Universidad Internacional de la Rioja (UNIR)

Máster en Inteligencia Artificial

Razonamiento y Planificación Automática

Actividad grupal: búsqueda de rutas en empresa de paquetería

Fecha: 23/12/2024

Autores: Sergio Pérez Rivero, Asier Marín Fernández, David García Úbeda

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Índice

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

1. Estudio inicial del problema

En la actividad propuesta se simula un problema en el que se plantea un entorno representado como una matriz rectangular (imagen) donde una furgoneta debe desplazarse desde un punto inicial (3,3) hasta un punto final (5,1). Existen restricciones como movimientos limitados a verticales y horizontales, el paso de una casilla a otra que esté justo a su lado y presencia de obstáculos no transitables.

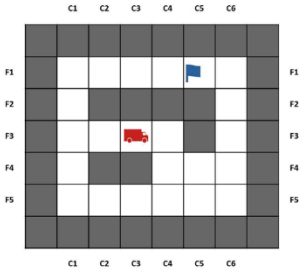


Ilustración 1: Puzle

El objetivo es aplicar diversos algoritmos de búsqueda informadas y no informadas para generar automáticamente la ruta, dando como resultado el coste total del recorrido, la longitud, los nodos expandidos por el algoritmo y evaluando la eficacia y lo óptimo que es cada uno.

2. Estudio Caso 1

En este caso se llevará el estudio de los algoritmos de búsqueda en amplitud y búsqueda en profundidad, donde el coste de avanzar hacia cada dirección es 1 en todas las direcciones.

2.1. Tabla comparativa algoritmo BFS y DFS

Algoritmo	Solución	Longitud	Coste de la solución	Nodos expandidos	Tamaño máximo de la lista
Algoritmo de búsqueda en amplitud (BFS)	######## # P·# # ####·# # T·#·# # ##···# # # ########	9	8	23	5
Algoritmo de búsqueda en profundidad (DFS)	######## # P·# # ####·# #··T #·# #·## ·# #·····# ########	15	14	15	5

Tabla 1: Comparativa algoritmo BFS y DFS

2.2. ¿Obtiene amplitud un resultado óptimo? ¿Por qué?

Como vemos, en este caso, el algoritmo de búsqueda en amplitud nos da mucho mejor resultado que el algoritmo de búsqueda en profundidad en términos de longitud y coste.

Esto es algo que podíamos prever que iba a ocurrir, debido a que, en términos de longitud, el algoritmo de búsqueda en amplitud siempre nos dará un óptimo, ya que encontrará la solución

en la longitud mínima. Por otro lado, en cuanto al coste de la solución, en este tipo de caso donde cada movimiento vale lo mismo independientemente de la solución (como aquí donde cada movimiento vale 1 sin importar hacia donde nos dirijamos), este algoritmo también nos dará una solución que nos proporcione el coste óptimo.

Esto es así porque evidentemente, si todos los movimientos tienen el mismo coste, optimizar el número de movimientos como este algoritmo hace, nos optimizará el coste indirectamente. En cambio, computacionalmente, este algoritmo ha sido menos eficiente por haber tenido que explorar más nodos que el algoritmo en búsqueda en profundidad. Por lo tanto y como conclusión, remarcar como en este tipo de problemas donde los costes en movimiento son uniformes, si podemos asegurar que el algoritmo en amplitud nos da la solución óptima en términos de coste y movimientos. Además, como el espacio en este problema es finito y no demasiado extenso, no supone especial problema el hecho de expandir más nodos.

2.3. ¿Obtiene profundidad un resultado óptimo? ¿Por qué?

Por otro lado, el algoritmo de búsqueda en profundidad no garantiza encontrar una solución óptima en términos de longitud ni de coste. Esto es así por las características propias de este algoritmo, el cual explora cada rama del árbol de búsqueda de manera exhaustiva antes de retroceder para explorar otras alternativas. Esto es algo que podíamos prever debido a que este algoritmo prioriza la exploración en profundidad sin considerar si el nodo actual pertenece a la solución más corta o económica. Por esta razón, el algoritmo en profundidad no está diseñado para optimizar ni la longitud del camino ni su coste.

En términos de longitud, como se ha explicado en el anterior párrafo, el algoritmo de búsqueda en profundidad podría encontrar una solución que no sea mínima porque sigue explorando hasta que alcanza un nodo meta, independientemente de la profundidad en la que se encuentre dicha solución. Esto significa que podría ignorar soluciones con, por ejemplo, menor coste, simplemente porque se encuentra profundizando en otra rama. En cuanto al coste de la solución, ocurre lo mismo que con la longitud, ya que, aunque todos los movimientos tengan el mismo coste, este algoritmo no nos asegura que el camino encontrado sea el de menor coste, ya que como ocurría con la longitud, va a seguir la rama en la que se encuentre sin tener en cuenta otros factores como el coste para cambiar de rama. Computacionalmente, sí que nos da mejores resultados, ya que nos da una solución expandiendo menos nodos en este caso, pero como he explicado en la respuesta anterior, no es algo primordial en estos problemas con un espacio finito y no muy amplio. Por lo tanto, concluimos que el algoritmo de búsqueda en profundidad no proporciona una solución óptima en términos de coste ni de longitud, incluso en problemas donde los costes son uniformes, ya que su enfoque de exploración en profundidad no prioriza la búsqueda de caminos mínimos, y tampoco realiza una comparación exhaustiva de las posibles soluciones antes de finalizar.

2.4 ¿Cuál de los dos algoritmos es más eficiente en este caso?

Como se ha explicado en las anteriores respuestas, en términos de coste y longitud, es mucho más eficiente el algoritmo de búsqueda en amplitud, ya que en este caso particular nos da la

solución óptima, en detrimento del algoritmo de búsqueda en profundidad, el cual no nos asegura que sea una solución óptima.

3. Estudio caso 2

En este caso se llevará el estudio de los algoritmos de búsqueda en amplitud, el algoritmo Dijkstra y el algoritmo A*, donde el coste de avanzar hacia cada dirección viene definido por la siguiente estructura: COSTS = {"up": 5.0, "down": 5.0, "right": 2.0, "left": 2.0}

3.1. Tabla comparativa algoritmo BFS, UCS y A* con costes diferentes

Algoritmo	Solución	Longitud	Coste	Nodos expandidos	Tamaño máximo de la lista
Búsqueda en amplitud (BFS)	######## # P·# # ####·# # T·#·# # ##···# # # ########	9	28	23	5
Coste uniforme (UCS)	######## #····P # #·#### # #··T # # # ## # # # ########	9	22	22	6
A* (H1)	######## #····P # #·#### # #··T # # # ## # # # ########	9	22	20	6

*Tabla 2: Comparativa algoritmo BFS, UCS y A**

3.2. ¿Obtiene UCS (Dijkstra) el camino de coste óptimo?

Sí, para el mapa seleccionado con la estructura de coste indicada, el algoritmo UCS (Dijkstra) obtiene el camino de coste óptimo. Su funcionamiento se basa en evaluar los nodos por el coste acumulativo g(n) expandiendo primero los nodos con el menor coste posible. Este funcionamiento no utiliza heurísticas y evalúa las posibles opciones antes de cerrar un nodo. Comparando el resultado con los otros dos algoritmos del caso (BFS y A*), el resultado se puede comparar con el algoritmo A* ya que ambos dan como resultado el óptimo en función de coste y longitud.

3.3. ¿Obtiene A* el camino de coste óptimo?

Sí, para el mapa seleccionado (mapa) con el coste indicado en este caso (coste), el algoritmo A* obtiene el camino de coste óptimo. Gracias a la heurística que se está utilizando, que es la basada en Manhattan, se permite que A* encuentre el camino más corto posible haciendo que en este caso también sea el camino de costo óptimo. Dado el principio de heurística, A* se basa

principalmente en la longitud y en el coste. Comparando este resultado con los otros dos algoritmos del caso (BFS y A*), el resultado obtenido es óptimo y eficiente.

3.4. ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo, aunque se varíe el mapa?

Si el mapa varía, puede que los resultados obtenidos cambien y las respuestas no sean las mismas para las preguntas anteriores. En el caso del algoritmo UCS, siempre va a encontrar el camino de coste óptimo independientemente de las características del mapa, siempre que el mapa esté bien definido y mantenga una forma estable. Esto se debe a que la elección de cada nodo para construir la ruta no depende de heurísticas, solo depende del coste acumulado total, donde se garantiza que todos los nodos se cierren una vez se garantice su camino con menor coste. Para el caso del algoritmo A*, dependerá de la heurística que se utilice. Si el mapa cambia y se utiliza una heurística incorrecta, el algoritmo podría elegir caminos que no son eficientes como sería en el caso de mapas con caminos complejos.

3.5. ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo, aunque se varíen los costes?

Tampoco se puede garantizar que las respuestas 2 y 3 siempre tendrán la misma respuesta, aunque se varíe el coste, ya que cada algoritmo funciona de una forma u otra. El algoritmo UCS se centra únicamente en el coste acumulado con lo que por mucho que varíen los costes, se seguiría garantizando que encuentra el camino de coste óptimo. Siempre existen casos excepcionales lo que puede producir que el algoritmo no llegue a la solución, aunque para que no suceda el mapa tiene que estar bien planteado, los costes sean positivos y las decisiones que tome el algoritmo para casos donde el coste acumulado sea el mismo sean correctas (problema que no sucedería con A*). En el caso del algoritmo A*, el desempeño dependerá de la heurística que se utilice. Si esta tiene en cuenta el coste acumulado por una parte y la heurística por otra, el algoritmo podría decidir a qué nodo expandirse para llegar antes al objetivo y tener en cuenta el coste total del recorrido. También se pueden poner comprobaciones para que los nodos cerrados tengan una revisión a posterior para comprobar posibles recorridos que no se han tenido en cuenta y compararlos con el escogido en un principio.

3.6. ¿Cuál de los dos algoritmos (UCS o A*) es más eficiente en este caso?

El algoritmo A* es más eficiente que el UCS, frente a que ambos tienen el mismo resultado en coste y longitud, A* consigue expandir 20 nodos a diferencia de 22 que expande UCS. Esto se debe a la heurística que utiliza el algoritmo para hacer la búsqueda directamente hacia el nodo objetivo, reduciendo la cantidad de nodos que son innecesarios explorar. Un ejemplo claro que diferencia ambos algoritmos sería el caso en el que si el objetivo está hacía arriba pero el nodo inferior tiene menos coste, el algoritmo UCS se expandiría hacía el nodo inferior, en cambio el A* como utiliza la heurística que guía el recorrido hacía el objetivo, podría elegir el camino hacia arriba reduciendo el número de nodos expandidos y posiblemente el camino con coste óptimo.

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

4. Estudio Caso 3

En este caso se compara el algoritmo de búsqueda A* con diferentes heurísticas. Estas heurísticas se pueden describir de la siguiente manera. Teniendo en cuenta que (x, y) son los puntos cardinales equivalentes al eje de abscisas y al eje de ordenadas de la posición o estado del nodo. Y teniendo en cuenta también que (bx, by) son los puntos cardinales equivalentes al eje de abscisas y al eje de ordenadas de la posición objetivo:

1. La **heurística 1** tiene como definición:

$$h(n) = |x - gx| + |y - gy|$$

Esta definición es típicamente conocida como la distancia de Manhattan. Esta heurística trata de encontrar la distancia diagonal restringiendo un eje a la vez. Es especialmente importante destacar que la heurística en ningún caso subestima el coste real asegurando que el algoritmo encuentre la solución óptima y por lo tanto sea admisible.

2. La **heurística 2** tiene como definición:

$$h(x) = \max(|x - gx|, |y - gy|)$$

La heurística 2 a diferencia de la 1 selecciona el número de mayor valor de la distancia entre el eje de abscisas y el eje de ordenadas. Se conoce como la heurística de Chebyshev y calcula la distancia más corta entre dos puntos haciendo una estimación en base a cualquier dirección (incluyendo diagonal).

3. La **heurística 3** tiene como definición:

$$h(n) = 2 \cdot (|x - gx| + |y - gy|)$$

La heurística 3 duplica el valor de la distancia de Manhattan. Esto significa que hace una sobreestimación heurística de la distancia entre los puntos. Esta multiplicación puede tener diferentes implicaciones ya que puede alterar considerablemente la admisibilidad heurística subestimando el coste real para alcanzar el objetivo. Si se efectúa una comparativa del cálculo de la heurística en el primer movimiento P(3, 3) al P(2, 3) (punto rojo) del puzle con las diferentes heurísticas, se obtiene la siguiente tabla:

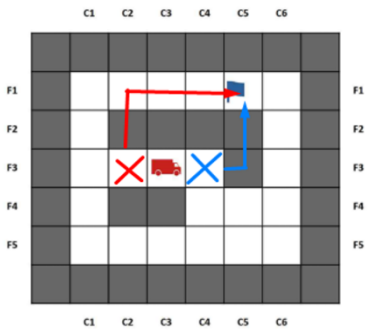


Ilustración 2: Puzle modificado

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Heurística	Punto	Resultado
Manhattan	P(2, 3)	$h(n) = (\|x - gx\| + \|y - gy\|) = (3 + 2) = 5$
Chebyshev	P(2, 3)	$h(n) = \max(\|x - gx\|, \|y - gy\|) = \max(3,2) = 3$
Manhattan (2*)	P(2, 3)	$h(n) = 2 \cdot (\|x - gx\| + \|y - gy\|) = 2 \cdot (3 + 2) = 10$
Manhattan	P(4, 3)	$h(n) = (\|x - gx\| + \|y - gy\|) = (1 + 2) = 3$
Chebyshev	P(4, 3)	$h(n) = \max(\|x - gx\|, \|y - gy\|) = \max(1,2) = 2$
Manhattan (2*)	P(4, 3)	$h(n) = 2 \cdot (\|x - gx\| + \|y - gy\|) = 2 \cdot (1 + 2) = 6$

Tabla 3: Ejemplo heurísticas

Para que la heurística sea admisible se debe cumplir $f(n) = g(n) + h(n)$ donde $h(n)$ no puede superar el coste real. En el caso de la heurística 3 con valor 10 es mayor que el coste real siendo 8.

Otra reflexión relevante es que la heurística no está teniendo en cuenta las paredes que se encuentran dentro del contexto. La heurística está haciendo un cálculo de la distancia entre dos puntos cardinales, pero no tiene conocimiento de las paredes que imposibilita el movimiento dentro del contexto. A consecuencia de esto, todas las heurísticas le están indicando al algoritmo que el camino más corto es hacía la derecha P(4, 3) y no la izquierda P(2, 3). A efectos prácticos en ambos puntos están a la misma distancia de la meta, siendo 7 movimientos, pero la heurística está llevando al algoritmo por el camino que cree que es más corto por el simple cálculo de distancia entre puntos. El algoritmo se moverá por la derecha hasta que el coste estimado sea menor por el camino de la izquierda a consecuencia de las paredes o nodos que no se pueden pasar.

4.1. Tabla comparativa de ejecuciones

El coste de avanzar hacia cada dirección es 1 en todas las direcciones:

COSTS = {"up": 1.0, "down": 1.0, "right": 1.0, "left": 1.0}

Algoritmo	Solución	Longitud	Coste	Nodos expandidos	Tamaño máximo de la lista
A* (H1)		9	8	16	5
A* (H2)		9	8	17	6

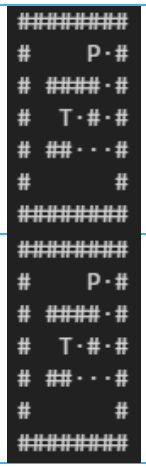
A* (H3)		9	8	11	4
UCS		9	8	23	5

Tabla 4: Comparación resultados caso 3

4.2. ¿Obtiene UCS (Dijkstra) el camino de coste óptimo?

UCS obtiene el camino con el coste óptimo en condiciones en las que todos los movimientos tienen el mismo coste. Realmente recorre al mismo tiempo ambos caminos (por la derecha y por la izquierda). Debido a esto se expanden más nodos para encontrar la solución correcta, ya que, va expandiendo según el coste acumulado. Si la posición de partida hubiera sido diferente probablemente también hubiera encontrado el camino de coste óptimo a coste de expandir más nodos que un algoritmo informado.

4.3. ¿Obtiene A* el camino de coste óptimo con todas las heurísticas?

El algoritmo A* encuentra el camino con el coste óptimo en todas las heurísticas, sin embargo, la heurística $h(n)$ no le proporciona información totalmente útil para recorrer el menor número de nodos posible. La heurística indica que el camino de la derecha es más corto debido a la distancia entre los dos puntos. El algoritmo no es consciente de las paredes que existe en el camino.

Las paredes dentro del puzle afectan significativamente al cálculo del coste y subestima el coste real. Como consecuencia, explora más nodos de lo necesario aumentando el tiempo de búsqueda. La complejidad espacial y temporal se ve afectada a causa de esto.

Para resolver este problema se podrían adaptar medidas sobre la heurística 1 y 2 para minimizar el número de nodos visitados. Se podría especificar a la heurística cuales son los nodos bloqueados o que éstos tengan un coste mayor. Por ejemplo, cuando la heurística calcula la distancia entre los puntos, si uno de los nodos es una pared, el costo debe ser mayor, de esa manera "se fuerza" a buscar el camino más corto considerando las paredes.

Por otro lado, la heurística 3 resuelve el problema de visitar los nodos adicionales proporcionándole una importancia mayor al multiplicar por 2 su valor. Esta sobre estimación se fuerza a que vaya "más recto" y visite menos nodos a costa de una solución subóptima más rápida. El factor $h(n)$ hace que expanda menos nodos con una actitud más "voraz". Aun así, hay que tener en cuenta los puntos negativos de sobreestimar el valor heurístico, ya que no garantiza encontrar la solución óptima a diferencia de la heurística 1 y 2.

Algoritmo	Ventajas	Desventajas
A* (H1)	- Obtiene la solución óptima. - Es admisible. - Es consistente.	- Expande más nodos de lo necesario debido a las paredes intermedias.
A* (H2)	- Obtiene la solución óptima. - Es admisible. - Es consistente.	- Expande más nodos de lo necesario debido a las paredes intermedias.
A* (H3)	- Más rápida, puede expande menos nodos.	- No es admisible heurísticamente hablando, el coste estimado supera al coste real. - No garantiza la solución óptima. - Puede convertirse menos eficiente dependiendo del contexto.
UCS	- Encuentra una solución de coste óptima.	- No tiene información del contexto y expande más nodos que los demás.

Tabla 5: Ventajas y desventajas algoritmos caso 3

4.4. ¿Se puede afirmar que la respuesta a 4 no varía al cambiar la especificación del mapa para ninguna de las heurísticas presentadas? ¿Puede probar esta afirmación diseñando un mapa que compruebe este hecho?

Se ha modificado el punto inicial para obtener un resultado diferente y comprobar que el algoritmo A* con las 3 heurísticas obtiene el coste óptimo. El punto elegido es P(2, 5), desde este punto se intentará comprobar que A* consigue el camino óptimo con todas las heurísticas.

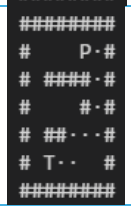
Algoritmo	Solución	Longitud	Coste	Nodos expandidos	Tamaño máximo de la lista
A* (H1)		10	9	21	4
A* (H2)		10	9	21	6
A* (H3)		10	9	13	4

Tabla 6: Comparación de resultados para las heurísticas utilizadas en el caso 3

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Los resultados son similares respecto a la tabla comparativa anterior. Se puede afirmar que A* encuentra el camino óptimo con todas las heurísticas. De nuevo, la heurística 3 es la más eficiente expandiendo tan sólo 13 nodos.

4.5. ¿Es el algoritmo A* igualmente eficiente en todos los casos? Intente explicar la razón de las diferencias observadas.

El algoritmo A* no es igualmente eficiente en todos los casos. Aun que en los 3 casos encuentra el camino óptimo, en la heurística 3 expande un número de nodos menor. En base a los resultados obtenidos, la heurística 3 es la más eficiente seguido de la heurística 1 que es algo más eficiente que la 2. Como se ha podido describir anteriormente, la heurística 3 tiene una actitud más “voraz” respecto al factor $h(n)$ y hace que vaya más "recto” en este contexto. Es especialmente importante recalcar que esto ocurre en este contexto, pero en contextos con más complejidad espacial no garantiza la solución óptima y puede convertirse en menos eficiente.

Anexos

Algoritmo búsqueda en profundidad (DFS)

Objetivo:

El objetivo principal del algoritmo de búsqueda en profundidad es explorar un grafo desde un nodo inicial hasta alcanzar un nodo objetivo. Este algoritmo se basa en expandir las ramas hasta que no se pueda expandir más o encontremos una solución, momento en el cual cambiamos de rama a expandir

Funcionamiento:

Se elige un nodo y se va expandiendo hacia niveles inferiores hasta que se encuentre una solución o no se pueda seguir expandiendo. Si no podemos seguir expandiendo, avanzaremos al último nodo no explorado.

Esto se repetirá hasta encontrar una solución, o en el caso de no poder seguir expandiendo ningún nodo, determinemos que no existe solución posible para el problema.

Problemas y limitaciones:

En primer lugar, este algoritmo no busca una solución óptima, si no que presenta la primera solución que presenta, sin tener que ser óptima en coste o profundidad.

Además, en espacios infinitos o muy profundos puede generar problemas de memoria al crecer demasiado.

Soluciones a los posibles problemas:

Una solución a los problemas de memoria es la utilización de límites en profundidad, lo cual hará que no se pueda pasar de X profundidad definida en búsqueda de la solución. Esto se conoce como algoritmo de búsqueda en profundidad limitado en profundidad.

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

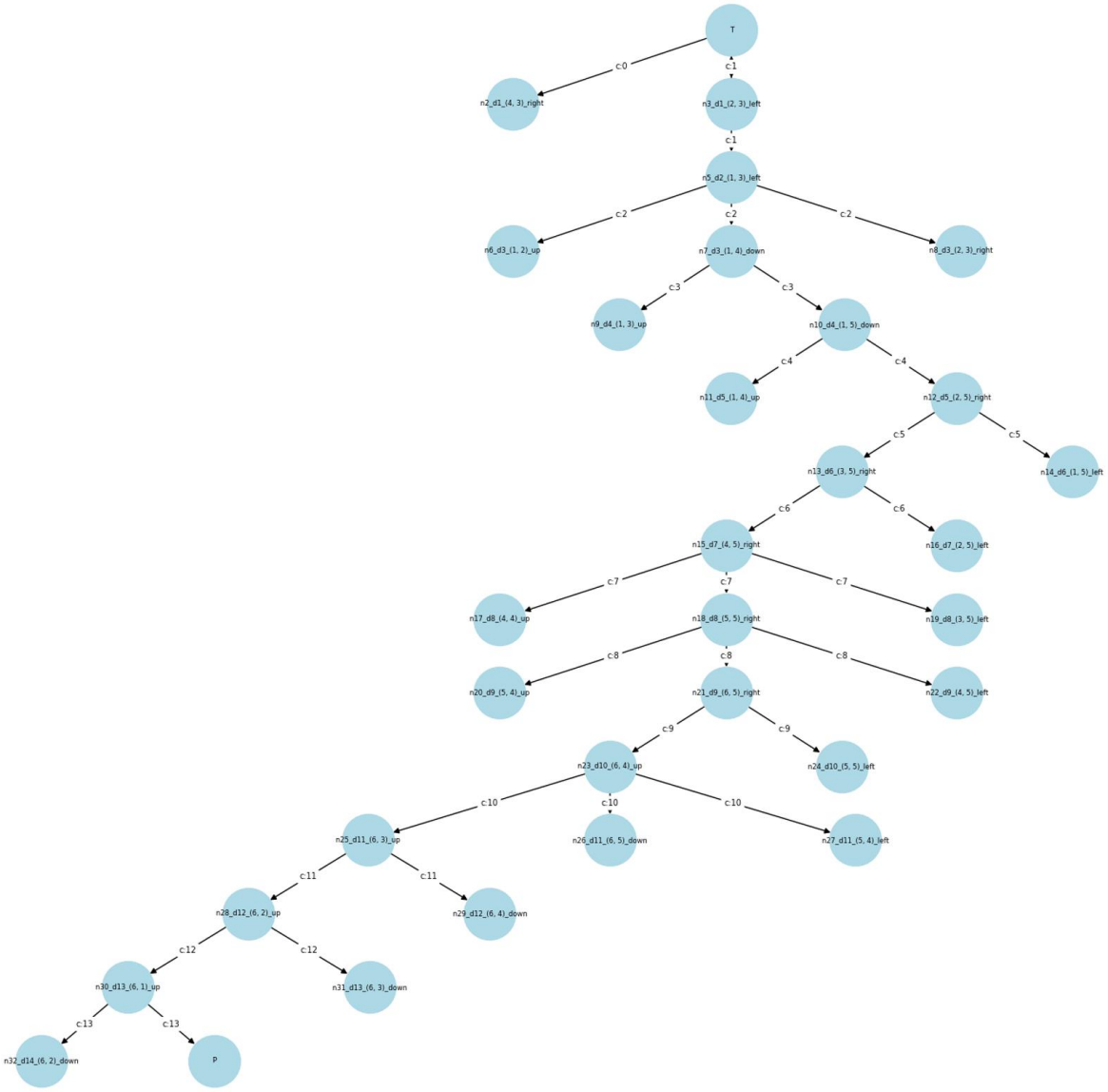


Ilustración 3: Algoritmo búsqueda en profundidad

Algoritmo búsqueda en amplitud (BFS)

Objetivo:

El objetivo del algoritmo de búsqueda en amplitud es explorar un grafo comenzando desde un nodo raíz, a partir del cual se irán expandiendo los nodos por niveles, finalizando siempre un nivel antes de pasar al siguiente nivel de profundidad. Este algoritmo asegura encontrar una solución en el menor número de pasos posible.

Funcionamiento:

Como se ha explicado, en este algoritmo, partiendo del nodo inicial, se van expandiendo todos los nodos posibles, de manera que, una vez todos los de un nivel de profundidad X se hayan completado, se podrá pasar a expandir los nodos de la profundidad X+1. De esta manera, nos aseguramos encontrar la solución en menor número de pasos.

Este algoritmo puede finalizar o buen por encontrar una solución, o porque no queden nodos que expandir, y por lo tanto no se haya encontrado una solución.

Además, este algoritmo, siempre que se encuentre en un problema con costes uniformes, asegura un coste óptimo también, ya que minimizaremos el número de pasos.

Problemas y limitaciones:

En primer lugar, encontramos con su alto coste computacional, ya que, en problemas con un gran número de nodos, pueden expandirse muchísimo. Además, si estas ramas son infinitas no podrá encontrar una solución.

Por otro lado, encontramos la limitación de que, si no hay costes uniformes, no nos asegura una solución coste óptimo.

Soluciones a los posibles problemas:

Para problemas con costes no uniformes, será recomendable usar una alternativa a este algoritmo, como pueda ser Dijkstra o A*.

Para reducir su coste computacional y evitar entrar en problemas infinitos, podemos limitar en profundidad el algoritmo. Otra opción para reducir el coste computacional es distribuirlo en distintas máquinas para su exploración en paralelo.

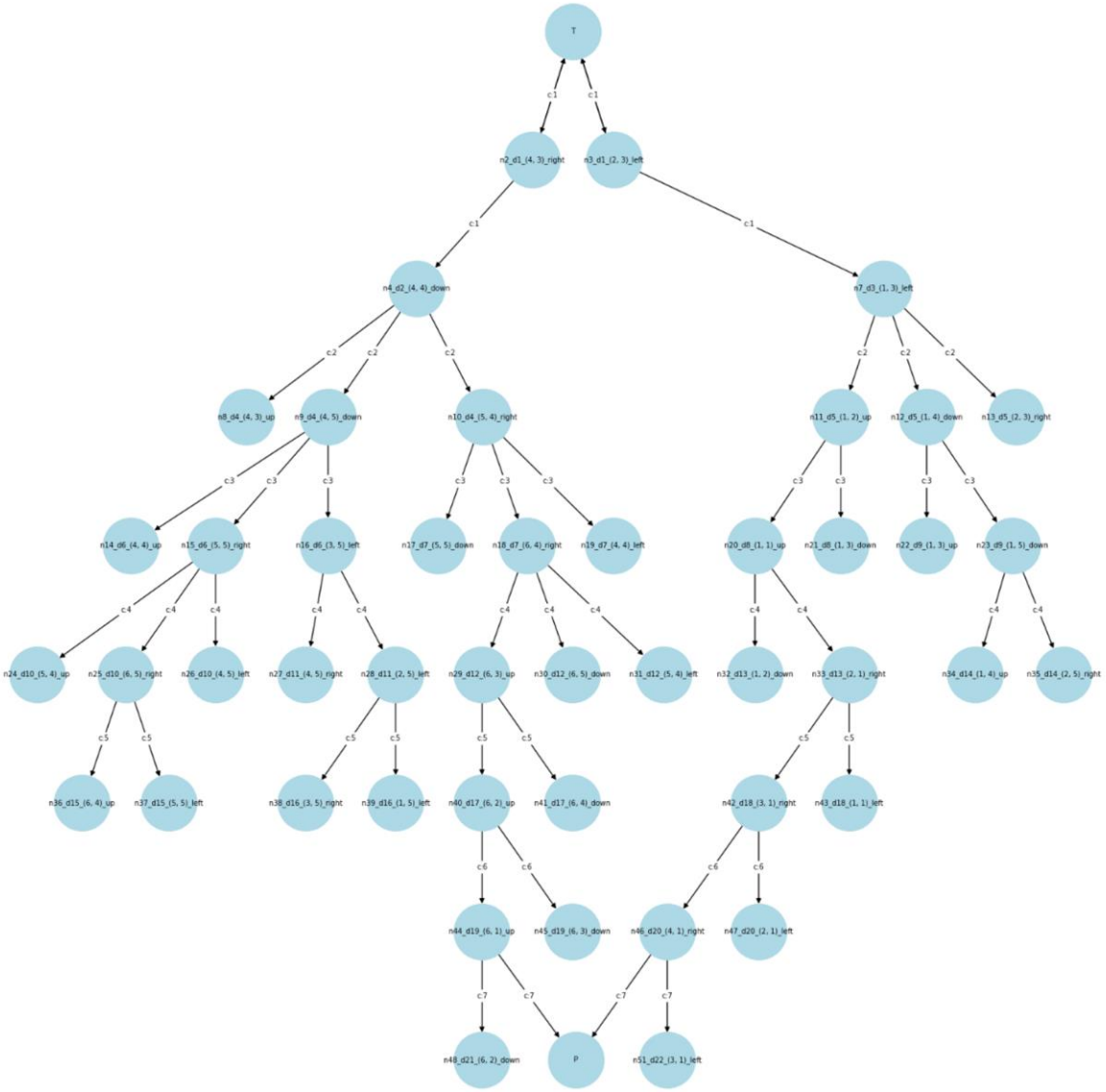


Ilustración 4: Algoritmo búsqueda en amplitud

Algoritmo de coste uniforme (Dijkstra o UCS)

Objetivo:

Se utiliza para encontrar el camino con menor coste desde un nodo inicial hasta el nodo objetivo.

Funcionamiento:

Su funcionamiento se basa en empezar desde el nodo inicial (nodo S) donde se expande a sus nodos vecinos (al ser el inicio no habrá ningún nodo vecino cerrado), y se calcula el coste que tiene llegar a cada uno de estos nodos.

Se escoge el camino con menor coste y este nodo escogido lo marcamos como cerrado (vamos a llamar a este nodo, nodo A). En este instante tendremos cerrados el nodo inicial (S) y este (A).

Posteriormente se tiene en cuenta la ruta con menor coste, considerando los nodos expandidos desde el nodo inicial y los nodos expandidos desde el nodo cerrado (A), donde se escogerá la ruta al siguiente nodo que no esté cerrado con menor coste posible, y este nodo escogido también lo cerraremos (nodo llamado B).

Repetimos el mismo caso para los nodos a los que nos podemos expandir desde los nodos cerrados (S, A, B). Y escogemos el nodo al que expandirnos que tenga el menor coste.

Se repite el caso hasta llegar a la solución o tener todos los nodos cerrados.

Problemas y limitaciones:

- **Nodos expandidos con mismo coste:**

Si hay que elegir entre dos o más nodos con el mismo coste acumulado, el algoritmo elige por la estructura que tiene, el primero en la lista. Esto puede dar problemas como por ejemplo el siguiente caso:

(A -> B) coste 5, (A -> C) coste 5, (B -> D) coste 6, (C -> D) coste 1

Según la estructura, el nodo A se puede expandir al nodo B y al C ambos con coste 5 pero al tener el mismo coste se eligiría por orden, eligiendo el nodo B. Posteriormente se cierra el nodo escogido (B) y llega al nodo final (D) con coste 11 (5+6). Si hubiese escogido el nodo C en vez del nodo B, hubiera llegado con coste final de 6 (5+1).

- **Nodos abiertos:**

Una vez se escoge el nodo con menor coste, este se cierra dejando la ruta marcada para llegar hasta este punto. El algoritmo funciona eligiendo el nodo al que expandirse en función del coste, y puede ser que se dé el caso siguiente:

(A -> B) coste 1, (A -> X) coste 10

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Para llegar al nodo final la única opción posible es por el camino que pasa por A y directamente a X. Como llegar a B tiene menor coste, elige ese camino y deja el nodo X abierto, con lo que no se llega a la solución por este nodo y se buscaría la solución por otro.

Soluciones a los posibles problemas:

Se pueden implementar heurísticas de estimaciones para guiar el proceso hacia los nodos más cercanos al objetivo, aunque estos no tengan el menor coste acumulativo, como pasaría con el algoritmo A*.

Mantener una lista de nodos cerrados para poder revisarlos periódicamente, de esta forma nos aseguramos de que todos los caminos son considerados.

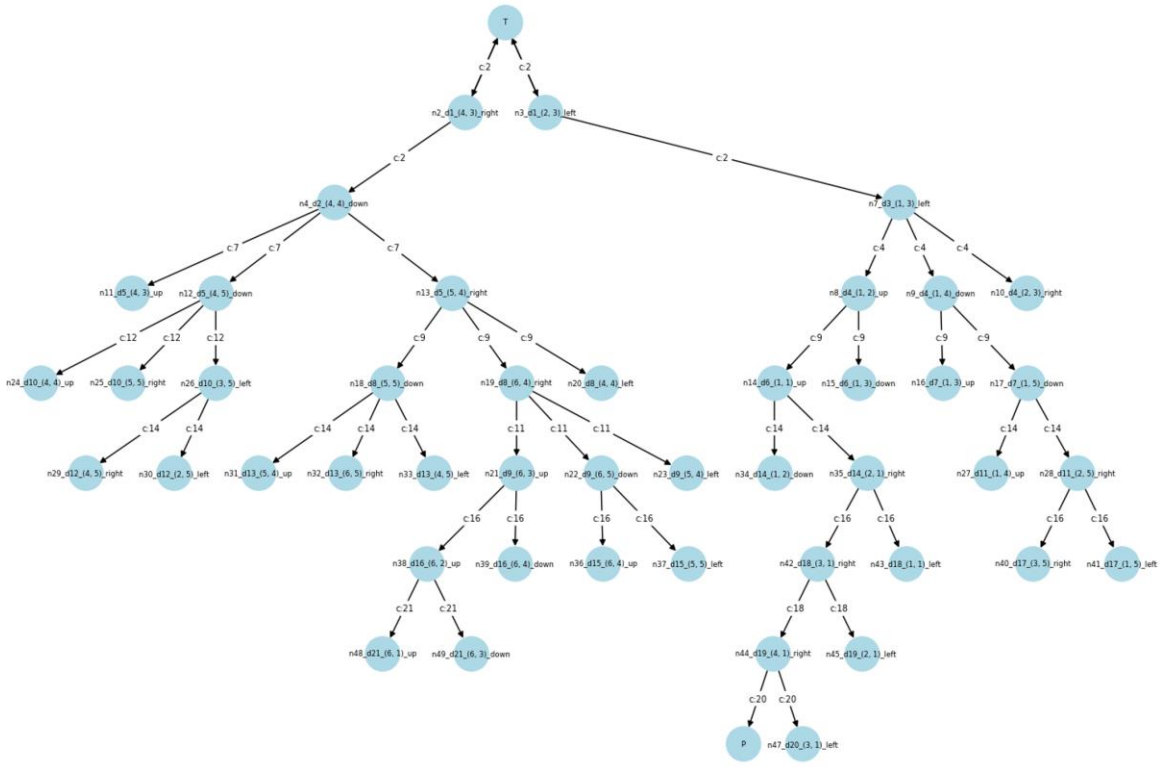


Ilustración 5: Algoritmo de coste uniforme

Algoritmo A*

Objetivo:

Su objetivo es encontrar el camino óptimo para llegar desde un nodo inicial hasta un nodo final, minimizando el coste acumulado y una heurística que vaya estimando el coste restante hasta llegar al destino.

Funcionamiento:

El funcionamiento que utiliza este tipo de algoritmo se basa en la siguiente fórmula:

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

Donde f(n) es el costo total, g(n) es el coste real acumulado desde el nodo inicial hasta el nodo actual y h(n) es la heurística que estima el coste restante al nodo objetivo.

Primero se empieza desde el nodo inicial (S) y se expande hacia los nodos a los que puede llegar, calculando el coste acumulado hasta cada nodo vecino, donde se evalúa el valor de f(n) y nos quedamos con el de menor valor cerrándolo.

Este proceso se repite hasta alcanzar el nodo objetivo o hasta que todos los posibles nodos se hayan visitado.

Para calcular h(n), existen diversos métodos que se determinan según el tipo de heurística con la que se esté trabajando, y unas de las más comunes son:

- **Heurística Manhattan**

Se utiliza normalmente para cuando los movimientos son ortogonales (arriba, abajo, izquierda, derecha). La heurística h(n) presenta la siguiente fórmula:

$$h(n) = |x - gx| + |y - gy|$$

Donde “x” es la coordenada actual en el eje horizontal en el nodo en evaluación, “gx” es la coordenada en el eje horizontal del nodo objetivo, “y” es la coordenada actual en el eje vertical en el nodo en evaluación y “gy” es la coordenada en el eje vertical del nodo objetivo.

- **Heurística Chebyshev**

Se utiliza normalmente cuando los movimientos además de ser ortogonales permiten ser diagonales. La heurística h(n) presenta la siguiente fórmula:

$$h(x) = \max(|x - gx|, |y - gy|)$$

Donde “x” es la coordenada actual en el eje horizontal en el nodo en evaluación, “gx” es la coordenada en el eje horizontal del nodo objetivo, “y” es la coordenada actual en el eje vertical en el nodo en evaluación y “gy” es la coordenada en el eje vertical del nodo objetivo.

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Problemas y limitaciones:

El algoritmo puede expandir muchos nodos que afecten al tiempo de ejecución y uso de memoria.

Cuando varios nodos tienen el mismo coste total calculado, dependerá del orden en el que estén los nodos almacenados lo que a veces lleva a decisiones que no son óptimas.

Si existen obstáculos inesperados, el algoritmo puede explorar nodos sin salida y no alcanzar el objetivo final.

Soluciones a los posibles problemas:

Es mucho mejor utilizar heurísticas que subestimen el coste real para encontrar soluciones óptimas. De esta forma, el algoritmo considerará todos los caminos posibles sin desviar el enfoque a rutas incorrectas.

Reducir la exploración innecesaria para llegar al objetivo estableciendo límites, como por ejemplo enfocar la búsqueda en las áreas más relevantes del espacio.

Establecer reglas para desempatar entre dos nodos con la misma f(n) y priorizar aquellos que estén más cerca del objetivo definido según la heurística.

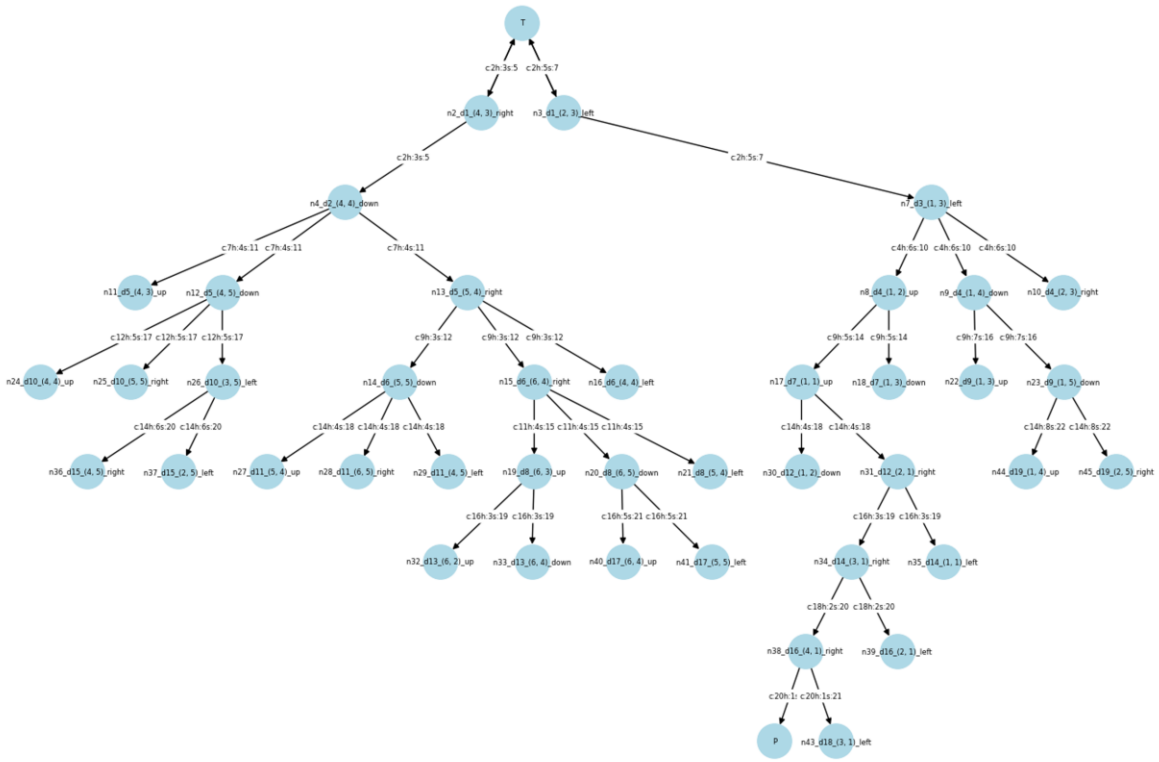


*Ilustración 6: Algoritmo A**

Sergio Pérez Rivero
Asier Marín Fernández
David García Úbeda

Código fuente utilizado

Se ha modificado el código para poder mostrar de manera individual cada algoritmo. También se han utilizado herramientas externas para mostrar árboles de nodos. La mayoría de los cambios se han realizado en el método actions() para obtener las cadenas de relaciones entre nodos.

Código en GitHub: https://github.com/UNIR-RPA/UNIR-RPA-ACTIVIDAD-1/blob/asier/miunar04_act1/Cuaderno_Actividad_1_B%C3%BAsqueda.ipynb

Herramientas de código completarías

- Librería networkx para mostrar árboles de nodos.
- Instalación de Graphiz para mostrar nodos.