Comparación de implementaciones de N-Puzzle

Daniel Min, Daniel Wohlgemuth

Universidad Nacional de Asunción, Paraguay

Resumen Uno de los objetivos de la inteligencia artificial en el área de problemas de búsqueda es poder diseñar algoritmos que puedan encontrar una solución incluso sin contar con algún precedente, en cuyo caso será difícil determinar si es la más óptima o no y, si se halla, será asumido como modelo inicial para versiones futuras mejoradas del mismo. En este documento mostraremos implementaciones de dos algoritmos de búsqueda, una con información y otra sin información acerca del problema a resolver y correremos dichos algoritmos para que resuelva un N-Puzzle, obtendremos algunas medidas de rendimiento de cada uno de ellos y demostraremos que aquellos que utilizan cierta información acerca del problema (heurística) es mejor que aquellos que no utilizan ninguna información.

Keywords: N-Puzzle, A*, Búsqueda en Anchura, Heurística.

1. Introducción

El N-Puzzle (El Juego de 15, entre otras denominaciones en español) es un juego de un solo jugador que se asume fue inventado por Noyes Chapman, un jefe de correos de Canastota, New York, cuya popularidad explosiva en Estados Unidos comenzó a inicios del 1880, disipándose en mayo del mismo año, según el resumen del libro The 15 puzzle: how it drove the world crazy[1]. Sin embargo, su invención es popularmente e incorrectamente atribuida a Samuel Loyd debido a que afirmó ser el inventor del juego desde 1891 hasta su muerte en 1911, lo que fue descubierto en la investigación minuciosa de los autores del libro mencionado.

En el área de inteligencia artificial, este juego atrae particular interés y preferencia de quiénes trabajan con problemas de búsqueda debido a que permite evaluar a los algoritmos de búsqueda en términos de la cantidad de estados del juego explorados y evaluados, antes de juzgar si es un estado objetivo o no, donde el estado del juego es el tablero obtenido con el siguiente movimiento o acción. En las secciones siguientes veremos con más detalle acerca de la mecánica del juego, sus propiedades y cómo exactamente contribuye en la medición de la eficiencia de algunos algoritmos de búsqueda en concreto, a saber, algoritmo de búsqueda en anchura, que es una técnica de búsqueda sin información acerca del problema (en este caso, el juego del 15) y algoritmo de búsqueda A* donde ya se incorporan ciertos conocimientos sobre el problema.

En la siguiente sección describimos la mecánica del juego y sus propiedades, en la sección 3 se explican los conceptos de los algoritmos implementados, la

sección 4 muestra los resultados de las mediciones realizadas con las diferentes implementaciones y finalmente se presenta la conclusión en la sección 5.

2. Descripción del Problema

Un N-Puzzle es un juego de rompecabezas que consiste en N piezas o fichas cuadradas numeradas y colocadas dentro de un marco o tablero cuadrado de tamaño $\sqrt{N+1} \times \sqrt{N+1}$ en donde un espacio queda sin ficha alguna. Este espacio es utilizado para mover las fichas adyacentes vertical u horizontalmente a fin de dejar el tablero con todas las fichas en orden de su numeración.

Además de esta simple mecánica de resolución (mover fichas adyacentes), también es interesante destacar que no todas las combinaciones posibles de las 15 fichas (más el hueco vacío) en el tablero es posible de resolver; exactamente la mitad de las 16! configuraciones posibles no tienen solución alguna, esto es, el tamaño del conjunto de estados del tablero que es posible alcanzarse a partir de la configuración correcta de fichas, es igual a 16!/2 [1]. Llamemos a estos estados como ***estados canónicos***. A pesar de que se ha implementado un algoritmo que comprueba la canonicidad del estado inicial del tablero, la demostración se deja fuera del contexto ya que no es relevante para el propósito de este documento.

Hasta ahora hemos hablado sobre las características del problema en particular que se utilizará como entrada para evaluar los diferentes algoritmos que se ha mencionado en el resumen. En la siguiente sección presentaremos los algoritmos que se implementan describiendo sus propiedades, diferencias y cómo mejorar el rendimiento de la búsqueda incorporando información adicional sobre el problema a la entrada del algoritmo.

3. Algoritmos implementados

Los algoritmos implementados para la comparación se pueden dividir en dos categorías: el algoritmo de búsqueda con información, que se explica en la siguiente subsección, y el algoritmo de búsqueda sin información, que será explicado más adelante.

A continuación se explican algunos conceptos básicos antes de introducir cada algoritmo.

Un nodo representa un estado del problema sobre el cual se pueda trabajar y aplicar alguna operación para conseguir sus siguientes estados, a los cuales denominamos **nodos hijos**. Cada **nodo padre** está conectado con sus nodos hijos directos a través de un camino con una **distancia** o **profundidad** 1. El valor 1 representa intuitivamente el costo de aplicar la operación, que para este trabajo es el desplazamiento de una pieza al espacio vacío, al nodo padre para llegar al nodo hijo directo. Se denomina **nodo raíz** al estado del cual se derivan todos los estados siguientes. Consideramos el estado inicial como la raíz en este trabajo. **Un árbol** es una estructura que tiene un nodo raíz y en donde de cada nodo padre se derivan cero o más nodos hijos. Un árbol posee la propiedad de

que existe un único camino de un nodo a otro nodo cualquiera, es decir, no existe redundancia de caminos.

3.1. Búsqueda con información

Búsqueda con información se refiere a que durante la exploración de las soluciones se tiene en cuenta alguna información acerca del problema para mejorar el rendimiento de la búsqueda en el caso medio. Para este trabajo utilizamos como método de búsqueda con información el algoritmo de búsqueda A*. En las subsecciones que siguen explicamos el algoritmo A*, las dos heurísticas que utilizamos para este algoritmo y cómo dedicir cuál heuristica es mejor, en el orden mencionado.

Búsqueda A* El algoritmo de búsqueda A* expande el árbol de búsqueda por el nodo con el menor costo total $f(n)$. $f(n)$ se compone de $g(n) + h(n)$, donde $g(n)$ representa el costo al que se incurre al recorrer el camino desde el nodo origen al nodo actual y $h(n)$ es la función heurística o estimación del costo desde el nodo actual hasta el nodo destino. Este algoritmo se puede implementar utilizando una cola de prioridad, una estructura de datos que siempre retorna el menor valor del conjunto de valores. Analizamos la ejecución del A* con dos heurísticas: la cantidad de piezas en posiciones incorrectas y la distancia de Manhattan.

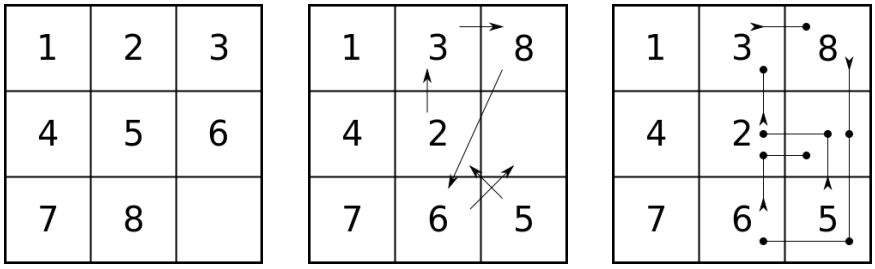
Cantidad de piezas en posiciones incorrectas Esta heurística cuenta las piezas que se encuentran en posiciones incorrectas, ignorando la pieza vacía. En la Figura 1b podemos identificar las piezas en posiciones incorrectas comparándola con la Figura 1a. Las flechas de la Figura 1b indican la posición objetivo en la cual cada pieza mal ubicada debería encontrarse. El valor de esta heurística para este ejemplo es 5.

Distancia de Manhattan Esta heurística suma las posiciones horizontales y verticales que se debería mover cada pieza, excepto la vacía, para ubicarla en su posición adecuada. Figura 1c muestra con lineas el camino que debería recorrer cada pieza, sin considerar colisiones con las demás piezas, para llegar a su posición objetivo. Los puntos sobre las líneas representan cada posición que la pieza visita al desplazarse por el camino. Para este ejemplo vemos que el valor de la heurística es 9, el cual se obtiene sumando los puntos de la figura.

Elegir la heurística dominate Supongamos que tenemos disponible varias heurísticas tanto admisibles como consistentes y no podamos determinar cuál de ellas es mejor, es decir, cuál heurística nos permite alcanzar al estado solución con mayor celeridad. Siendo admisible si nunca sobre estima el costo real y, consistente si se cumple $g(n) + h(n) \leq g(n') + h(n')$, donde n es un nodo cualquiera y n' es un nodo hijo de n.

Consideremos entonces una serie de heurísticas admisibles-consistentes $h_1, h_2, ..., h_m$ y sea n el estado actual en el que nos encontramos en el proceso de búsqueda con algún algoritmo de búsqueda informada. Diremos que h_i **domina a** h_j si $h_i(n) > h_j(n) \quad \forall (i,j) \in [1,2,...,m] \quad \& \quad i \neq j$.

Entones, si ninguna domina explícitamente a las demás, entonces para cada estado n_i analizado por el algoritmo, se utiliza $max\{h_k\} \quad \forall k \in [1,2,...,m]$.



(a) Estado objetivo (b) Piezas incorrectas (c) Distancia de Manhattan

Figura 1: Ilustración de las heurísticas utilizadas para A*

3.2. Búsqueda sin información

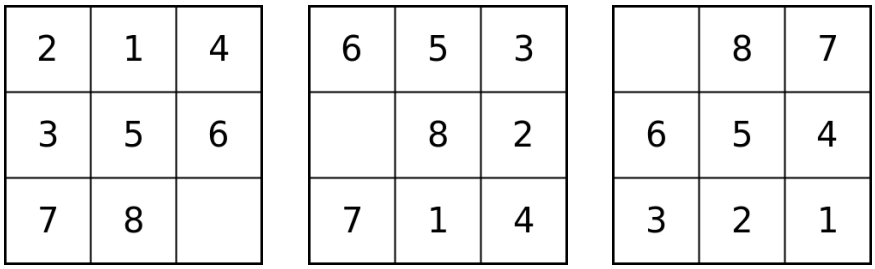
En la búsqueda sin información se ignora cualquier conocimiento sobre el problema que podría mejorar el rendimiento y se explora progresivamente todas las opciones posibles, comprobando en cada instante si se cumple la condición de finalización.

Búsqueda en anchura La búsqueda en anchura es una búsqueda sin información que evalúa todos los nodos de un nivel antes de evaluar los nodos del siguiente nivel. Esta búsqueda puede ser implementada con una cola simple en donde se extrae el siguiente estado a evaluar del inicio y se inserta los nuevos estados generados al final.

4. Resultados experimentales

Los tres métodos, búsqueda A* con distancia de Manhattan como heurística (h_2), búsqueda A* con cantidad de piezas incorrectas como huerística (h_1) y búsqueda en anchura, fueron ejecutados y evaluados para comparar los resultados y determinar la superioridad de un método u otro mediante pruebas experimentales. Las métricas utilizadas son el tiempo de ejecución en segundos y la cantidad de nodos explorados. En la Figura 2 se muestran los estados iniciales con los cuales se realizaron las mediciones. Los estados fueron elegidos de forma

experimental según el tiempo de ejecución de la búsqueda A* con distancia de Manhattan como heurística. La Figura 2a representa un estado simple para el cual la solución es encontrada en un tiempo reducido comparado con los demás. La Figura 2b muestra un estado que requiere un tiempo de procesamiento considerablemente mayor al anterior. La Figura 2c muestra un estado con todas las piezas en orden invertido comparado con el estado objetivo (Compárese con la Figura 1a), siendo éste el que necesita el mayor tiempo de cómputo.



(a) Estado de inicio simple (b) Estado de inicio moderado (c) Estado de inicio invertido

Figura 2

Los resultados se presentan en los Cuadros 1, 2 y 3 en la siguiente página y corresponden a los estados iniciales de las Figuras 2a, 2b y 2c respectivamente. Los tiempos medidos se promediaron sobre 10 ejecuciones de cada método con cada estado inicial. La máquina que utilizada para el experimento posee las siguientes características: Ubuntu 16.04.1, Python 3.5.2, Procesador i7-3820, 16GB RAM.

El Cuadro 1 muestra los resultados del estado de inicio simple. Cada método encontró lo solución a una profundidad de 12 pasos, siendo el A* con h_2 el que terminó en el menor tiempo y con el menor número de nodos explorados, seguido por el A* con h_1 con un tiempo levemente mayor y alrededor del doble de nodos explorados. La búsqueda en anchura requirió el mayor tiempo y la mayor cantidad de nodos explorados para encontrar la solución.

En el Cuadro 2 se encuentran los resultados del estado inicial moderado. La solución fue encontrada a una profundidad de 23 pasos por A* con h_2 y h_1. El A* con h_2 nuevamente presenta los mejores resultados. No se obtuvieron resultados para la búsqueda en anchura por falta de memoria.

Para el Cuadro 3 únicamente el A* con h_2 se ejecutó con éxito, encontrando una solución a una profundidad de 28 pasos. Los dos métodos restantes no pudieron finalizar por falta de memoria.

En cada caso el A* con h_2 presenta los mejores resultados, seguido por A* con h_1. La búsqueda en anchura por el contrario es la menos adecuada para

Cuadro 1: Resultados del estado de inicio simple

Método	Tiempo (s)	Explorados
A* con h_2	0,0066	136
A* con h_1	0,0073	263
Búsqueda en anchura	0,0553	16.490

Cuadro 2: Resultados del estado de inicio moderado

Método	Tiempo (s)	Explorados
A* con h_2	1,8661	39.235
A* con h_1	29,5300	918.411
Búsqueda en anchura	-	-

Cuadro 3: Resultados del estado de inicio inverso

Método	Tiempo (s)	Explorados
A* con h_2	5,9956	133.691
A* con h_1	-	-
Búsqueda en anchura	-	-

problemas de gran tamaño, por su elevado tiempo de búsqueda y consumo de memoria.

4.1. Nota acerca de la implementación

Una mejora realizada sobre los algoritmos implementados fue la de evitar el deslizamiento de la misma pieza en la dirección opuesta a la que se deslizó en el paso anterior. Esto se realizó porque es un movimiento que no mejora la solución, ya que se generaría un nodo duplicado con un costo mayor al del nodo padre de su nodo padre y además por tener una implementación sencilla. El Cuadro 4 muestra los resultados del estado inicial simple (Figura 2a) sin dicha restricción. Los tiempos de ejecución y los nodos explorados, comparado con los resultados del Cuadro 1, son considerablemente mayores, especialmente para la búsqueda en anchura. Con la restricción aplicada podemos ver una reducción del 86 % del tiempo de ejecución y una reducción del 82 % de los nodos explorados con el algoritmo A* con h_2. La búsqueda en anchura experimenta una mejora más pronunciada con la restricción. El tiempo de ejecución disminuye 98 % y el número de nodos explorados se reducen en un 97 %.

Cuadro 4: Resultados del estado de inicio simple sin la restricción

Método	Tiempo (s)	Explorados
A* con h_2	0,0503	765
A* con h_1	0,0940	2.663
Búsqueda en anchura	4,2207	678.283

5. Conclusiones

Dentro del alcance del experimento se ha demostrado la eficiencia de un algoritmo de búsqueda informado sobre uno no informado, algoritmo A* y Primero en Anchura respectivamente, tratando un mismo tipo de problema como el N-Puzzle. Sin embargo, el algoritmo A* no es la solución definitiva para todo problema de búsqueda; si la heurística acarrea pasos muy pequeños, es decir, dirige la búsqueda proyectando hacia el estado solución pero no es suficiente para cubrir la trayectoria en un tiempo razonable, este método sufre del mismo problema que el algoritmo de búsqueda en anchura con ramificación infinita o puede ocurrir lo contrario, podemos tener la mejor heurística de alguna forma pero si la cantidad de nodos a explorar en la trayectoria hacia el estado solución es exponencialmente grande, estaríamos ante el mismo problema. Por tanto, siempre es un desafío poder diseñar una heurística lo bastante buena que, además de ser

necesariamente admisible y consistente, que permita alcanzar al estado objetivo dentro de un tiempo prudencial y depende fuertemente del problema inicial de entrada.

También se observó y se comprobó que si se cuenta con más de una heurística, aquella que sea mayor o dominante permite mayor celeridad en la exploración y alcanzar el estado solución más rápidamente.

Además, se podría introducir posibles mejoras a los algoritmos implementados:

- **Evitar repetir estados ya generados.** Esto tal vez no tendría un gran efecto en las búsquedas informadas pero podría aumentar considerablemente el rendimiento de la búsqueda sin información.
- **Reemplazar búsqueda en ancho por profundización iterativa** para minimizar problemas de falta de memoria.

Referencias

1. Archer, A.: The 15 puzzle: how it drove the world crazy. The Mathematical Intelligencer 29.2, 83-85 (2007).
2. Ratner, D., Warmuth, M.: The (n2-1)-puzzle and related relocation problems. Journal of Symbolic Computation, 10(2), 111-137 (1990).
3. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, (1995).