PROYECTO FINAL DE ANÁLISIS DE ALGORITMOS

Fabián Andrés Merchán Jiménez
merchanf@javeriana.edu.co
José Miguel Sánchez Sanabria
jsanchez@javeriana.edu.co
Miguel Ángel Bermeo Ayerbe
bermeom@javeriana.edu.co

RESUMEN: *El proyecto final de la asignatura Análisis de Algoritmos de la Pontificia Universidad Javeriana, consta de ordenar un taquín haciendo uso de un algoritmo, dónde el tamaño del tablero varía desde un tablero de 2x2 hasta 10x10, en el cual el número de fichas está dado por* $N^2 - 1$ *siendo un tablero de NxN. El algoritmo debe competir contra otros algoritmos propuestos, por tanto se debe hacer uso de una interfaz, proporcionada por otros estudiantes del curso, la cual permite competir 2 algoritmos al tiempo. El algoritmo propuesto a continuación para resolver un taquín, es "Iterative deepening A*" (IDA*) usando como heurísticas la distancia de Manhattan, conflicto lineal ("Linear Conflict") y fichas mal colocadas ("Misplaced Tiles") para encontrar una solución.*

PALABRAS CLAVE*: Estructura de Datos, Grafo, Taquín, Heuriscta, IDA*, Distancia de Manhattan*

ABSTRACT: *The final project of the subject Analysis of Algorithms of the Pontificia Universidad Javeriana University, consists of ordering a sliding puzzle block employing an algorithm. The board size varies from a board of 2x2 up to 10x10, in which the number of blocks is given by* $N^2 - 1$*. The algorithm must compete against other proposed algorithms, therefore, it must make use of an interface provided by other students in the course, which allows to compete 2 algorithms at a time. The following algorithm to solve sliding puzzle uses "Iterative deepening A*" (IDA*) using as heuristic Manhattan Distance, Linear Conflict and misplaced Tiles to find a solution..*

KEYWORDS: *Taquín, Heuristic, IDA*, Manhattan distance*

I. INTRODUCCIÓN

El Taquín es un problema interesante para resolver usando heurística debido la alta cantidad de combinaciones posible, siendo $\frac{N!}{2}$ para un tablero de *NxN* esto significa que es un problema NP ya que validar que el resultado es en tiempo polinomio. El tablero está compuesto de $N^2 - 1$ piezas, cada una con un identificador único que debe estar en una posición del tablero para estar ordenado. Para armar el tablero se usa el espacio en vacío que hace la esencia de la dinámica del juego, la pieza faltante puede intercambiar permite mover una de las piezas que estén a su alrededor. Limitando la pieza vacía a cuatro movimientos posibles dependiendo de donde se encuentre del tablero, es decir si la pieza se encuentra en un borde, solo tiene tres movimientos disponibles y si está en una esquina solo dos.

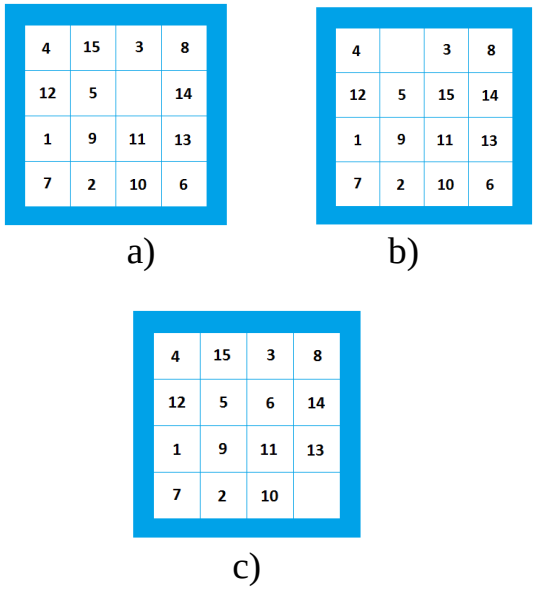

Figura. 1 a) tablero de 4×4 con la pieza vacía alejada de los bordes **b)** tablero de 4×4 con una pieza en un borde **c)** tablero de 4×4 con la pieza vacía en una esquina

La solución de tableros pequeños como *2x2* y *3x3*, solo basta con crear un grafo que contenga todas las combinaciones posibles ya que no son muchas (*2x2* tiene 12 combinaciones y *3x3* tiene 181440) y aplicar un algoritmo de búsqueda como Djisktra para encontrar el camino óptimo. Ya para los tableros de *4x4* es aumento es considerable siendo 1.0461395e+13, dificultando la memorización del grafo. Por esta razón se evaluó un algoritmo de búsqueda que baje la complejidad en tiempo y memoria haciendo uso de heurísticas

Se consideraron dos algoritmos A* e IDA*, son algoritmos que en principio no consumen memoria ya que no utilizan programación dinámica. Los dos requieren de una función de heurística, esta es la función que permitirá que los dos algoritmos converjan a la solución deseada. La diferencia de los dos algoritmos está en que A*

necesita incluir todos los nodos posibles por visitar mientras que IDA* solo los estrictamente necesarios bajando la complejidad. En la implementación nuestra IDA* ajustamos el algoritmo, asegurando que converja más rápido a la solución.

II. ANÁLISIS

A. *Entradas y salidas*

1. ***Entradas:***
   - *N Siendo el tamaño de la parte lateral del tablero NxN.*
     $$2 \leq N \leq 10$$
   - *Una secuencia de secuencias que contenga el tablero desordenado*
     $Root = \{\{row11,..row1N\}..$
     $\{rowN1,..,rowNN\}\}$
2. ***Salida:***
   - *Un conjunto de movimientos que representan el camino para llegar a la solución desde el estado de Root.*
     $$Path = \{Root, Node1,.. NodeM\}$$
     Siendo M el número de movimiento que deben hacerse.

Como se ha mencionado anteriormente el algoritmo espera una configuración del Taquin como la siguiente dentro de las posibles:

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

Figura. 2: Estado de inicio ("*Root*").

Finalmente se espera un conjunto de movimientos que lleven del tablero de la **Fig.2** a la **Fig.3.**

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

Figura. 3: Estado objetivo ("*Goal*").

Teniendo como precondición que todas las configuraciones que se reciban fueron obtenidas después de desordenar el tablero objetivo, entonces todos los movimientos sin repetir estados o configuraciones deben converger a ese tablero. Una representación de este grafo está en la figura 3.

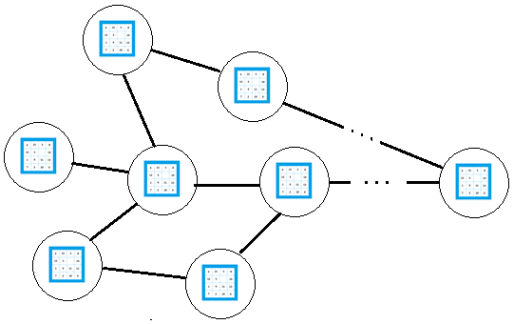

Figura. 3: Grafo con todas posibles combinaciones para resolver el tablero de *NxN*.

III. DISEÑO

El algoritmo propuesto busca un camino desde una configuración inicial a un objetivo, se busca que sea un camino óptimo, como se mencionó anteriormente la gran cantidad de combinaciones para los tableros de *4x4* en adelante. Por esta razón se implementaron dos algoritmos para resolver un conjunto de tableros:

A. Djisktra: se implementó un algoritmo que crea un grafo con todas las posibles convenciones para tableros 2x2 y 3x3. Luego se calcula el camino optimo usando el algoritmo Djisktra.

B. IDA y heurísticas: Para los tablero con tamaños iguales o superiores a 4x4, se usó IDA* como algoritmos de búsqueda basándose en tres heurísticas:*

i. Distancia de Manhattan.
ii. Conflicto lineal ("Linear Conflict").
iii. Fichas mal colocadas ("Misplaced Tiles")

A. *Djisktra*

Este algoritmo requiere de un grafo armado por lo que es necesario primero crear el grafo este proceso consta en hacer una búsqueda en profundidad o en anchura partiendo de un tablero ordenado. Calculado el grafo se tiene como entrada algoritmo las mismas mencionadas

anteriormente y el grafo creado. Entonces, seudocódigo seria el siguiente:

Procedure $taquin(Root, N)$
$G <- \emptyset$
$S <- \emptyset$
G $< V, E > <- armarGrafo(N)$
Goal $<- crearGoal(N)$
$S <- dijsktra(G, Root,$ Goal$)$
return S
end Procedure

Donde:

$\boldsymbol{G}$: $\boldsymbol{rmarGrafo}$(**N**) Crea el grafo recreando las posibles jugadas según el tamaño *NxN* del tablero siendo *V* los nodos del grafo y E los arcos.

$\boldsymbol{Goal}$: $\boldsymbol{crearGoal}(\boldsymbol{N})$ Entrega el nodo objetivo el cual es el tablero con las piezas ordenadas.

$\boldsymbol{S}$: $\boldsymbol{dijkstra}(\boldsymbol{G}, \boldsymbol{Root}, \boldsymbol{goal})$ Encuentra la serie de pasos que se deben realizar para resolver el tablero $Root$ de la forma más eficiente. Para ello el algoritmo se basa en el algoritmo Dijkstra. El algoritmo recibe el grafo de posibles soluciones G, el tablero ordenado $Goal$ y el tablero que se quiere ordenar $Root$. El algoritmo devuelve la secuencia de pasos a realizar en una estructura S.

B. *IDA* y heurísticas*

Lo más importante del uso de este algoritmo es usar una buena heurística. Esta está inmersa en una función *F*, que es usada por IDA* para decidir qué camino tomar. Esta función está compuesta por:

$$F(n) = h(n) + g(n) \quad (1)$$

Siendo:

h(n): La heurística para el estado *n*.

g(n): El costo que lleva desde el estado de inicio hasta el estado *n*. Para este problema se asumió un costo de 1 para pasar entre estados.

Las heurísticas usadas son:

h1(n): Distancia de Manhattan.

$$h(s) = \sum_{i=1}^{8} (|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|). \quad (2)$$

Está usando Conflicto lineal ("Linear Conflict"), es una corrección a la Distancia de Manhattan para las piezas están adyacentes de la posición objetivo pero requieren que una baje para que se ubique la otra. Dado esto se penaliza con dos movimientos.

h2(n): Fichas mal colocadas ("Misplaced Tiles")

Indica cuantas fichas no están en la posición que debería estar.

Finalmente se calculó la heurística como:

$$h(n) = \max\big(h1(n), h2(n)\big) \quad (3)$$

El algoritmo en inicialmente fue hecho con una búsqueda en profundidad como es en realidad pero no converge tan rápido. Debido a esto fue necesario implementarlo con una cola de prioridad pero filtrando estados viejos luego de pasar un tamaño *M*. Esto con era para los tableros más grande que *7x7* para disminuir el gasto de memoria. El algoritmo es el siguiente:

Procedure $ida\_star(Root, N)$
$bound <-$ max(dM(Root), misplaced(Root))
$path <- search(root, 0, bound)$
return $path$
end Procedure

Procedure $search(Root,$ g, bound$)$
$f <- g + bound$
$pq <- \emptyset$
$path <- \emptyset$
$insert(pq, node, h, g, f)$
$insert(path, node, node)$
$//structure, sour, succ$
while ! isEmpty(pq) **then**
[node, g, h, f] $<-$ extractMin(pq)
//take min element whith min f $<-$ g + h
for succ **in** successors(node) **do**
if is_goal(succ) **then**
insert(path, node, succ)
break
else if ! isMarckNode(succ)**then**
h $<-$ max(dM(succ), misplaced(succ))
insert(pq, succ, h, g + 1, h + g + 1)
insert(path, node, succ)
end if
end for
end while
return $path$
end Procedure

extractMin(**pq**) Siendo *pq* la cola de prioridad esta función al calcula cual estado de *pq* tiene el menor F(n), lo retorna y es eliminado de *pq*.

dM(**succ**) Calcula la distancia de Manhattan y además penaliza las piezas con Conflicto lineal. El cálculo de las distancia de Manhattan ya está recalculado para cada ficha con el fin de disminuir el tiempo.

Para calcular los tableros aleatorios se usó esta mima lógica pero en inversa obligando a tomar las peores decisiones de esta forma se calcula tablero más complejos.

IV. INVARIANTES

Para cada ciclo de los dos algoritmos se cumple que los estados de la cola por evaluar son diferentes y cada vez que se evaluar un estado este es nuevo y no se ha calculado antes.

V. COMPLEJIDAD

La complejidad del primer algoritmo es:

$$armarGrafo(N) \rightarrow O(N!/2) \quad (4)$$

$$dijsktra(G, Root, \text{Goal}) \rightarrow O\big((V+E)Log(V+E)\big) \quad (5)$$

La complejidad para IDA* debido a que el problema es NP y la solución depende de una heurística esta hace que el algoritmo converja más rápido a la respuesta pero en caso de ser una mala heurística en el peor de los casos seria la misma complejidad de Dijkstra ecuación (5) porque el algoritmo de IDA* termina siendo una búsqueda por anchura con una cola de prioridad.

VI. RESULTADOS OBTENIDOS

Para los tamaños *2x2* y *3x3* como se esperaba funciono. Para los casos de 4x4 el algoritmo IDA* se demora un poco en encontrar la solución, para los siguiente casos se demora más de un minuto. Esto se debe la gran cantidad de combinaciones posibles, para forzar que el algoritmo llegue más rápido a la solución se multiplico la heurística $h(n)$ por la raíz cuadrad de ella misma para el caso de *5x5* y para los otros casos se multiplicó por la mima heurística. De esta forma se sacrifican un camino óptimo por uno no tan óptimo pero que es calculado en menor tiempo.

Tamaño	Tiempo en calcular la solución
4x4	2.6s
5x5	1.67s
6x6	2.611s
7x7	6.518s
8x8	34m 38s

Tabla 1. Pruebas del algoritmo IDA*

Se realizando pruebas al IDA* con diferentes tamaños y diferentes configuraciones de tableros, se probaron para cada tamaño 3 tablero y se tomaron 5 tiempos por cada configuración se calculó el promedio y están en la tabla 1. Para estas pruebas el tamaño máximo del tablero fue de *8x8* debido a que tarda mucho tiempo en llegar a la solución debido a esto no se realizó esta prueba con los tamaños 9x9 y 10x10 pero si se validó que el algoritmo sea capaz de resolver estos tablero.

Otra prueba realizada fue aplicar "Pattern +Database", en las heurísticas, para esto se usó como patrón ordenar primero los bordes superiores y laterales izquierdos de esta forma obligaba primeo ubicar los elementos del bode y luego los del mas centro. Esta heurística se probó con tamaños *4x4*, *5x5* y *6x6* tenía una alta convergencia pero se demoraba más y el camino era más largo. Se deseaba probar esta heurística para bajar el tiempo solución para los tableros *8x8*, *9x9* y *10x10*, pero las pruebas con el tablero de *7x7* demostraron que era más lento.

VII. CONCLUSIÓNES

Los algoritmos implementados con las heurísticas demostraron que si funcionan, como se esperaba a medida que aumenta el tamaño del tablero aumenta el tiempo que necesita el algoritmo para calcular el número de pasos que necesita el tablero para ser solucionado.

Para mejorar los tiempos de respuesta de tableros más grandes que *5x5* fue necesario multiplicar la heurística por un factor que sea proporcional o igual a misma heurística. Pero esta manera de forzar el algoritmo de busque que llegue a una solución más rápido, sacrifica caminos más óptimos. Además al probar esta método usando la heurística o de un orden más alto el comportamiento de la búsqueda ya no es la misma y puede que no converja tan rápido.

VIII. REFERENCIAS

[1] Asociación de Academias de la Lengua Española, «http://www.rae.es/,» Real Academia Española, 30 Mayo 2016. [En línea]. Available: http://dle.rae.es/?id=Z9td84q. [Último acceso: 30 Mayo 2016].

[2] L. Florez, «www.sophia.javeriana.edu.co,» 1 Marzo 2016. [En línea]. Available: https://sophia.javeriana.edu.co/mooc/pluginfile.php/4658/mod_resource/content/1/enunciado.pdf. [Último acceso: 30 Mayo 2016].

[3] Universidad de Pamplona, «http://www.unipamplona.edu.co/,» [En línea]. Available: http://www.unipamplona.edu.co/unipamplona/portalIG/home_23/recursos/general/11072012/grafo3.pdf. [Último acceso: 30 Mayo 2016].

[4] T. J. Misa , «An Interview With Edsger W. Dijkstra,» *Communications of the ACM*, vol. 53, nº 8, pp. 41-47, 2002.

[5] E. Benavent, «http://www.uv.es/,» [En línea]. Available: http://www.uv.es/benavent/apuntes-grafos.pdf. [Último acceso: 30 Mayo 2016].

IX. ANEXOS

Código:

https://github.com/bermeom/solve-taquin