Práctica 3: Búsqueda Informada

1. Objetivos

El objetivo es profundizar en algunos conceptos de la búsqueda en espacios de estados, con la aplicación de algoritmos de búsqueda no informada e informada (en concreto, el A*) a un problema concreto mediante la propuesta de heurísticos, desarrollo de código en Java y obtención y análisis de resultados experimentales.

Los alumnos han de elaborar y defender una práctica sobre algoritmos de búsqueda aplicados al problema (ya conocido) presentado en la Sección 2. Más concretamente, se pide:

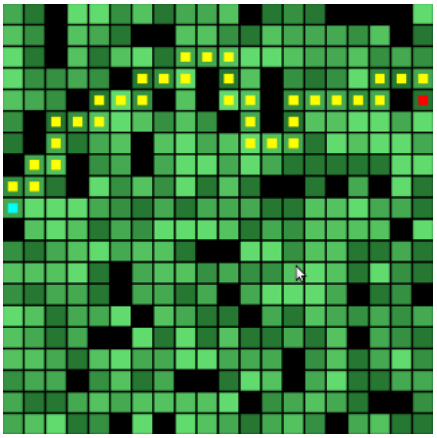
1. Recordar la formalización del problema como un problema de búsqueda en espacio de estados.
2. Reflexionar sobre la admisibilidad y consistencia del heurístico Manhattan, así como de otro propuesto por los alumnos.
3. Desarrollar el árbol de búsqueda que se obtiene para el ejemplo de juguete proporcionado en este guión utilizando el heurístico Manhattan.
4. Completar el código fuente proporcionado para resolver el problema.
5. Elaborar una memoria para entregar junto con el código fuente.

2. El problema de búsqueda de rutas en cuadrículas

Recordemos que el problema de búsqueda de rutas en cuadrículas (*grid-based pathfinding*) consiste en lo siguiente:

- Un agente tiene que recorrer un mundo bidimensional en forma de cuadrícula de tamaño $M \times N$, avanzando por casillas adyacentes en vertical u horizontal.
- Cada posición de la cuadrícula tiene un peso asociado al coste de transitar por dicha posición y puede haber posiciones que no sean accesibles para el agente (las representaremos con un 0), constituyendo obstáculos en la trayectoria.
- El objetivo de la búsqueda es viajar desde una posición de inicio (START) a una posición objetivo (GOAL) evitando obstáculos y *con el mínimo coste* posible.

Este problema tiene interés en diversos campos: en videojuegos comerciales (por ejemplo, Age of Empires o Los Sims) para controlar el movimiento de los agentes, en navegación autónoma de vehículos o en logística, para planificar rutas de transporte. Obviamente, la versión del problema aquí considerada es una simplificación y admite muchas variantes con mayor grado de sofisticación (y dificultad).



Ejemplo de problema y solución (los costes se representan con tonos de verde)

Ejemplo de aplicación a videojuegos.

Figura 1: Problemas de búsqueda de rutas en cuadrículas.

2.1. Función heurística

Un posible heurístico para este problema es la distancia Manhattan: si (X, Y) es la posición actual y (X_G, Y_G) es el objetivo, entonces:

$$h(X, Y) = |X_G - X| + |Y_G - Y|$$

donde $|\cdot|$ denota el valor absoluto. Se puede (y se debe) demostrar que este heurístico es admisible. ¿Es consistente? Razonad la respuesta.

3. Trabajo a realizar

Después de completar el código proporcionado, los alumnos han de lanzar una experimentación y elaborar una memoria según las indicaciones que se proporcionan a continuación.

Código a completar

En la práctica 2 hemos formulado el problema de búsqueda de caminos en cuadrículas o *grid pathfinding* y hemos completado el código de Java proporcionado para resolverlo utilizando los tres algoritmos de búsqueda no informada visto en clase.

Puesto que esta práctica es una continuación de la 2, hay que "copiar" de la misma el código :

1. El método `tratarRepetidos()`, que se implementó para la clase `BusquedaCosteUniforme` (que ahora queda obsoleta), para la clase `BusquedaPrimeroMejor` (que hereda de la clase abstracta `Busqueda` y sustituye a coste uniforme).
2. Clase `AccionGPF`.
3. Métodos de la clase `ProblemaGPF`.

Además, hay que añadir algo más de código en el paquete `busqueda.GPF`:

1. Completar la clase `HeuristicoGPFManhattan`, que calcula el valor del heurístico Manhattan.
2. Añadir una nueva clase `HeuristicoMalo` que calcule un heurístico (inventado por vosotros) que sea "malo" (no admisible y no dominante).

3. Completar la clase `Tester` para lanzar una experimentación exhaustiva que permita analizar el comportamiento de A* con los distintos heurísticos (tal y como se indica a continuación).

Pueden encontrarse más indicaciones en el código fuente.

Contenido de la memoria

La memoria ha de contener las siguientes secciones, con la respuesta a las preguntas que se plantean a continuación:

1. **Heurísticos considerados.** Describe los heurísticos considerados, a saber: el heurístico trivial (h_0), el heurístico Manhattan (h_M) y otro no admisible (h_{NA}) propuesto por los componentes del grupo. Junto con esta descripción, ha de aportarse una justificación de su consistencia o admisibilidad (o de la falta de las mismas).

2. **Ejemplo ilustrativo.** Desarrolla el árbol de búsqueda utilizando A* con el heurístico trivial h_0 y el heurístico Manhattan h_M para el ejemplo de tamaño 6×6 en la Figura 2. ¿Obtienes el mismo árbol con ambos heurísticos? Sin desarrollar ningún árbol más, ¿sabrías cómo se ha desarrollado la búsqueda de coste uniforme? ¿Por qué?

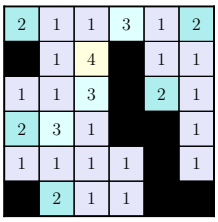


Figura 2: Ejemplo de cuadrícula 6×6 donde el nodo de inicio es el que está en la posición $(4, 5)$ y el nodo final en la posición $(4, 1)$ (suponiendo que la posición (0,0) es la de arriba a la izquierda) y las posiciones inaccesibles están en negro.

3. **Implementación.** De manera sucinta, aporta las aclaraciones o justificaciones que consideres relevantes sobre detalles de implementación.

4. **Resultados experimentales.** Proporciona y analiza resultados experimentales sobre un número razonable de instancias, de manera que puedas comparar los algoritmos y heurísticos propuestos en términos de optimalidad y coste de la búsqueda, relacionando los resultados con lo afirmado en teoría sobre cada tipo de búsqueda y sobre admisibilidad y dominancia de heurísticos. Más concretamente:

   - Completa el código fuente para que se generen aleatoriamente $N = 30$ problemas de tamaño 10×10 con costes entre 1 y 9 y de manera que la probabilidad de que una casilla sea un obstáculo sea 0,2. Estos problemas han de guardarse cada uno en un fichero `test<i>.txt` donde `<i>` es el número que identifica el problema (por ejemplo, `test1.txt` o `test20.txt`), de modo que los resultados que se presenten en la memoria sean reproducibles.
   - Modifica el tester para que con cada uno de los N problemas se lancen 3 búsquedas A* (con heurístico $h = 0$, heurístico Manhattan h_M y el heurístico no admisible propuesto h_{NA}) y se guarde la siguiente información sobre cada una de las búsquedas: número de nodos en frontera y número de nodos explorados (sumándolos se sabe el número total de nodos generados) y coste de la solución (si se ha encontrado).
   - Con los resultados, elabora una tabla con una fila por cada problema y tres grupos de columnas con la información recabada por cada búsqueda. ¿Puede haber algún problema en

el que A* con h_0 encuentre una solución con coste distinto que A* con h_M? ¿Y que A* con h_{NA}? ¿Por qué?

- A la vista de los resultados, ¿crees que tiene sentido hacer una media del número de nodos generados/expandidos en los N problemas para comparar los distintos algoritmos? Igualmente, ¿tiene sentido ver lo lejos que se ha quedado una solución no óptima de la óptima en términos de la diferencia absoluta de costes (por ejemplo, si la óptima tiene coste 100 y la no óptima tiene coste 113, decir que una es peor que la otra en 13 unidades)? Razona y justifica tus respuestas.
- Procesa los resultados de la tabla para obtener los siguientes datos:
  a) Porcentaje de las N ejecuciones en las que, de existir solución, cada una de las versiones del algoritmo ha alcanzado el óptimo (en realidad, ya antes de obtener resultados deberías saber qué número te saldrá para alguna de las versiones; ¿por qué?).
  b) Error relativo[1] (en porcentaje) del coste obtenido con el heurístico no admisible h_{NA} respecto a las otras dos versiones de A*. ¿Por qué hacemos sólo esta comparación y no comparamos A* usando h_0 con A* usando h_M? (la respuesta está relacionada con la de la pregunta anterior).
  c) Finalmente, error relativo en número de nodos generados (frontera + visitados) y número de nodos expandidos (visitados) de A* con h_0 respecto a A* con h_M.

  A la vista de estos datos, reflexiona brevemente sobre si se cumple empíricamente lo comentado en teoría sobre optimalidad y coste de A* en función de la admisibilidad y dominancia del heurístico usado.

Aclaraciones

A la hora de realizar esta práctica se observarán las siguientes normas de funcionamiento:

- La práctica se realizará en grupos de 4 ± 1 personas.
- Durante la realización de la misma es esperable que surjan dudas; se recomienda a los alumnos que se dirijan a los profesores para resolverlas y obtener ayuda y aclaraciones adicionales al contenido de este documento.
- Parte del trabajo puede y debe realizarse en las sesiones de prácticas, contando con la supervisión y ayuda de los profesores.
- Para la evaluación de la práctica no sólo se tendrá en cuenta que el programa "funcione"; también se valorará la claridad y calidad del código, la claridad y corrección de la memoria, la capacidad de síntesis de los alumnos, la corrección de los resultados experimentales y el análisis asociado, etc.

4. Referencias

Referencias básicas Todo el material necesario relacionado con búsqueda no informada y heurística se encuentra en las notas proporcionadas en teoría, así como en la bibliografía de la asignatura, tanto la básica [8] como la complementaria [3],[4],[7] (véase la guía de lectura). También se explica desde el punto de vista de su aplicación a los videojuegos en [5].

Referencias adicionales para los curiosos Una referencia sobre el uso de A* para el problema de planificación de rutas en videojuegos es [1], referencias a su uso para planificación de caminos de vehículos autónomos pueden encontrarse en [6],[2] y [9] y una propuesta para usar este problema en prácticas de Sistemas Inteligentes puede encontrarse en [10].

[1] Si un método obtiene coste C_1 y otro coste C_2, el error relativo de C_2 respecto a C_1 es $\frac{C_2-C_1}{C_1}$

Referencias en la red Una breve búsqueda en internet aportará abundante (quizás excesivo) material relacionado con el problema de búsqueda de rutas en cuadrículas y su resolución con el algoritmo A* o variantes. Dos buenos puntos de partida son:

- A* Pathfinding for Beginners: `http://www.policyalmanac.org/games/aStarTutorial.htm`
- Amit's A* Pages: `http://theory.stanford.edu/~amitp/GameProgramming/`

Referencias

[1] X. Cui and H. Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security* 11(1): 125–130. 2011.

[2] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel. Practical search techniques in path planning for autonomous driving. En *Search in Artificial Intelligence and Robotics. Papers from the AAAI Workshop* WS-08-10: 32–37. AAAI Press, 2008.

[3] N. J. Nilsson. *Artificial Intelligence: a new synthesis.* Morgan Kaufmann, 1998.

[4] J. T. Palma y R. Marín. *Inteligencia artificial: métodos, técnicas y aplicaciones.* McGraw-Hill, 2008.

[5] I. Millington y J. D. Funge. *Artificial Intelligence for Games.* Taylor & Francis, 2nd. ed. 2009.

[6] M. Montemerlo *et al.* Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics* 25(9): 567–597. 2008.

[7] D. L. Poole and A. K. Mackworth. *Artificial Intelligence. Foundations of Computational Agents.* Cambridge University Press, 2010. `http://artint.info/index.html`

[8] S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach.* Prentice-Hall Int., 3rd ed, 2010. `http://aima.cs.berkeley.edu/`

[9] S. Thrun. Toward robotic cars. *Communications of the ACM* 53(4): 99–106. 2010.

[10] M. Zyda and S. Koening. Teaching Artificial Intelligence playfully. En *Papers from the 2008 AAAI Workshop AI Education Colloquium* volume WS- 08-02, 90–96. AAAI Press, 2008.