    #    #
1 1  1 1  1 #  1 #
```

```
#    #    #    #
     #    #    #
#    A    #    #
#    1    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

```
#    #    #    #
     A    #    #
#    1    #    #
#    #    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

```
#    #    A    #    #
     #    1    #    #
#    #    #    #    #
#    #    #    #    #
1    #    #    #    #
1 1  1 1  #  1 #  1 #
```

```
#    A    #    #
     #    1    #
#    #    #    #
#    #    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

```
#    #    #    #
     A    1    #
#    #    #    #
#    #    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

```
#    #    #    #
     #    1    #
#    A    #    #
#    #    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

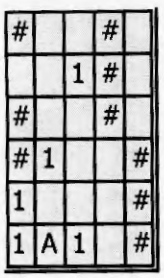
```
#    #    #    #
     #    1    #
#    #    #    #
#    A    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

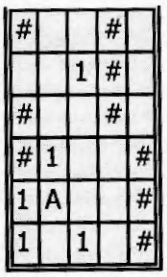
```
#    #    #    #
     #    1    #
#    #    #    #
#    A    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

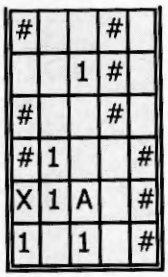
```
#    #    #    #
     #    1    #
#    #    #    #
#    1    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

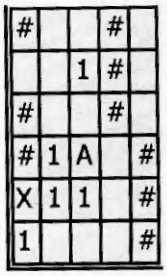
```
#    #    #    #
     #    1    #
#    #    #    #
#    1    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

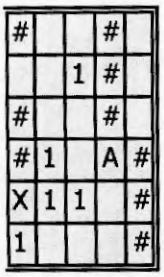
```
#    #    #    #
     #    1    #
#    #    #    #
#    1    #    #
1    #    #    #
1 1  1 1  1 #  1 #
```

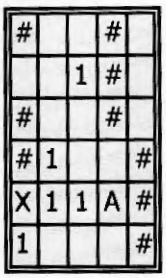


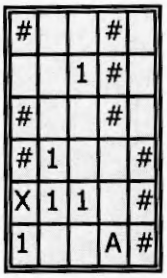


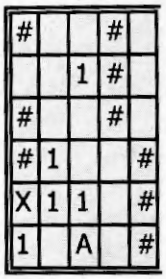


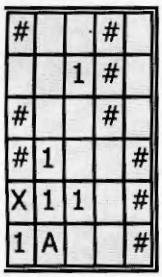


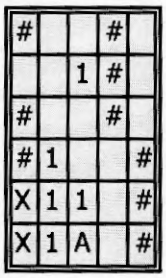


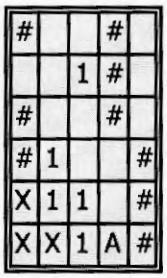


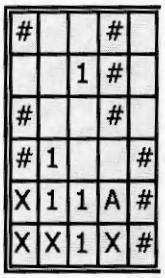


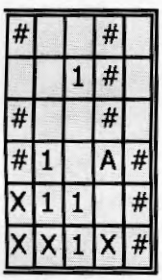


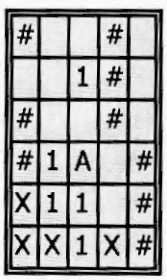


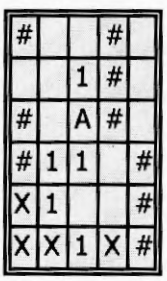


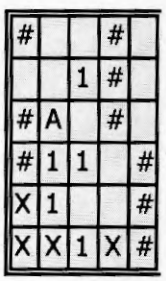


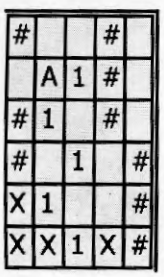


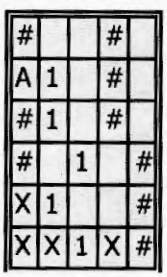




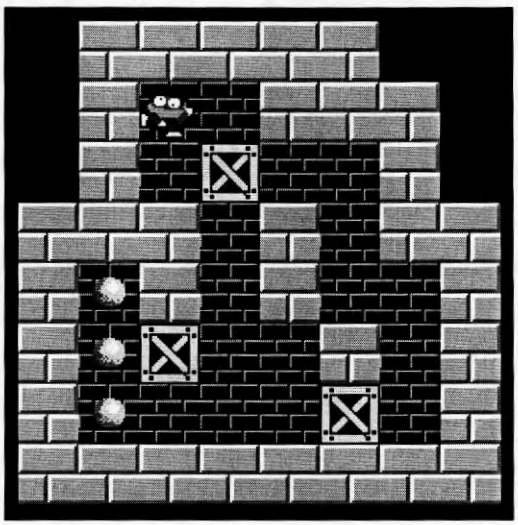








Ejemplo 4.- Listado que muestra la ruta de solución del nivel 5 del Boxworld



```
#    ###
#     ##
## # # #
1# # #
1A   # 
1
```

```
#    ###
#     ##
## # # #
1# # #
 1A  # 
1
```

```
#    ###
#     ##
## # # #
1# # #
  1A# 
1
```

```
#    ###
#     ##
## # # #
1# # #
  1 # 
1  A
```

```
#    ###
#     ##
## # # #
1# # #
 1  # 
1A
```

```
#    ###
#     ##
## # # #
1# # #
 1  # 
1A
```

```
#    ###
#     ##
## # # #
1# # #
 1  # 
 1A
```

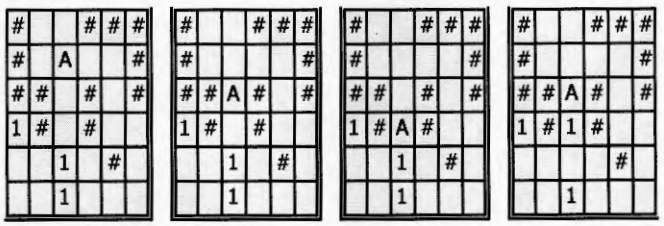
```
#    ###
#     ##
## # # #
1# # #
 1  # 
 1 A
```

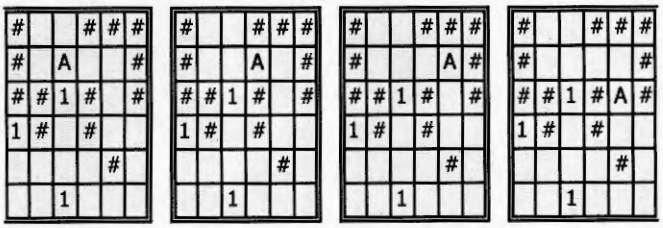
```
#    ###
#     ##
## # # #
1# # #
 1  #A
 1
```

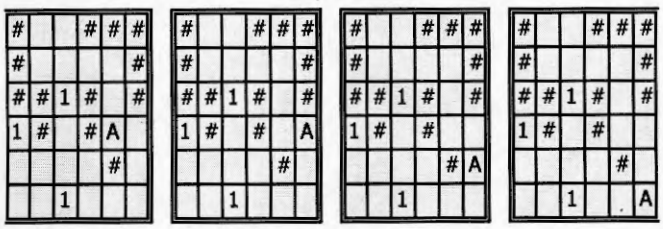
```
#    ###
#     ##
## # # #
1# # #
 1  # 
 1
```

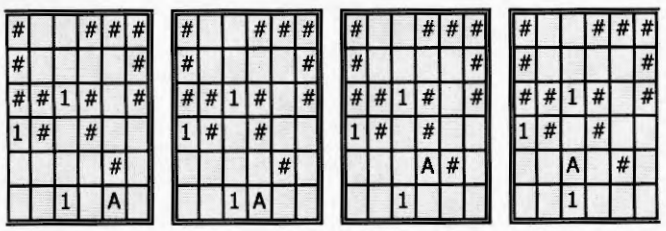
```
#    ###
#     ##
## # # #
1# # #A
 1  # 
 1
```

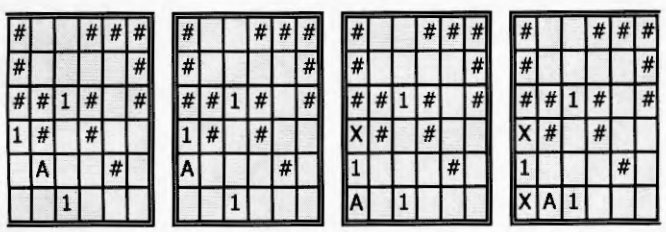
```
#    ###
#     A#
## # # #
1# # #
 1  # 
 1
```

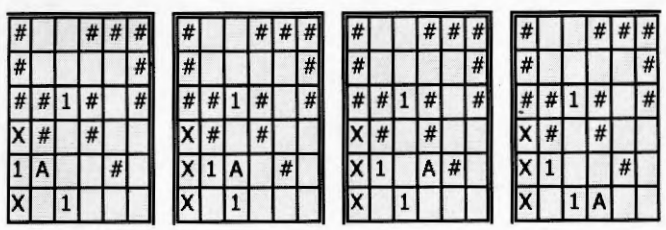


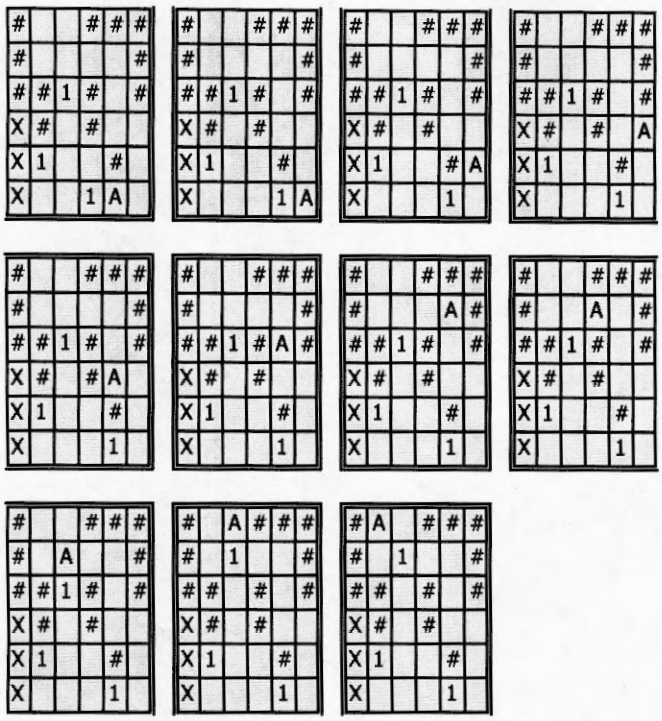











# # #
# #
1 #
X # #
X 1 #
X 1 A
# # #
# #
1 #
X # #
X 1 #
X 1 A
# # #
# #
1 #
X # #
X 1 # A
X 1 A
# # #
# #
1 #
X # #
X 1 #
X 1
# # #
# #
1 #
X # # A
X 1 #
X 1
# # #
# #
1 #
X # #
X 1 #
X 1
# # #
# #
1 # A
X # #
X 1 #
X 1
# # #
# #
1 #
X # #
X 1 #
X 1
# # #
A
1 #
X # #
X 1 #
X 1
A # #
1
# # # #
X # #
X 1 #
X 1
A # #
1
# # # #
X # #
X 1 #
X 1