Comparación de algoritmos de búsqueda en la resolución de los problemas 8-puzzle y TSP

Carlos Sanabria Miranda. UO250707@uniovi.es. Sistemas Inteligentes. Grado en Ingeniería Informática. EII. Universidad de Oviedo. Campus de los Catalanes. E-33007. Oviedo

Abstract. Tanto el 8-puzzle como el Travelling Salesman Problem (TSP) son problemas clásicos, que se pueden resolver mediante el paradigma de búsqueda en espacios de estados. El TSP también puede ser resuelto utilizando Algoritmos Genéticos. En este artículo se van a explicar dichos problemas, así como distintos algoritmos de búsqueda, tanto a ciegas (DFS, BFS, Coste Uniforme, …) como inteligente (A^*, PEA^*, …), con distintos heurísticos; así como el AG simple. Por último, se va a realizar un estudio experimental (utilizando el software aima-java) en el que se van a resolver dichos problemas con los algoritmos indicados, para así observar las diferencias entre los algoritmos y contrastar los resultados obtenidos con la teoría.

Keywords: búsqueda en espacios de estados, algoritmos de búsqueda a ciegas, algoritmos de búsqueda inteligente, 8-puzzle, TSP, A^*, algoritmos genéticos

1 Introducción

La búsqueda juega un papel fundamental en la IA (el núcleo de un Sistema Inteligente es un algoritmo de búsqueda). Los algoritmos de búsqueda a ciegas se quedan cortos a la hora de satisfacer las necesidades de problemas reales, dado que realizan búsquedas exhaustivas, y esto supone un gran coste temporal. Eso mismo es lo que se va a intentar plasmar en este artículo, observando como las diferencias entre los algoritmos a ciegas y los inteligentes son significativas.

Dentro de estos últimos, se van a comparar también distintos algoritmos, algunos admisibles (siempre encuentran la solución óptima) y otros simplemente completos (siempre encuentran solución, aunque no garantizan que sea óptima). El hecho de utilizar algoritmos inteligentes completos, pero no admisibles, viene dado por la necesidad de, en instancias muy grandes, obtener soluciones en tiempo y resultado razonable. Para A^* se van a comparar también los heurísticos utilizados, teniendo en cuenta su admisibilidad y consistencia, y analizando cuales están mejor informados.

Por último, se va a analizar también el AG simple, comparándolo con A^* en la resolución del problema del TSP. Este algoritmo tiene su utilidad para instancias grandes, en las que los algoritmos admisibles se quedan cortos, y puede presentar una alternativa a otros algoritmos no admisibles, como el PEA^*.

Para realizar el estudio experimental se va a utilizar el software *aima-java*, que implementa los algoritmos descritos en [1].

2 Algoritmos de Búsqueda a Ciegas

2.1 Algoritmos de Búsqueda a Ciegas iterativos

Si el espacio de búsqueda es un árbol, la implementación de los siguientes algoritmos se basa en el Algoritmo General de Búsqueda en Árboles (AGBA), mientras que, si es un grafo, se basa en el Algoritmo General de Búsqueda en Grafos (AGBG). Ambos se encuentran bien explicados en [2].

Los dos utilizan las siguientes estructuras de datos: Abierta/Frontera (lista de nodos candidatos a ser expandidos) y Tabla_A (conjunto que almacena, entre otras cosas, los nodos ya visitados, y permite reconstruir el camino desde el inicial al objetivo).

2.1.1 Depth First Search o Búsqueda Primero en Profundidad

Abierta es una pila. Si el espacio de búsqueda es un árbol, se repiten estados, por lo que éste tendrá ramas infinitas, de longitud infinita; y lo más probable es que no encontremos solución. Si es un grafo, no se repiten estados, y este algoritmo será completo, aunque en el peor de los casos tendrá que recorrer todo el espacio de búsqueda.

2.1.2 Breadth First Search o Búsqueda Primero en Anchura

Abierta es una cola. Tanto en árboles como en grafos, el algoritmo es completo.

2.1.3 Coste uniforme

Abierta se ordena en función del valor del coste de cada nodo al inicial (almacenado en la Tabla_A). Tanto en árboles como en grafos, el algoritmo es admisible.

2.2 Algoritmos de Búsqueda a Ciegas recursivos

Están basados en el algoritmo de Backtracking, muy bien descrito en [3]. El espacio de búsqueda ha de ser obligatoriamente un árbol. Son más eficientes en memoria.

2.2.1 Búsqueda en profundidad limitada

Establece una profundidad límite, más allá de la cual no va a comprobar nodos. De esta forma, el algoritmo se asegura terminar, pero es posible que no encontremos una solución si ésta se encuentra a una profundidad mayor que la límite.

En caso de encontrar una solución, tampoco garantiza que sea óptima.

2.2.2 Búsqueda en profundidad iterativa

Realiza continuas llamadas al algoritmo anterior, dándole un valor a la profundidad límite, que se incrementa en cada llamada. Gracias a esto, el algoritmo es completo, pero el tiempo de ejecución del algoritmo es mayor que en el caso anterior (es la suma de los tiempos de cada una de las diversas llamadas al algoritmo de Búsqueda en profundidad limitada).

3 Algoritmos de Búsqueda Inteligente

3.1 El algoritmo A*

Fue propuesto por Peter E. Hart, Nils J. Nilsson y Bertram Raphael en [4] y está muy bien descrito en [5] y [6]. Sus propiedades formales están bien explicadas en [2].

Se trata de un caso particular del algoritmo Best First (BF)[1], en el que $f(n) = g(n) + h(n)$. $g(n)$ representa el mejor coste del inicial a n hasta el momento, y $h(n)$ es una estimación positiva del coste mínimo de n al objetivo más próximo a n. La función h hay que definirla de forma que incorpore conocimiento sobre el dominio del problema. Su coste no es despreciable, así que sería conveniente que fuera polinomial.

En el prototipo tenemos 3 versiones (normal, rectifica y reinserta), descritas en [2].

3.2 Ponderación estática PEA*

Es una variante del algoritmo A^*, en la que $f(n) = g(n) + (1+\varepsilon)\, h(n)$, siendo ε un valor pequeño ($\varepsilon \geq 0$). Se engloba dentro de la categoría de algoritmos ε-admisibles (aquellos que encuentran con seguridad o con una alta probabilidad una solución cuyo coste no excede $(1+ \varepsilon)\, C^*$), siendo C^* el coste de la solución óptima. Este tipo de algoritmos renuncian a la admisibilidad (dado que en algunas circunstancias se puede cumplir que $(1+\varepsilon)\, h(n) > h^*(n)$) en beneficio de obtener una mejora en el tiempo de ejecución del algoritmo, y acotando el coste de la solución obtenida mediante ε.

3.3 Greedy Best-First Search

Es otra variante de A^*, en la que $f(n) = h(n)$. Se trata de un algoritmo voraz y no admisible, que no permite establecer ninguna cota sobre la calidad de la solución.

3.4 Algoritmos genéticos

Se basan en la probabilidad y en la evolución natural. Las soluciones se codifican mediante cadenas de símbolos, denominadas cromosomas. Se parte de una población inicial de cromosomas de un determinado tamaño, y se la hace evolucionar a lo largo de un cierto número de generaciones (mediante el operador de cruce se generan cromosomas nuevos a partir de los ya existentes, intentando conservar características relevantes de padres a hijos). Para evaluar la calidad de un cromosoma, se decodifica (se transforma en solución) y se le aplica la función de fitness que, en principio, es la única que incorpora conocimiento sobre el problema. Están bien descritos en [7].

4 El problema del 8-puzzle

Una breve descripción de este problema se encuentra en [8].

[1] BF se basa en el AGBG. Ordena Abierta por el valor de la función f (en sentido creciente).

4.1 Enunciado del problema

Se parte de un tablero de tamaño 3x3 en el que hay 8 baldosas, con números del 1 al 8, y una posición vacía. Una baldosa sólo se puede mover a una posición vacía si ésta es adyacente ortogonalmente. El objetivo es llegar a una situación del tablero en la que cada número esté situado en una posición específica.

2		3
1	8	4
7	6	5

Estado inicial

1	2	3
8		4
7	6	5

Estado objetivo

4.2 Modelado en el marco de búsqueda en espacios de estados

Cada estado representa una situación del tablero válida. Los estados inicial y objetivo coinciden con las situaciones inicial y objetivo del tablero, respectivamente.

Las reglas indican hacia donde se mueve la casilla vacía, y permiten pasar de un estado a otro. El coste de cada regla es siempre 1, por tanto, el coste del camino equivale al número de arcos en el mismo.

4.3 Heurísticos

$h_0(n) = 0, \forall n$. Es el heurístico nulo. Es admisible y consistente.

$h_1(n)$ = número de fichas que están descolocadas con respecto al estado final. Es admisible y consistente.

$h_2(n)$ = suma de distancias ortogonales de cada ficha a su posición final (distancia Manhattan). Es admisible y consistente.

$h_3(n) = \sum (2 \cdot$ número de fichas a distancia ortogonal 2 de su posición final). Es admisible, pero no consistente.

$h_{1\text{-}2}(n)$ = para cada ficha, cuenta 1 si no está en la fila y 1 si no está en la columna que le corresponden en el objetivo. Es admisible y consistente.

$h_4(n) = 1.31\ h_2(n) - 2.19$. No es admisible ni consistente.[2]

5 Estudio experimental

Para comparar los resultados de los algoritmos de búsqueda en espacios de estados, se va a utilizar: el número de nodos expandidos, como medida del tiempo de resolución[3], y el coste de la solución que encuentran (en caso de que el algoritmo termine).

---

[2] La intuición del heurístico h_4 viene dada por la siguiente fórmula: $y = 1.31x - 2.19$, que representa una regresión lineal que intenta encontrar una relación entre el valor de h_2 (distancia Manhattan) y el de h^*, para instancias del 15-puzzle. Su obtención está bien descrita en [10].

[3] Esto se debe a que el tiempo no es una medida objetiva, ya que depende de la máquina en la que se realicen las pruebas y, además, puede variar ligeramente entre cada ejecución.

5.1 Banco de ejemplos

El banco de ejemplos utilizado está compuesto por instancias para las cuales se conoce el coste de la solución óptima. Para cada tipo de coste se dispone de 5 instancias, para las cuales se calculará la media de los resultados.

Table 1. Instancias del 8-puzzle utilizadas como banco de ejemplos.

Coste	Instancias				
	1ª instancia	2ª instancia	3ª instancia	4ª instancia	5ª instancia
5	1,0,3,8,2,5,7,4,6	1,4,2,0,8,3,7,6,5	1,3,4,8,6,2,7,0,5	1,2,3,7,8,0,6,5,4	1,4,2,8,3,0,7,6,5
10	8,2,1,7,0,3,6,5,4	1,4,0,8,5,2,7,3,6	8,1,3,7,0,5,4,2,6	8,1,2,4,0,6,7,5,3	0,1,3,7,2,5,4,8,6
15	4,8,2,6,3,5,1,0,7	1,4,5,2,7,0,8,6,3	1,3,8,6,7,4,2,0,5	2,0,8,7,5,3,4,1,6	7,1,3,4,5,0,8,2,6
20	6,2,7,4,5,1,0,8,3	4,7,2,1,0,6,3,5,8	7,1,5,4,0,8,2,6,3	5,1,6,4,0,3,8,7,2	7,1,4,5,0,6,3,2,8
25	6,7,4,0,5,1,3,2,8	6,0,7,5,4,1,3,8,2	3,4,8,5,7,1,6,0,2	4,5,3,7,6,2,8,0,1	2,7,8,5,4,0,3,1,6
30	5,6,7,2,8,4,0,3,1	5,6,7,4,0,8,3,2,1	5,4,7,6,0,3,8,2,1	3,8,7,4,0,6,5,2,1	5,6,3,4,0,2,7,8,1

5.2 Resultados experimentales

5.2.1 Algoritmos de búsqueda a ciegas

5.2.1.1 Espacio de búsqueda representado mediante un grafo

Table 2. Media del coste (coste obt.) y número de nodos expandidos (#nodos exp.) obtenidos por DFS, para todas las instancias del 8-puzzle, cuando el espacio de búsqueda es un grafo.

Resultados	Coste de las instancias					
	5	10	15	20	25	30
coste obt.	51149,8	89034	67637,8	49424,4	49285	57537,6
#nodos exp.	55686	95860,6	74636,4	51747,8	52027,2	60724,4

La Tabla 2 muestra los resultados para el algoritmo DFS. Como es esperable, el algoritmo siempre encuentra una solución, pero en todas las instancias evaluadas dicha solución difiere muchísimo de la óptima y, además, incluso para instancias de coste bajo, expande muchos nodos, en comparación con los siguientes algoritmos.

Table 3. Media del número de nodos expandidos por BFS y Coste Uniforme (Coste Unif.), para todas las instancias del 8-puzzle, cuando el espacio de búsqueda es un grafo.

Algoritmo	Coste de las instancias					
	5	10	15	20	25	30
BFS	26	393	4699,2	38396	116083,2	180874,4
Coste Unif.	46,6	697,2	5910,4	53585,8	133927,2	181365,8

La Tabla 3 muestra los resultados para BFS y Coste Uniforme. Ambos dan siempre con la solución óptima, y expandiendo muchos menos nodos que el DFS. Esto se debe

a que, en el caso del 8-puzzle, BFS además de completo es admisible[4], ya que los costes de todas las reglas son siempre 1.

También se observa que, a medida que aumenta el coste de la solución óptima, va siendo necesario expandir más nodos para llegar a ella. Para instancias de coste 30, prácticamente es necesario visitar todo el espacio de búsqueda[5].

Aunque se observe un menor número de nodos expandidos en BFS que en Coste uniforme, esto es debido a la implementación de dichos algoritmos.[6] De todas formas, son diferencias que, para instancias de coste grande, son insignificantes.

5.2.1.2 Espacio de búsqueda representado mediante un árbol

Con el algoritmo DFS no se encuentra solución para ninguna instancia, dado que el espacio de búsqueda es infinito, y el algoritmo cae por una rama de longitud infinita. Es probable que no encuentre solución para ninguna instancia del 8-puzzle.

Como en el caso de los grafos, los algoritmos BFS y Coste uniforme encuentran una solución óptima, pero en este caso expanden muchos más nodos, debido a la repetición de estados, que no se da en el caso de los grafos.

Table 4. Media del número de nodos expandidos por BFS y Coste Uniforme (Coste Unif.), para todas las instancias del 8-puzzle, cuando el espacio de búsqueda es un árbol.

Algoritmo	Coste de las instancias					
	5	10	15	20	25	30
BFS	65,6	13765,4	2620776	No termina, se queda sin memoria		
Coste Unif.	247,4	41752,8	No termina, se queda sin memoria			

La Tabla 4 muestra los resultados para BFS y Coste Uniforme. En el caso del BFS, con instancias de coste 20 o superior el algoritmo no es capaz de terminar, debido a que se agota la memoria disponible para su ejecución. Lo mismo sucede con el algoritmo Coste Uniforme para instancias de coste 15 o superior.

Table 5. Media del coste (coste obt.) y número de nodos expandidos (#nodos exp.) obtenidos por el algoritmo de búsqueda en profundidad limitada, para instancias del 8-puzzle de coste 15, y valores de profundidad límite de 14, 15, 16 y 17.

Resultados	Profundidad límite			
	14	15	16	17
coste obt.	-	15	15	17
#nodos exp.	No encuentra solución	2235212,6	6502423,2	3553424,2

La Tabla 5 muestra los resultados del algoritmo de búsqueda en profundidad limitada. Se observa como, para profundidades menores de 15, el algoritmo no encuentra

[4] Esto no tiene por qué ser así para todos los problemas, ya que los nodos a la misma profundidad en el árbol no tienen por que tener caminos hasta el inicial del mismo coste.

[5] Cuando se representa mediante un grafo, tiene un tamaño de 9!/2 = 181440 nodos.

[6] A cómo se ordena Abierta en caso de empates en el coste de los caminos (y como los costes de todas las reglas son siempre 1, se producen muchos empates).

solución; para profundidad 15, encuentra la solución óptima; y para profundidades mayores de 15, encuentra soluciones, pero que no tienen por qué ser óptimas[7]. Además, expande una gran cantidad de nodos (similar a BFS), pero no agota la memoria.

El algoritmo de búsqueda en profundidad iterativa, para coste 15, expande casi el doble de nodos que el de búsqueda en profundidad limitada con profundidad límite 15. Pero, aunque expanda más nodos, es un número de nodos del mismo orden de complejidad, y para el 8-puzzle encuentra siempre el óptimo, en contrapartida con el de profundidad limitada, que sólo lo encuentra si la profundidad límite coincide con el del coste óptimo. Para instancias de coste 20 o superior, ambos tardan demasiado.

5.2.2 Algoritmos de búsqueda inteligente

5.2.2.1 Algoritmo A*

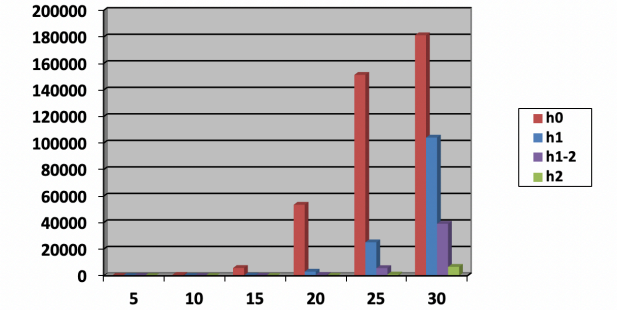

Fig. 1. Media del número de nodos expandidos (eje y) por A* (versión que ni rectifica ni reinserta) con los heurísticos h_0, h_1, $h_{1\text{-}2}$ y h_2, para instancias de cada coste del 8-puzzle (eje x).

La Figura 1 muestra los resultados para A* con los heurísticos h_0, h_1, $h_{1\text{-}2}$ y h_2. Los 4 son admisibles y consistentes, por lo que con esta versión de A* encuentran el óptimo. Se puede observar como h_0 es el heurístico que más nodos expande, h_1 expande menos que h_0, $h_{1\text{-}2}$ expande menos que h_1 y, por último, h_2 expande menos que $h_{1\text{-}2}$. Esto es debido a que $h_0 < h_1 \leq h_{1\text{-}2} \leq h_2$ para cualquier nodo, excepto los nodos finales.

La dominancia nos dice que, si $h_x < h_y \leq h^*$ para todo nodo no final, entonces todo nodo expandido por $A^*(h_y)$ es expandido también por $A^*(h_x)$. Dado que $h_0 < h_1$, en la práctica vemos como h_1 expande menos nodos que h_0, sobre todos si el coste óptimo de las instancias es grande. Para que se pueda cumplir esta propiedad formal, la relación de dominancia debe ser estricta, por lo que para el caso de $h_1 \leq h_2$, la teoría no nos asegura que con $A^*(h_2)$ se vayan a expandir menos nodos que con $A^*(h_1)$ (incluso se podría dar el caso de que fuera al revés). Sin embargo, como son monótonos, la dominancia amplia sí que nos da un cierto grado de confianza de que h_2 es mejor que h_1. En la práctica, vemos como h_2 siempre expande menos nodos que h_1, y lo mismo sucede entre $h_{1\text{-}2}$ y h_1 (ya que $h_1 \leq h_{1\text{-}2}$), y entre h_2 y $h_{1\text{-}2}$ (ya que $h_{1\text{-}2} \leq h_2$).

[7] Esto se debe a que, en el 8-puzzle, el coste de los arcos es 1, por lo que, si el coste de la solución óptima es 15, ésta estará a profundidad 15. Por tanto, a profundidades menores no habrá ninguna solución, y a profundidades mayores las soluciones serán no óptimas.

Table 6. Media del coste (coste obt.) y nº de nodos expandidos (#nodos exp.) por A* (versión que ni rectifica ni reinserta) con el heurístico h_3, para todas las instancias del 8-puzzle.

Resultados	Coste de las instancias					
	5	10	15	20	25	30
coste obt.	5	10	15	20,4	25	30,4
#nodos exp.	16,2	107,4	791	9978,4	41777,4	129483,2

La Tabla 6 muestra los resultados para A^* con h_3, que es admisible, pero no consistente. Debido a su no consistencia, la versión que no rectifica ni reinserta de A^* no garantiza que se vaya a encontrar la solución óptima utilizando dicho heurístico, aunque sea admisible. En la práctica, vemos como para muchas instancias sí que encuentra la solución óptima, pero hay algunas para las que no (como la 3ª de coste 20).

Table 7. Media del número de nodos expandidos por A^* con las versiones que rectifican y reinsertan y los heurísticos h_2 y h_3, para todas las instancias del 8-puzzle.

Algoritmo	Coste de las instancias					
	5	10	15	20	25	30
Reinserta h_2	5	14,6	61,4	146,4	1284	6732
Reinserta h_3	16,2	109,2	749,2	8138,4	51653,4	131468,6
Rectifica h_2	5	14,6	61,4	146,4	1284	6732
Rectifica h_3	16,2	109,2	789,2	8106,6	43818,4	128055,2

La Tabla 7 muestra los resultados para las otras dos versiones de A^*, con h_2 y h_3. Como h_2 es un heurístico consistente, nunca es necesario reinsertar o rectificar nodos ya expandidos, por lo que con cualquier versión de A^* da el óptimo (y debería expandir el mismo número de nodos, aunque en las pruebas varía ligeramente debido a la implementación de los algoritmos). Sin embargo, con h_3 no sucede esto, dado que no es consistente. Con la versión que reinserta o con la que rectifica los sucesores encuentra siempre la solución óptima, a diferencia de con la versión normal. Con h_3 también se observan pequeñas diferencias en los nodos expandidos en estas dos versiones.

Table 8. Media del número de nodos expandidos por A* con el heurístico h_2 cuando el espacio de búsqueda (Esp. búsq.) es un grafo (con la versión que ni rectifica ni reinserta) y cuando el espacio de búsqueda es un árbol, para todas las instancias del 8-puzzle.

Esp. búsq.	Coste de las instancias					
	5	10	15	20	25	30
Grafo	5	14,6	61,2	157,6	989,8	6746,2
Árbol	5	14,8	347,8	794	63500,2	479860,5

La Tabla 8 compara los resultados de A^* con h_2 en función de la representación del espacio de búsqueda. La representación en forma de grafo es muy superior. Se puede observar como, para instancias de coste 30, expande 71 veces menos nodos.

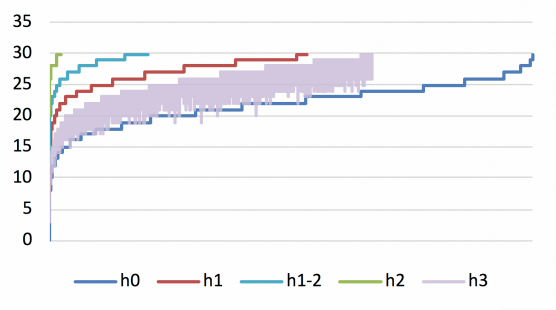

Fig. 2. Valores de f(n) (eje y) para cada uno de los nodos expandidos (eje x) por A* (versión que rectifica) con los heurísticos h_0, h_1, $h_{1\text{-}2}$, h_2 y h_3 para la 1ª instancia de coste 30 del 8-puzzle.

La Figura 2 muestra una comparativa de los valores de f(n), para cada uno de los nodos expandidos por A* (versión que rectifica) con los distintos heurísticos, para la 1ª instancia de coste 30. Esta información permite determinar, de forma experimental, cuando un heurístico no es consistente. Si para un heurístico el valor de f(n) decrece alguna vez, ese heurístico es no consistente. Sin embargo, que dicho valor no decrezca no nos asegura que sea consistente (puede serlo o no). Como se puede observar, el valor de f(n) utilizando h_3 decrece muchas veces, por lo que nos permite confirmar experimentalmente que no es consistente. Además, cuanta menor sea la distancia entre el valor de f(inicial) y f(objetivo), mejor será el heurístico (por ejemplo, h_0 parte del valor 0, mientras que h_2 parte del 20, y sabemos que h_2 es mejor que h_0).

5.2.2.2 Algoritmo PEA*

Table 9. Media del coste (coste obt.) y número de nodos expandidos (#nodos exp.) obtenidos por PEA*, con el heurístico h_2, para instancias del 8-puzzle de coste 25 (primera mitad de filas) y 30 (segunda mitad), dándole a ε los siguientes valores: 0, 0.1, 0.2, 0.3, 0.5, 1, 2 y 5.

Resultados	Valor de ε							
	0 (A*)	0.1	0.2	0.3	0.5	1	2	5
coste obt.	25	25	25	25,4	25,8	28,6	30,6	41
#nodos exp.	1284	689,6	688	698,4	522	444,8	295,4	370
coste obt.	30	30	30,4	30,4	30,4	31,2	36	50
#nodos exp.	6732	5292	4342,2	3092	1535,4	1399,6	478,8	400,4

La Tabla 9 muestra los resultados para PEA* dándole distintos valores a ε. No hay ningún resultado teórico que asegure que al aumentar ε disminuya el numero de nodos expandidos. Puede ocurrir que aumente el coste y además el número de nodos expandidos, como se observa al pasar de ε = 0.2 a ε = 0.3 en las instancias de coste 25. Para ε = 5, los costes obtenidos difieren mucho del óptimo, dado que el algoritmo se aleja mucho de A*. Aún así, se sigue cumpliendo que el coste está acotado por (1+ ε) C*, como se puede observar, por ejemplo, en las instancias de coste 30 con ε = 5: (1+5)30=180 ≥ 50. A la vista de los resultados, un buen valor de ε podría estar entre 0.1 y 0.5.

5.2.2.3 Otros algoritmos no admisibles

Table 10. Media del coste (coste obt.) y número de nodos expandidos (#nodos exp.) obtenidos por A^* (versión que reinserta) con el heurístico h_4 (primera mitad de filas); y por Greedy Best-First Search con el heurístico h_2 (segunda mitad de filas), para todas las instancias del 8-puzzle.

Resultados		Coste de las instancias					
		5	10	15	20	25	30
$A^* h_4$	coste obt.	5	10	15	20	25,8	30,4
	#nodos exp.	5	11,2	50,4	124,8	697,6	3011,4
GBF h_2	coste obt.	5	16,4	32,2	45,6	66,6	52,4
	#nodos exp.	5	39	172,8	152,8	306,4	218,6

La Tabla 10 muestra los resultados para A^* (versión que reinserta) con h_4, y para GBF con h_2. Se puede ver como GBF se comporta bastante mal, dado que, aunque para instancias grandes expande menos nodos, los costes de las soluciones se alejan mucho del óptimo (no hay ninguna cota). Mientras tanto, A^* con h_4 funciona mucho mejor, dado que apenas se aleja del óptimo y, para instancias de coste 30, expande la mitad de nodos que A^* con h_2, y tiene un comportamiento similar al PEA^* con $\varepsilon = 0.3$ (aunque este nos asegura que el coste de la solución no será mayor de $1.3 \cdot 30 = 39$).

5.2.2.4 Algoritmo A^* y algoritmos genéticos con el problema del TSP

Table 11. Número de nodos expandidos por A^* (con la versión que ni rectifica ni reinserta) y los heurísticos h_0, $h_{sum\_min\_arcs}$ y h_{MST}, para todas las instancias del TSP (son las incluidas en el prototipo aima-java: las 3 primeras son simples y la última es la gr17 de la TSPLIB).

Algoritmo	Coste de las instancias			
	21	85	115	2085
h_0	81	55	28	524289
$h_{sum\_min\_arcs}$	44	36	21	411108
h_{MST}	16	28	12	30891

La Tabla 11 muestra los resultados para A^* y el problema del TSP (bien descrito en [9]), utilizando los heurísticos h_0 (heurístico nulo), $h_{sum\_min\_arcs}$ (sumatorio de los arcos mínimos que tocan a la última ciudad visitada y a cada ciudad no visitada) y h_{MST} (calcula el árbol de expansión mínimo), todos consistentes (los dos últimos podemos asegurar que lo son, dado que fueron obtenidos a través del método de la relajación del problema). Además, sabemos que h_{MST} está mejor informado que $h_{sum\_min\_arcs}$, dado que h_{MST} utiliza un subconjunto de relajaciones del problema de las que utiliza $h_{sum\_min\_arcs}$. Se puede ver como h_{MST} expande muchos menos nodos que h_0 y que $h_{sum\_min\_arcs}$. Aunque $h_{sum\_min\_arcs}$ expande menos nodos que h_0, tarda mucho más tiempo, dado que calcular el heurístico $h_{sum\_min\_arcs}$ para cada nodo visitado tiene un coste temporal mayor que h_0, que es $\theta(1)$. h_{MST} tiene también un coste temporal similar a $h_{sum\_min\_arcs}$, pero es el heurístico que menos tiempo tarda globalmente, dado que el coste temporal del heurístico se compensa con el poco número de nodos que expande.

Table 12. Media (de 10 ejecuciones) del coste del mejor individuo de la población inicial (Coste pobl. inicial), coste del mejor individuo de la población final (Coste pobl. final) y tiempo en milisegundos (Tiempo (ms)) obtenidos por el algoritmo genético con el operador de cruce en dos puntos OX (AG op. cruce b.), y con el operador de cruce malo y sin escalar los fitness (AG op. cruce m.), para instancias del TSP de coste 115 (instancia simple) y 2085 (gr17 de la TSPLIB).

Tipo AG e instancias		Resultados		
		Coste pobl. inicial	Coste pobl. final	Tiempo (ms)
AG op. cruce b.	Coste 115	115	115	9054,5
	Coste 2085	3453	2311,4	40348
AG op. cruce m.	Coste 115	157	136	6160,8
	Coste 2085	3481,9	3481,9	15255,6

La Tabla 12 muestra los resultados del problema del TSP al utilizar AG con las siguientes características: codificación con permutaciones, población inicial aleatoria, tamaño de la población y número de generaciones 150, probabilidad de cruce 1 y de mutación 0.2, dos operadores de cruce distintos (cruce en dos puntos y cruce malo (no traslada a los hijos características relevantes)), operador de selección por ruleta (con escalado del fitness, salvo para cruce malo) y con elitismo. Con el operador de cruce en dos puntos los resultados son aceptables: para la instancia de coste 115 encuentra el óptimo en la población inicial, mientras que para la instancia de coste 2085, el coste medio obtenido en la población inicial es de 3453, y termina con un coste medio de 2311,4. Con el operador malo los resultados son mucho peores: para la instancia de coste 115 parte de un coste de 157 y termina con un coste de 136, mientras que para la de 2085 parte de un coste de 3481,9 y no consigue ninguna mejora (no converge).

Si comparamos los resultados del AG (con el operador de cruce en dos puntos) con los de A*, vemos como para la instancia de coste 2085 (17 ciudades) A* tarda menos tiempo, y encima encuentra la solución óptima. Pero, para instancias más grandes, seguramente no sea viable utilizar el algoritmo A*, dado que tarde demasiado tiempo, y sea más recomendable utilizar un algoritmo genético que, aunque no garantiza encontrar la solución óptima, sí que da una solución cercana en un tiempo aceptable.

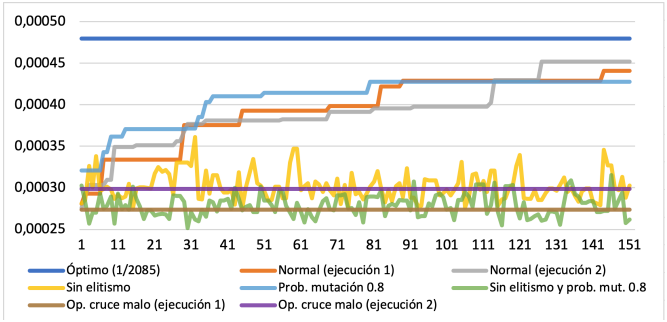

Fig. 3. Valores del fitness (eje y) del mejor individuo de cada generación (eje x), en AGs con distintos parámetros, para la instancia de coste 2085 del TSP. "Normal" utiliza los parámetros descritos previamente, con el op. cruce en dos puntos, y "Óptimo" es el fitness óptimo.

La Figura 3 compara la evolución del AG con distintos parámetros para la instancia de coste 2085. El AG con los parámetros descritos previamente converge hacia soluciones óptimas. Incluso el AG con probabilidad de mutación de 0.8 converge (se aconseja que sea baja, para no obtener una búsqueda aleatoria), gracias al elitismo. En ambos, se puede ver como a partir de la generación 100 casi no se obtienen mejoras. Sin embargo, sin elitismo perdemos esa convergencia, y los resultados se empeoran con la probabilidad de mutación de 0.8. Con el operador de cruce malo tampoco hay convergencia, y el coste de las soluciones no varía a lo largo de las generaciones.

6 Conclusiones

Podemos comprobar como, para el 8-puzzle, la mejor representación del espacio de búsqueda es en forma de grafo, y si tuviéramos que quedarnos con un algoritmo de búsqueda a ciegas para este problema (con las impl. del prototipo), sería el BFS.

También queda demostrada la superioridad de los algoritmos inteligentes sobre los algoritmos a ciegas (p.ej. A^* con h_2 expande casi 27 veces menos nodos que BFS).

Los resultados permiten recalcar la importancia de los heurísticos elegidos en A^* y sus variantes. Deberían ser admisibles y, si es posible, consistentes. Si entre dos heurísticos se da una relación de dominancia estricta, entonces en la práctica uno de ellos va a expandir menos nodos. Normalmente, también va a darse esta situación aunque la relación no sea estricta, sobretodo si ambos son consistentes. Si el heurístico elegido es admisible pero no consistente, deberíamos usar la versión de A^* que rectifica o la que reinserta para obtener soluciones óptimas (aunque en la práctica, si usamos la versión normal, en muchas ocasiones sí alcanza la solución óptima).

Para el TSP, h_{MST} es el mejor heurístico encontrado, y el AG simple con codificación con permutaciones permite obtener soluciones en tiempo y coste aceptables.

Referencias bibliográficas

1. Pearl, J., Heuristics, Morgan Kauffman, San Francisco, CA, 1983.
2. Palma Méndez, J. T. y Marín Morales R. Inteligencia Artificial: Técnicas, métodos y aplicaciones, McGraw-Hill, Murcia, 2008.
3. Steven S. Skiena, Miguel A. Revilla. Programming challenges: The Programming Contest Training Manual, Springer, New York, 2003.
4. Korf, Richard. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search". Artificial Intelligence 27: 97–109 (1985).
5. Peter E. Hart, Nils J. Nilsson & Bertram Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics 4(2):100–107, 1968.
6. Nilsson, N., Principles of Artificial Intelligence, Tioga, Palo Alto, CA, 1980.
7. Kramer, O. Genetic Algorithm Essentials, Springer, 1st ed. 2017.
8. Russell, S. and Norvig P. Artificial Intelligence, A Modern Approach, Prentice Hall, New Jersey, 3th ed. 2010.
9. Brucato C. The Traveling Salesman Problem. (2013).
10. Stern, R., Felner, A., Berg, J., Puzis, R., Shah, R., Goldberg, K., Potential-based bounded-cost search and Anytime Non-Parametric A∗, Artificial Intelligence 214, 1-25, (2013).