Sistemas Inteligentes

Práctica 1: Búsqueda en el espacio de estados

Pablo Gomariz Martínez

Daniel Tomás Martínez

Índice

Índice de Ilustraciones

Índice de tablas

1. Introducción al problema

El problema planteado consiste encontrar un camino mediante el cual un coche o varios lleguen al final de un laberinto, dada una pista con obstáculos en la que tenemos una línea de inicio, una línea de meta y un cierto número de coches.

Explicado con más detalle, tendremos varios coches situados al azar en casillas de la línea de inicio. Estos coches deberán moverse por una pista cuadrada sorteando los obstáculos para llegar a la línea de meta. Cuando todos los coches se encuentren en la línea de meta se finaliza satisfactoriamente.
El diseño la pista puede ser modificado, modificando la semilla, el tamaño de la pista cuadrada, y el número de coches. Esto generará una instancia del problema distinta, con una distribución de obstáculos y coches distinta.
Cabe destacar que los distintos coches no compiten por llegar a la meta, si no que colaboran para llegar a la meta.

Para el desarrollo del proceso, se definen cuatro acciones para cada coche, las cuales tienen todas coste uno:

- COCHE *n* ABAJO: Avanza el coche *n* hacia la casilla de abajo.
- COCHE *n* IZQUIERDA: Avanza hacia la casilla de la izquierda.
- COCHE *n* DERECHA: Avanza hacia la casilla de la derecha.
- COCHE *n* ARRIBA: Avanza hacia la casilla de arriba.

En total habrá $num\_coches\ x\ 4$ acciones por lo que el número de acciones posibles dependerá de la instancia del problema.

Todas estas acciones no estarán disponibles siempre, pues habrá momentos en los que un coche se encuentre junto al límite de la pista o un obstáculo y no pueda ir hacia arriba, por ejemplo.
Una vez hemos definido el entorno del problema y las normas de procedimiento, vamos a definir el estado genérico de los coches, pues en base a los datos que contenga, realizaremos los algoritmos de búsqueda.

1.1 Descripción de los estados

Para encontrar una solución adecuada, los algoritmos explorarán distintos estados hasta encontrar el que cumpla las condiciones del estado final.

Un estado genérico en este problema se define por:

- El número de coches que componen el problema
- La posición exacta en la que se encuentra cada coche, indicada mediante coordenadas en los ejes x e y.

En cuanto a los estados finales, en este problema solo pueden dar una situación que generen un estado final y es cuando todos los coches han llegado a la línea de meta.
Si bien es posible que en la generación de la pista de forma aleatoria pudiera llevarnos a una instancia del problema irresoluble (por ejemplo, si hay una barrera de obstáculos cruzando la pista de lado a lado), este trabajo se centrara en las instancias que si tienen solución y no contemplará que no pudiese haberla.

2. Implementación

Partiendo de la información del problema de la práctica, los estados, las acciones que se pueden realizar, y el objetivo de los algoritmos, comentaremos los detalles que hemos valorado a la hora de implementar la práctica. El lenguaje utilizado para la programación de la práctica ha sido Python 3 (En concreto, la versión 3.7).

Hemos optado por realizar la practica en este lenguaje, creando unas clases específicas para los nodos, las acciones posibles, y los estados solución, y los 6 algoritmos que hemos estudiado en clase (búsqueda en anchura, profundidad, profundidad limitada, coste uniforme, primero mejor y A Estrella).

La práctica se estructura de la siguiente forma:

- **Clases**
    - **Nodo**: En esta clase incluimos los **estados**, representados como **listas de tuplas** que **contienen la posición de cada coche que forma parte del problema**. Por ejemplo, si tenemos 3 coches, la lista de sus posiciones podría ser: $[(0,1), (0, 3), (0,4)]$
      Además, contiene información sobre cada nodo, como **el nodo padre**, para poder recorrer la solución encontrada hacia atrás, **el coste** de llegar a ese nodo, la **acción realizada** para llegar hasta él, **el valor de la heurística** (usado en los casos de A Estrella y Primero Mejor), y **el valor de la función de evaluación**.

    - **Acción:** Para **representar las acciones**, hemos utilizado un tipo de datos enumerado, incluyendo las acciones **en el orden**: $\{\mathrm{Abajo}, \mathrm{Izquierda}, \mathrm{Derecha}, \mathrm{Arriba}\}$. Comprobar las acciones posibles en **este orden beneficia la búsqueda de la solución**, dando preferencia a la acción de bajar hacia la meta, evitando explorar nodos innecesarios que realizan acciones en otras direcciones al principio. En definitiva, damos la mayor prioridad a bajar, y la menor prioridad a subir.
      Además, la clase Acción contiene el coche que se ha movido, de modo que podamos identificar qué coche ha cambiado de posición para llegar a cada estado. Cuando un coche realiza la acción *Abajo,* a su posición se le suma $(0,1)$, al ir a la *Izquierda*, $(-1, 0)$, a la *Derecha*, $(1, 0)$, y para subir, $(0, -1)$,

    - **Solución:** Esta clase es utilizada para devolver los datos recogidos al final de cada búsqueda. En esta clase se recogen la **secuencia de acciones que llevan al nodo solución**, el **coste total de la solución**, el número total de **nodos generados**, el número de **nodos expandidos**, el número de **nodos explorados**, el número **máximo de nodos simultáneos en la lista de abiertos**, y el número **máximo de nodos simultáneos en memoria**.

    - **Maze:** Clase básica ofrecida para la práctica, de la que obtenemos la **instancia del problema** mediante $\mathrm{getProblemInstance}(\mathrm{n}, \mathrm{nCars}, \mathrm{seed})$, que nos da una **matriz de tamaño $\boldsymbol{n} * \boldsymbol{n}$**, con un determinado número **de coches en la primera fila, representados con un número**, una serie de **"muros"** representados **mediante el valor $-\mathbf{1}$**, y generado **a partir de una semilla**, de manera aleatoria.

- **Algoritmos:** Se basan en la búsqueda en grafos, de tal manera que evitamos explorar estados repetidos
    - **Búsqueda en Anchura**: Se recorren los nodos del árbol respetando el orden en el que son generados. Cuando exploramos un nodo, añadimos los nodos hijos al final de la lista de nodos abiertos, y vamos explorando el árbol escogiendo el nodo que está en primera posición. (Se expande siempre el nodo que queda a menor profundidad del árbol de búsqueda).

    - **Búsqueda en Profundidad/Profundidad Limitada**: Se recorren los nodos dando preferencia a los nodos que se generan primero, expandiendo antes los nodos que están a mayor profundidad (se expanden los nodos de un camino completo). Los hijos de cada nodo se añaden a la lista de abiertos por el principio, de modo que los nodos que quedan en cabeza siempre están a mayor profundidad. Podemos poner una profundidad límite a este algoritmo, y dependiendo de este límite, podemos tener casos en los que no encontremos solución, si este límite es muy pequeño, o casos en los que el algoritmo recorra todo el árbol de búsqueda como si realizásemos una búsqueda en anchura, si el límite es muy ajustado.
      En este problema, el límite es útil para evitar soluciones a profundidades extremas, que quedan muy lejos de soluciones óptimas.

    - **Coste Uniforme**: Se generan los nodos hijo de cada nodo, y se ordenan en la lista de abiertos de menor a mayor coste, explorando los de menor coste primero. (Es decir, la función de evaluación $f(n)$ tiene en cuenta el coste $g(n)$) Al ser un problema donde el coste de cada acción es 1, el resultado del algoritmo de búsqueda con coste uniforme es equivalente al de búsqueda en anchura.
      $$f(n) = g(n)$$

    - **Primero Mejor**: Se generan los nodos hijo de igual manera que en coste uniforme, pero son ordenados en la lista de nodos abiertos según la heurística, que es el coste esperado de llegar a la solución desde un estado. (La función de evaluación $f(n)$ tiene en cuenta el coste estimado $h(n)$) Este es un algoritmo voraz, que se comporta de manera similar a la búsqueda en profundidad, pero sin el riesgo de llegar a soluciones a una profundidad extrema, ya que al ser una búsqueda informada, tiene en cuenta el valor del coste esperado.

      $$f(n) = h(n)$$

    - **A Estrella**: Los nodos generados se ordenan en la lista de abiertos según el valor de su evaluación. La evaluación de cada nodo se realiza calculando el coste de llegar a ese nodo, mas la suma de las heurísticas de cada coche según su posición. (La función de evaluación $f(n)$ tiene en cuenta el coste $g(n)$ y la heurística $h(n)$)

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

- **Consideraciones y mejoras**
  - **Permutaciones de estados:** Dado que los coches podrían intercambiar posiciones en muchos estados, hemos decidido, además de no considerar estados repetidos, ya que realizamos una búsqueda en grafos, rechazar también las permutaciones de coches que pasan por posiciones repetidas, de tal manera que si tenemos un estado explorado en la lista de cerrados como el siguiente:

    $$[(0,1),(0,3),(0,4)],$$

    si pasamos por un estado como $[(0,1),(0,4),(0,3)]$, lo consideraremos un estado repetido. Hacemos esto en los algoritmos donde no utilizamos una heurística (Anchura, Profundidad y Coste Uniforme). Esto permite evitar la posibilidad explorar estados adicionales que sean producto de intercambios de posición de los coches, lo que supondría explorar nodos con un coste mayor, que podríamos descartar.

  - **Lista de nodos en búsqueda en profundidad:** En lugar de mantener una lista de nodos cerrados con los estados explorados, consideramos la lista de cerrados como la lista de la rama que estamos explorando. Cuando llegamos a un nodo en el que no encontramos solución, realizamos backtracking sobre la rama que estamos explorando, y así tenemos en memoria solo los nodos de la rama que estamos explorando, y los nodos elegibles de la lista de abiertos. Esto también nos da la posibilidad de explorar estados de otros nodos que hayan estado en cerrados, pero que tengan un coste mejor.

  - **Lista de nodos cerrados:** Para simplificar la lista de nodos cerrados, guardamos en la lista según los estados visitados, esto quiere decir que no guardamos nodos diferentes que puedan tener el mismo estado, lo que simplifica la búsqueda en la lista a la hora de comparar si el estado de un nuevo nodo a explorar ha sido visitado o no.

  - **Heurísticas:** Hemos probado 2 heurísticas a lo largo de la práctica, y hemos realizado la muestra de datos con la segunda.
    La primera heurística era la más simple, y consistía en **calcular el coste esperado durante la exploración de cada nodo**, sumando la **distancia de cada coche en línea recta hasta la meta**. Eso se hacía en la exploración de cada nodo lo cual **consume mucho tiempo**, y esta heurística es muy mejorable.

    La segunda heurística, más precisa, **calcula para cada posición** del maze **la heurística correspondiente a dichas posiciones**. Esto se realiza **una sola vez**, lo que ahorra mucho tiempo en la ejecución del algoritmo de búsqueda.

    La generación de dichos valores se realiza mediante un algoritmo que va evaluando las posibles casillas vacías del maze y calculando la distancia que se debe recorrer hasta llegar a la meta desde la casilla evaluada. Este algoritmo **va evaluando las casillas desde la meta y hacia atrás y aprovecha el valor de la casilla anterior para conformar el valor heurístico de esa posición.**
    El valor de la heurística de un estado se calcula sumando el valor en la matriz de cada posición que ocupa cada coche.
    Esta heurística **es optimista** porque calcula exactamente el coste de que un único coche vaya desde esa casilla hasta la meta y sin pasarse. De hecho, es una heurística perfecta para el caso concreto en el que el número de coches es igual a 1. El coste real podrá ser

mayor debido a que cuando hay varios coches, podrían competir por el mismo espacio obligando a hacer movimientos adicionales.

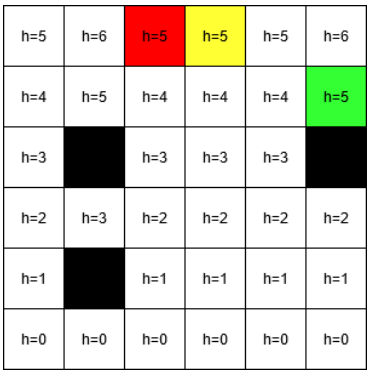

Ilustración 1: Ejemplo de heurística en un estado

En la *Ilustración 1,* podemos ver un ejemplo del cálculo de las heurísticas de cada posición del maze. Tomando como estado inicial $e_i = [(0,2),(0,3),(0,5)]$, el estado de la ilustración representa la acción de bajar una posición el "coche verde":

$$e_1 = [(0,2),(0,3),(1,3)]$$

Teniendo en cuenta que el coste de la acción es 1, y que las heurísticas de cada coche son 5 en todas sus posiciones, el valor de la función de evaluación es:

$$f(e_1) = g(e_1) + h(e_1) = 1 + (5 + 5 + 5) = 16$$

- **Métodos**
  - $\boldsymbol{SumaTuplas(a,b)}$: Utilizada para sumar el valor de una acción la posición de un coche.
  - $\boldsymbol{eInicial(maze,n,ncars)}$: Obtiene el estado inicial de un problema, dado el entorno (maze), el tamaño del maze, y el número de coches. El estado inicial es una lista de tuplas con las posiciones de cada coche.
  - $\boldsymbol{aplicaAccion(estado,accion)}$: Devuelve un nuevo estado, como resultado de aplicar una acción al estado actual, sumando el valor de la acción al coche indicado.
  - $\boldsymbol{Sucesores(listaAcciones,nod)}$: Devuelve la lista de los nodos sucesores, dado un nodo, y las acciones posibles que se pueden realizar. La lista de acciones contiene objetos de la clase *Acción,* con el coche que se mueve, y la dirección. Con el estado del nodo actual, calcula cada nuevo sucesor.
  - $\boldsymbol{AccionesPosibles(maze,n,estado)}$: Devuelve la lista de acciones posibles dado el problema, el tamaño, y el estado inicial. Para cada coche, calcula la posición a la que puede llegar, si no es un muro, no es un extremo del maze, o no es la posición de uno de los otros coches (los coches no se superponen).
  - $\boldsymbol{esSolución(estado,n)}$: Comprueba si el estado pasado como parámetro es solución.

- $\boldsymbol{InicializaHeuristica(n, maze)}$: Recibe el tamaño e instancia del problema y genera una matriz donde almacena los valores de nuestra heurística para cada coche.

- $\boldsymbol{Heuristica(estado)}$: Recibe un estado y usando la matriz generada en *InicializaHeuristica(n, maze)* calcula y devuelve el valor de la heurística para ese estado.

- $\boldsymbol{mazePreview(num, maze)}$: Imprime por consola una vista gráfica de la posición inicial de los coches, el valor de la heurística de cada casilla del maze, y la posición de los muros.

3. Ejecución de la práctica

En este apartado vamos a especificar como se realizan las llamadas para la ejecución de la práctica, desde la consola de Windows o en Linux.
Las llamadas siguen este esquema:

$$\text{python practica1.py} < \text{tamaño del problema} > < \text{nº de coches} > < \text{semilla} >< -l\ limite > \ -- \text{algoritmo}$$

Vemos un ejemplo de una llamada genérica para la ejecución de la práctica:

$$\text{python practica1.py } 5\ 3\ 54 -- \text{ Aestrella}$$

Y en el caso específico de profundidad limitada:

$$\text{python practica1.py } 5\ 3\ 54 - \text{l } 10 -- \text{ profundidad}$$

Los nombres de los algoritmos validos como parámetro son:

anchura, profundidad, AEstrella, primeroMejor y *costeUniforme*

4. Estudio de los resultados de los algoritmos

En este apartado vamos a abordar el estudio de los datos obtenidos tras la ejecución de los algoritmos. Los datos se han obtenido a partir de la ejecución de los algoritmos a partir de unas semillas determinadas, que se incluyen en el paquete de código que ha generado los datos, para poder repetir los experimentos. Se han realizado 25 ejecuciones por cada tamaño y número de coches, de tamaño 5 a 10, y entre 1 y 3 coches (450 ejecuciones por cada algoritmo, para poder obtener datos y medias algo más precisas).

Los datos que vamos a evaluar son **el coste de las soluciones** (que equivale a la profundidad en este problema), **el número de nodos generados** durante la búsqueda**, el número de nodos expandidos** (es decir, el número de nodos de los cuales hemos generado sus hijos), **el número de nodos explorados** (por los cuales hemos preguntado si son solución), **el número máximo de nodos abiertos en memoria** (la cantidad más grande de nodos generados que quedan por explorar**), y el tiempo de ejecución**.

El coste ideal de las soluciones óptimas, si los coches no se encontrasen con ningún obstáculo, sería:

$$(Tamaño\ del\ problema - 1) * \ Nº\ de\ coches$$

Pero esto es muy difícil que se cumpla, excepto en el caso de que el problema sea con un solo coche, pues con más de un coche, lo normal es que aunque uno de ellos pueda llegar a la meta de una tirada, los demás tengan que moverse hacia el camino en el que puedan llegar directamente a la meta, por lo que el coste de la solución sería, suponiendo que algún coche tuviera acceso directo a la meta desde su posición:

$$\big((Tamaño\ del\ problema - 1) * \ Nº\ de\ coches\big) + \sum_{k=0}^{Nºcoches-1} k$$

Por ejemplo, para un problema de tamaño 5 y 3 coches, el coste de la solución ideal sería:

$$\big((5-1) * \ 3\big) + \sum_{k=0}^{3-1} k \rightarrow (12) + (1+2) = 14$$

4.1 Algoritmos de búsqueda no informada

4.1.1 Anchura

Vamos a ver un resumen de los datos obtenidos con la ejecución de la búsqueda en anchura.

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	35,49	12,01	24,75	26,26	0,0002
1	6	5,69	53,88	17,80	40,13	35,15	0,0003
1	7	7,13	75,38	25,12	59,25	45,08	0,0004
1	8	7,95	102,79	33,35	83,29	57,18	0,0006
1	9	9,09	134,34	43,23	112,20	69,11	0,0010
1	10	10,07	162,82	52,37	138,45	81,67	0,0013
2	5	9,27	974,15	178,74	816,50	371,98	0,0194
2	6	11,17	2293,72	395,27	1999,15	737,16	0,0934
2	7	14,15	4018,11	692,63	3584,51	1175,20	0,3011
2	8	15,84	7521,55	1252,02	6909,04	1958,96	1,0429
2	9	18,86	11847,10	1985,49	10998,37	2910,01	2,7080
2	10	21,95	18025,07	2988,01	17001,84	4083,77	6,3919
3	5	14,31	10878,85	1452,50	10158,53	2634,05	1,9923
3	6	17,59	38989,51	4723,73	37187,53	7967,27	24,8141
3	7	21,25	91886,32	11080,72	88380,70	16956,02	148,1134

Tabla 1: Resumen de medias de datos de la ejecución de la búsqueda en anchura.

En primer lugar, vamos a fijarnos (***Tabla 1***) en los costes de las soluciones obtenidas, que al ser un problema en el que todas las acciones tienen el mismo coste (coste 1), coincidirá con la profundidad de la solución. Podemos ver como los costes de las soluciones se aproximan al tamaño del problema por el número de coches. Como sabemos, la búsqueda en anchura nos ofrece soluciones óptimas, y podemos ver que la profundidad media de las soluciones se encuentra en ese lugar.
Este algoritmo explora todos y cada uno de los nodos generados hasta llegar a la solución, lo que implica problemas en cuanto a memoria (para tamaños de problema muy grandes, o varios coches, el espacio que ocupa la lista de nodos abiertos es muy grande), y en cuanto a tiempo, debido a que recorrer estas listas de nodos tan grandes cuando el tamaño de problema es muy grande es muy costoso.

De hecho, con este algoritmo (y con la búsqueda con coste uniforme, que como veremos, es equivalente a este) no ha sido posible encontrar soluciones en un tiempo óptimo para tamaños de problema mayores que 7, y con más de 2 coches, pues con tamaño 7 y 3 coches, tarda alrededor de 2 minutos y medio en encontrar una solución, y si subimos a tamaño 8 con 3 coches, el tiempo empleado es inadmisible.

4.1.2 Profundidad

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	14,51	4,48	5,53	14,41	0,0001
1	6	5,76	18,72	5,79	6,88	18,59	0,0001
1	7	7,28	23,84	7,33	8,39	23,70	0,0001
1	8	8,24	27,00	8,46	9,69	26,40	0,0002
1	9	9,36	31,83	9,78	11,22	30,70	0,0002
1	10	10,17	33,28	10,30	11,50	32,86	0,0002
2	5	10,35	52,16	10,35	11,77	51,67	0,0002
2	6	13,43	72,88	13,43	15,05	71,98	0,0003
2	7	18,40	94,30	18,40	20,12	93,17	0,0004
2	8	19,62	107,99	19,62	21,83	105,94	0,0004
2	9	23,88	120,87	23,88	26,09	119,17	0,0005
2	10	40,12	212,67	40,12	48,51	200,70	0,0010
3	5	18,62	118,95	18,62	21,34	116,55	0,0005
3	6	22,78	159,61	22,78	24,99	157,95	0,0006
3	7	28,18	192,28	28,18	31,00	189,96	0,0007
3	8	35,85	248,03	35,85	40,09	243,86	0,0010
3	9	46,21	313,34	46,21	52,54	306,36	0,0014
3	10	47,91	350,91	47,91	51,53	346,86	0,0015

Tabla 2: Resumen de medias de datos de la ejecución de la búsqueda en profundidad.

La diferencia más notable, observando la ***Tabla 2***¸ a la hora de ejecutar este algoritmo es el tiempo de ejecución. Mientras que la búsqueda de soluciones en anchura puede tardar horas en tamaños grandes y varios coches, este algoritmo de carácter voraz encuentra soluciones en un tiempo muy reducido, del orden de milisegundos en tamaños grandes, y microsegundos en tamaños pequeños y 1 o 2 coches.

También cabe destacar la diferencia que marca este algoritmo en cuanto al número de nodos generados, y al número de nodos mantenidos en memoria. El número de nodos expandidos normalmente coincide con la profundidad de la solución, ya que explora ramas completas, encontrando soluciones en ellas.

Algo muy característicos que observamos en los datos es la profundidad de las soluciones encontradas. Debido al orden en el que consideramos las acciones que realizan los coches (dando preferencia a bajar), y a que acotamos la búsqueda eliminando estados que sean repetidos, y posibles permutaciones (que no son muy habituales en la búsqueda en profundidad), este algoritmo encuentra soluciones con costes que no son muy elevados, considerando la rapidez con la que se ejecuta. Como veremos a continuación, las soluciones ofrecidas no están a un coste muy elevado, pero si hay casos con tamaños más grandes en los que se producen soluciones con profundidad no admisible, aunque no son muchas.

Los nodos abiertos en memoria son muchos menos que en la búsqueda en anchura porque solo expandimos los nodos correspondientes a la rama que estamos explorando, y en nuestro caso, como usamos la lista de nodos explorados como una lista de cerrados en la que guardamos la rama que estamos explorando (como comentamos en el *Apartado 2, página 5:* ***Lista de nodos en búsqueda en profundidad***), el tamaño de la lista de explorados se reduce, y nos permite ver que se corresponde casi siempre con el coste de la solución, lo que indica que n muy pocos casos encuentra soluciones en ramas distintas a la que explora, es decir, en pocas ocasiones se realiza backtracking.

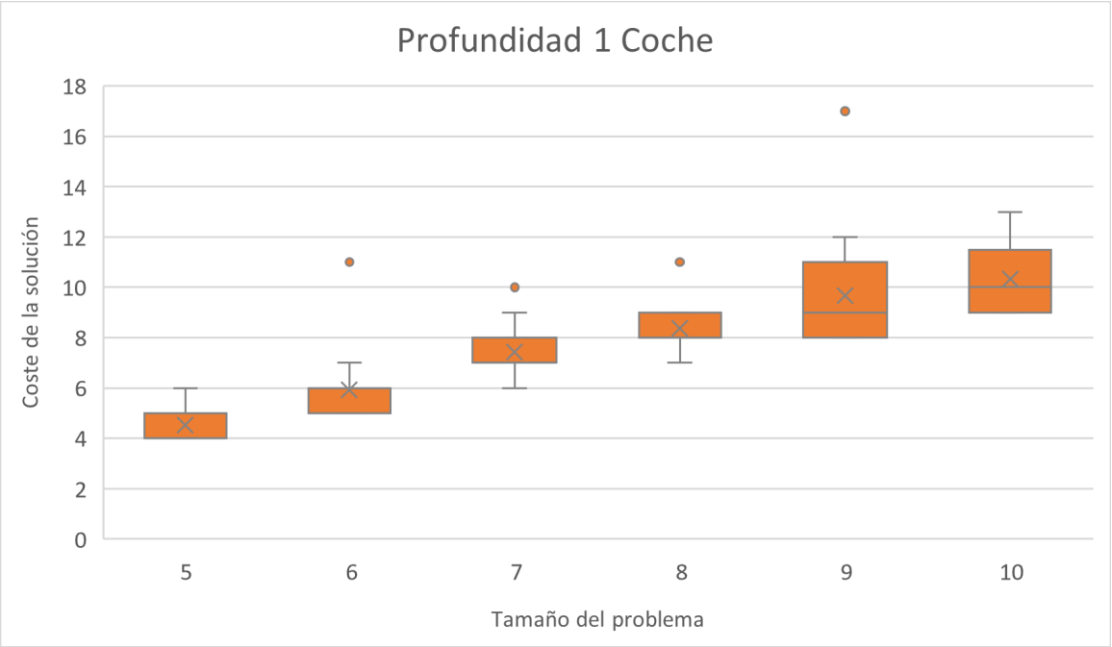

Ilustración 2: Costes obtenidos en búsqueda en profundidad con 1 coche.

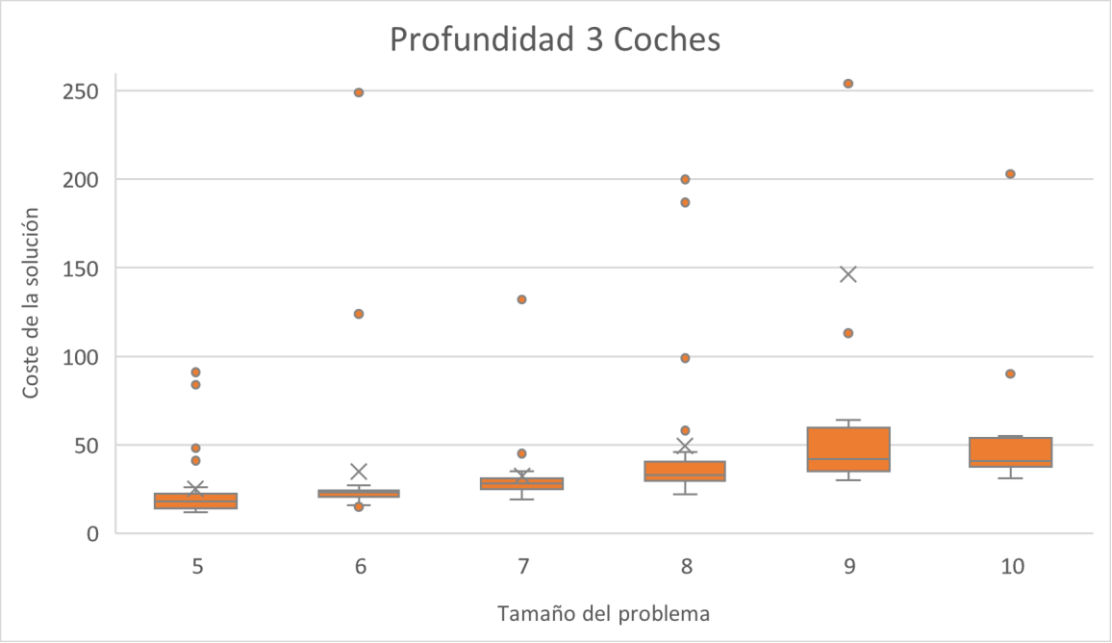

Ilustración 3: Costes obtenidos en búsqueda en profundidad con 3 coches.

Como podemos observar en la ***Ilustración 2****,* en los tamaños 6, 7, 8 y 9, solo una solución de las 25 ejecuciones de cada tamaño nos da un coste excesivo en comparación con la media. Sin embargo, en la ***Ilustración 3***, podemos ver como muchas más ejecuciones en todos los tamaños nos dan costes que se salen de los márgenes de las soluciones óptimas, por lo que cuanto más aumente el tamaño y el número de coches, es menos seguro que obtengamos soluciones cercanas al óptimo.

Aun así, considerando que este algoritmo voraz suele ofrecer soluciones que quedan muy lejos del óptimos, puede ser necesario tenerlo en cuenta a la hora de querer resolver problemas de gran tamaño, si el tiempo de ejecución es un factor importante para nosotros., ya que en este problema las soluciones no se alejan demasiado de las óptimas.

4.1.3 Profundidad limitada

Dado que las soluciones que encuentra la búsqueda en profundidad son muy cercanas a las óptimas que proporciona anchura, no tiene mucho sentido en algunos escenarios poner un límite, pero si queremos acotar los valores extremos que produce con tamaños altos, vamos a aplicar un límite un 50% (aunque existen soluciones que quedan por encima de esa profundidad, ya que la situación de los "muros" es muy desfavorable) mayor que la solución óptima, la cual calculamos con la fórmula:

$$1.5 * \left[\left((Tamaño\ del\ problema - 1) * \ Nº\ de\ coches \right) + \sum_{k=0}^{Nºcoches-1} k \right]$$

Por ejemplo, para tamaño 5 y 3 coches, el límite sería:

$$1.5 * \left[\left((5 - 1) * \ 3 \right) + \sum_{k=0}^{3-1} k \right] \rightarrow 14 * 1.5 = 21$$

Tomando la parte entera (en caso de que obtuviéramos decimales), el límite lo establecemos en 21.

Vamos a ver los datos obtenidos de la ejecución con estos límites.

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	14,51	4,48	5,53	14,41	0,0001
1	6	5,69	18,65	5,79	6,93	18,25	0,0002
1	7	7,18	24,48	7,48	8,74	23,59	0,0002
1	8	8,06	28,62	8,87	10,29	26,35	0,0003
1	9	9,19	31,71	9,76	11,34	30,30	0,0003
1	10	10,17	33,28	10,30	11,50	32,86	0,0003
2	5	9,79	58,38	11,43	13,17	50,31	0,0004
2	6	12,61	79,63	14,51	16,13	68,56	0,0005
2	7	16,42	109,06	21,09	22,51	83,68	0,0003
2	8	17,84	121,72	21,57	23,43	98,61	0,0003
2	9	22,13	140,85	27,52	29,63	113,00	0,0009

Tabla 3: Resumen de medias de datos de la ejecución de la búsqueda con Profundidad Limitada.

El problema principal que hemos encontrado al ejecutar la búsqueda con profundidad limitada es el tiempo cuando la solución se encuentra en un extremo del árbol. Entonces, la búsqueda se convierte en una búsqueda en anchura, pero en algunos casos puede tardar mucho más. Tanto es así, que no ha sido posible encontrar soluciones en tiempos razonables a partir de 2 coches, y tamaño 9.

Además, la media de tiempos de ejecución mostrados en la ***Tabla 3*** no refleja fielmente las desviaciones en cuanto a tiempos de ejecución, que en muchos casos son muy superiores a los tiempos de la búsqueda en profundidad, e incluso en anchura, ya que se analizan los caminos con mayor profundidad (la del límite calculado) antes de llegar a la solución, que puede estar muy lejos, e incluso no ser encontrada, ya que en situaciones desfavorables las soluciones quedan fuera del límite.

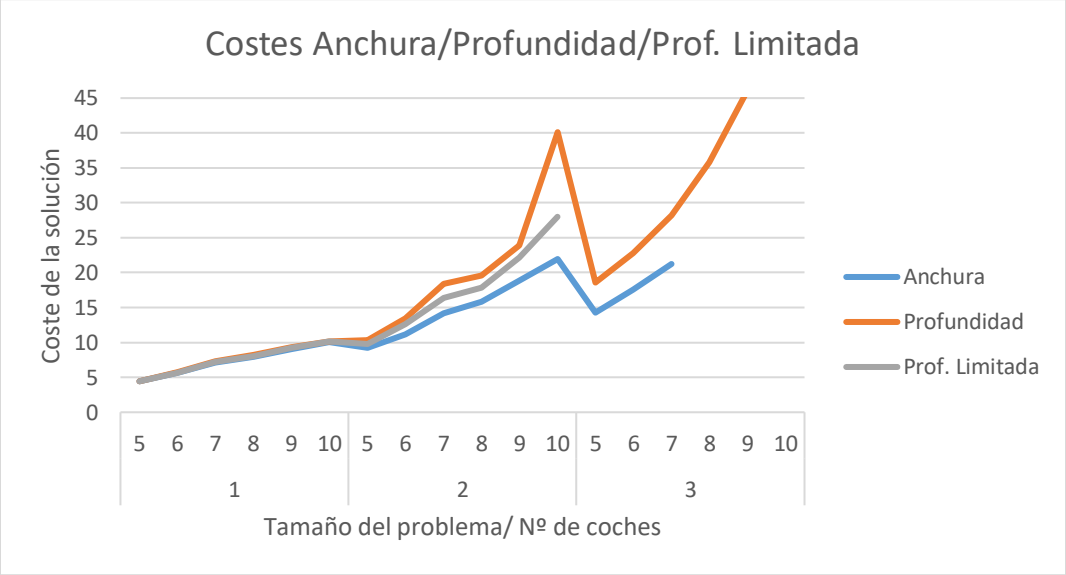

Ilustración 4: Comparación de costes de las soluciones entre Anchura, Profundidad y Profundidad Limitada

En la ***Ilustración 4*** podemos observar como las soluciones obtenidas se acercan a las óptimas de la búsqueda en anchura, pero se hace imposible continuar la ejecución del algoritmo a partir de 2 coches y tamaño 10 debido al coste en tiempo. Con 1 coche, la diferencia no es significativa en cuanto tiempos de ejecución ni costes de la solución.

Para comprender el problema de los tiempos de ejecución, vamos a echar un vistazo a la ***Ilustración 5:***

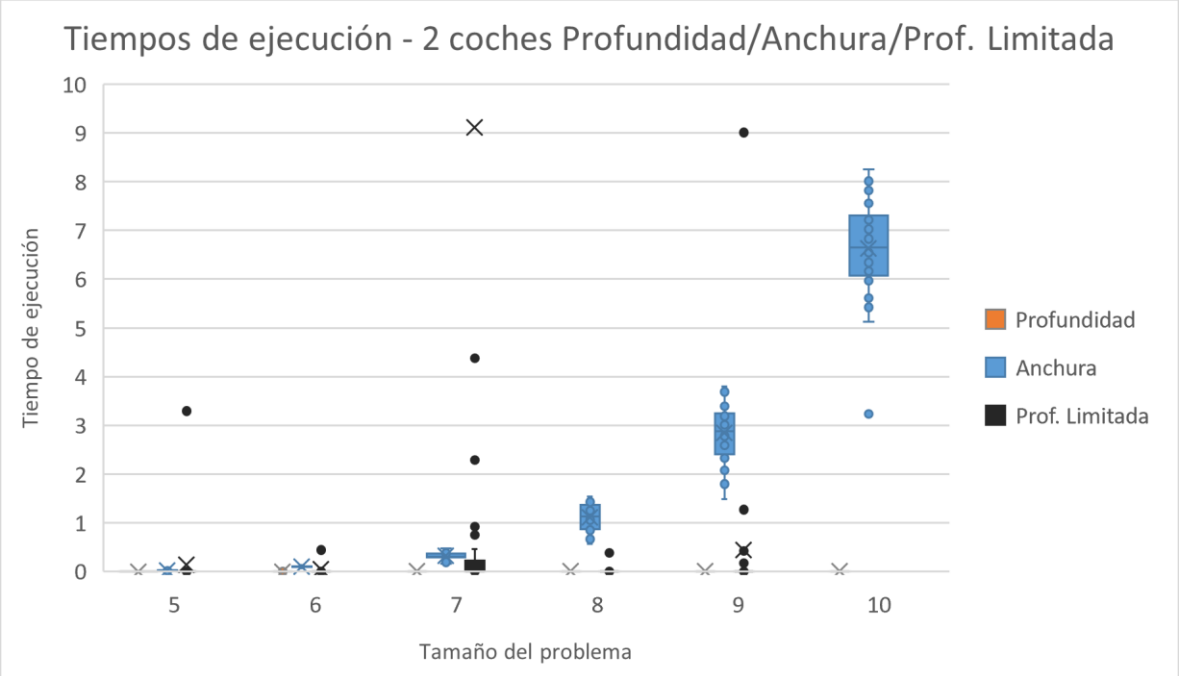

Ilustración 5: Diferencias en tiempo de ejecución entre la Búsqueda en Profundidad, Anchura y Profundidad Limitada.

Para comprender la ***Ilustración 5***, debemos saber que, para cada tamaño del problema, el primer elemento (de color naranja, y no visible, excepto el símbolo que marca la media) representa el diagrama de cajas del tiempo de ejecución de la búsqueda en profundidad. Al ser del orden de milisegundos, quedan fuera del gráfico.
El segundo elemento representa los diagramas de cajas de los tiempos de ejecución de la búsqueda en anchura, cuyos tiempos de búsqueda varían desde las décimas de segundo, hasta los 9 segundos en casos extremos, subiendo el tamaño del problema.
Como podemos observar en cuanto a los resultados de la búsqueda con profundidad limitada, las **medias de tiempos se sitúan por debajo de las medias de los tiempos de anchura, pero se producen búsquedas con tiempos mucho mayores cuando las soluciones son más difíciles de encontrar**, y que no se aprecian en las medias. Por lo tanto, **el uso de este algoritmo no es muy recomendable, debido al tiempo tan variable para encontrar una solución, y por la posibilidad de no encontrar la solución con el limite impuesto**. Como veremos, **es mucho más eficiente utilizar la búsqueda Primero Mejor, o la búsqueda AEstrella**. Con tamaño de problema 10, ni siquiera es factible buscar soluciones debido al tiempo que supone.

4.2 Algoritmos de búsqueda informada

4.2.1 Coste Uniforme

Este algoritmo nos proporciona los mismos resultados en cuanto a costes de solución y nodos explorados que la búsqueda en anchura, debido a que el coste de cada acción es el mismo (1), con diferencias muy poco significativas en cuanto al tiempo de ejecución. Dado que los estados generados se ordenan por coste y por antigüedad, la lista de nodos abiertos es la misma, y se van explorando en el mismo orden. Veamos los datos obtenidos en la ***Tabla 4.***

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	35,49	12,01	24,75	26,26	0,0002
1	6	5,69	53,88	17,80	40,13	35,15	0,0003
1	7	7,13	75,38	25,12	59,25	45,08	0,0005
1	8	7,95	102,79	33,35	83,29	57,18	0,0007
1	9	9,09	134,34	43,23	112,20	69,11	0,0010
1	10	10,07	162,82	52,37	138,45	81,67	0,0014
2	5	9,27	974,15	178,74	816,50	371,98	0,0195
2	6	11,17	2293,72	395,27	1999,15	737,16	0,0993
2	7	14,15	4018,11	692,63	3584,51	1175,20	0,3117
2	8	15,84	7521,55	1252,02	6909,04	1958,96	1,0666
2	9	18,86	11847,10	1985,49	10998,37	2910,01	2,7481
2	10	21,95	18025,07	2988,01	17001,84	4083,77	6,4006
3	5	14,31	10878,85	1452,50	10158,53	2634,05	2,0050
3	6	17,59	38989,51	4723,73	37187,53	7967,27	23,9692
3	7	21,25	91886,32	11080,72	88380,70	16956,02	147,1514

Tabla 4: Resumen de medias de datos de la ejecución de la búsqueda con Coste Uniforme.

Una diferencia con respecto a la búsqueda en anchura es que el tiempo de ejecución puede ser algo mayor, debido a que este algoritmo realiza las comparaciones de coste a la hora de insertar los nodos generados en la lista de abiertos, cosa que la búsqueda en anchura no hace, ya que los inserta directamente en el momento en el que se generan, por lo que el tiempo empleado en realizar las operaciones de comparación de los costes puede aumentar el tiempo de ejecución, aunque no es una diferencia significativa.

Además, **en los algoritmos de búsqueda informada, hemos utilizado Colas de prioridad** (*heap en Python*) para la lista de nodos abiertos, **que aligeran la búsqueda y la inserción de nodos**, simplificando la reordenación de esta, dado que, para tamaños muy grandes, y varios coches, **reordenar una lista simple con cada inserción supone un coste en tiempo muy alto.**

4.2.2 Primero Mejor

Este algoritmo de carácter voraz realiza la búsqueda de las soluciones teniendo en cuenta el valor de la heurística que hemos empleado, en tiempos comparables a los de la búsqueda en profundidad, pero a diferencia de este, no nos ofrece soluciones a profundidades que no son admisibles.

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	14,44	4,44	5,45	14,44	0,0001
1	6	5,69	18,32	5,69	6,70	18,32	0,0002
1	7	7,13	23,27	7,13	8,14	23,27	0,0002
1	8	7,95	26,13	7,95	8,96	26,13	0,0002
1	9	9,09	30,47	9,09	10,10	30,47	0,0003
1	10	10,07	32,90	10,07	11,08	32,90	0,0003
2	5	9,31	48,72	9,31	10,32	48,72	0,0003
2	6	11,22	64,72	11,22	12,23	64,72	0,0003
2	7	14,31	78,83	14,31	15,32	78,83	0,0004
2	8	15,93	93,78	15,93	16,94	93,78	0,0005
2	9	18,98	106,40	18,98	19,98	106,40	0,0006
2	10	22,04	129,60	22,04	23,05	129,60	0,0007
3	5	14,41	101,70	14,98	16,06	101,63	0,0005
3	6	18,02	138,55	18,30	19,31	138,55	0,0007
3	7	21,53	166,87	22,27	23,35	166,74	0,0008
3	8	24,81	198,58	25,23	26,28	198,51	0,0010
3	9	28,32	227,90	28,80	29,91	227,70	0,0011
3	10	31,57	270,48	32,06	33,07	270,48	0,0013

Tabla 5: Resumen de medias de datos de la ejecución de la búsqueda Primero Mejor.

Como podemos ver en la ***Tabla 5***, los tiempos de ejecución similares a los de la búsqueda en anchura, con muy poca diferencia. Además, incluso conseguimos generar menos nodos, al explorar directamente el camino que mejor coste esperado nos otorga, gracias a la heurística, evitando hacer backtracking. Los costes de las soluciones son mucho más ajustados, y sin picos como nos pasaba en la búsqueda en profundidad.

Podemos ver en la ***Ilustración 6***, como las soluciones que ofrece el algoritmo de búsqueda en profundidad son de una profundidad mayor cuando aumentamos el tamaño del problema y el número de coches, **pero las soluciones que ofrece la búsqueda Primero Mejor son mucho más ajustadas a las soluciones óptimas**.

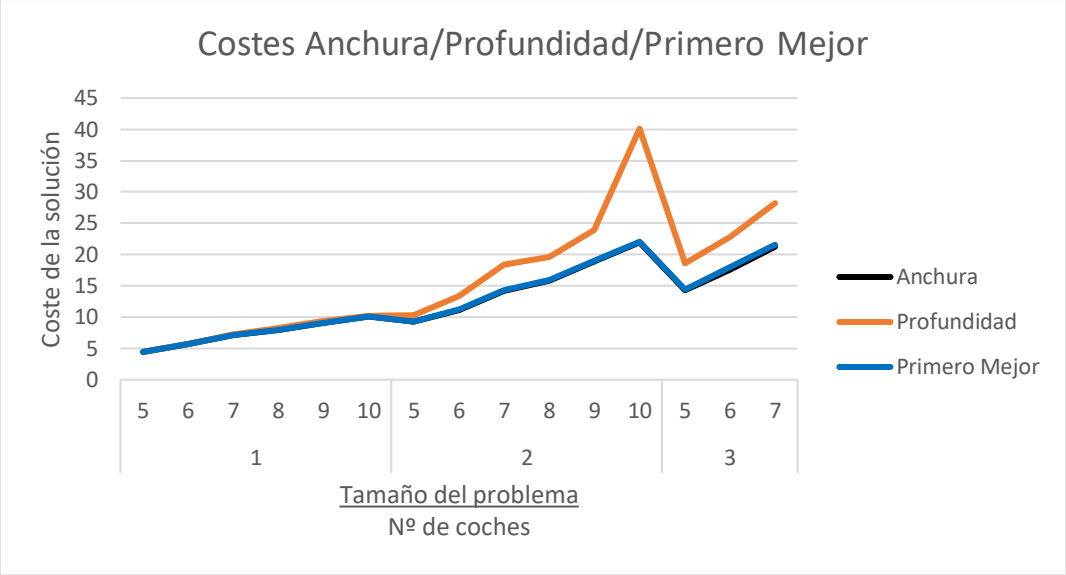

Ilustración 6: Comparación de costes de las soluciones entre Anchura, Profundidad y Primero Mejor.

4.2.3 AEstrella

Este algoritmo utiliza la heurística y la información del coste del camino recorrido para encontrar soluciones óptimas. Es el que **nos ofrece las mejores soluciones en un tiempo razonable**, dándonos la posibilidad de resolver problemas de mayor tamaño y con mayor número de coches sin preocuparnos por el tiempo, como si lo hacíamos con la búsqueda en profundidad o con coste uniforme. Vamos a ver los datos obtenidos.

Coches	Tamaño	Coste	Nodos Generados	Nodos Expandidos	Nodos Explorados	Max. nodos en memoria	Tiempo de ejecución (s)
1	5	4,44	15,72	4,83	6,05	15,62	0,0001
1	6	5,69	20,13	6,23	7,64	19,74	0,0002
1	7	7,13	28,52	8,79	10,86	27,42	0,0002
1	8	7,95	29,48	8,99	10,70	28,73	0,0003
1	9	9,09	35,29	10,42	12,41	34,22	0,0003
1	10	10,07	40,35	12,18	14,21	39,29	0,0004
2	5	9,27	165,54	29,25	40,81	155,42	0,0009
2	6	11,17	247,60	40,03	57,34	231,71	0,0013
2	7	14,15	421,91	70,37	104,22	388,05	0,0027
2	8	15,84	540,14	83,66	125,77	498,57	0,0035
2	9	18,86	840,02	134,35	217,30	754,79	0,0071
2	10	21,95	1139,78	178,80	298,45	1017,42	0,0108
3	5	14,31	1400,66	174,56	281,84	1274,31	0,0109
3	6	17,59	2436,37	275,60	452,52	2235,95	0,0215
3	7	21,25	3825,29	456,69	813,16	3410,39	0,0456
3	8	24,44	4807,96	521,90	831,78	4463,83	0,0583
3	9	28,07	8356,46	888,51	1516,28	7637,25	0,1436
3	10	31,45	8514,58	885,10	1435,42	7893,23	0,1258

Tabla 6: Resumen de medias de datos de la ejecución de la búsqueda AEstrella.

Como podemos observar en la ***Tabla 6*, el tiempo de ejecución se sitúa en el orden de las décimas de segundo** para casos en los que **la búsqueda en anchura podría tardar horas**. Esto también se debe a la eficiencia de las colas de prioridad que hemos utilizado en los algoritmos de búsqueda informada, ya que la inserción y búsqueda de nodos en las colas de prioridad es mucho más eficiente que el uso de listas de tuplas comunes.

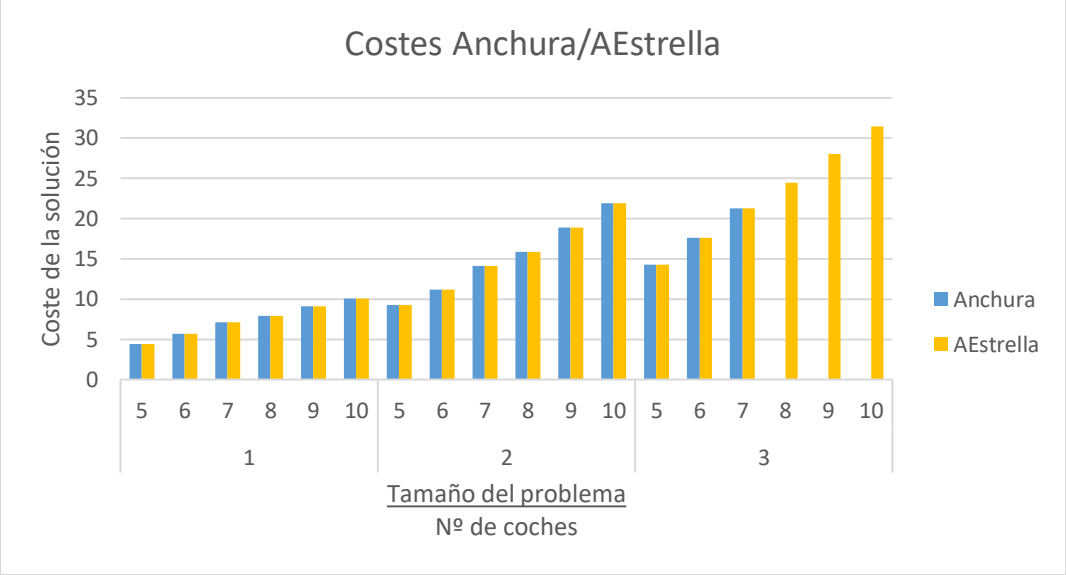

Ilustración 7: Comparación de costes de las soluciones entre Anchura y AEstrella.

Como vemos en la ***Ilustración 7***, los resultados ofrecidos por AEstrella son óptimos, dando soluciones con costes iguales a los de la búsqueda en anchura, pero en tiempos mucho menores, ya que con este algoritmo si que podemos explorar laberintos de tamaño 3 y más de 7 coches.

Además, en cuanto a la generación de nodos, AEstrella genera del orden de 10 veces menos nodos que la búsqueda en anchura en tamaños grandes (a partir de 2 coches), y la mitad de nodos con un solo coche. Esto se debe a que este algoritmo solo abre los nodos con mejor evaluación, enfocando la búsqueda, lo que evita expandir niveles enteros en el árbol de búsqueda. Además, como comentamos en el *Apartado 2, página 6:* ***Heurísticas***, sobre la segunda heurística, que es la que hemos utilizado, el hecho de calcular previamente la heurística para cada posición del maze reduce el tiempo que se emplea en la realización de operaciones, puesto que solo tenemos que obtener el valor de la función de evaluación para la posición de cada coche, y sumarlas para obtener la evaluación del estado, junto con el coste.

5. Conclusiones

- Las soluciones que nos da la **búsqueda en anchura son óptimas porque el coste de realizar una acción es unitario, siempre es igual, por lo tanto, la profundidad de la solución coincide con el coste**. A menor profundidad, la solución es más optima, y la búsqueda en anchura siempre encuentra las soluciones a la menor profundidad, porque recorre el árbol por niveles. El tiempo de ejecución aumenta mucho con respecto al tamaño del problema y el número de coches, y el coste en memoria es muy alto, ya que guardamos todos los nodos en memoria.

- La **búsqueda en profundidad** nos otorga **soluciones con profundidades relativamente cercanas** a las óptimas porque **evaluamos primero la acción de ir hacia abajo en el laberinto, lo que nos garantiza encontrar antes la solución, en un tiempo muy reducido.** Sin embargo, para tamaños más grandes, y más coches, es más difícil encontrar un camino óptimo que vaya directo a la solución, y la posición de los "muros" en el problema dificulta la búsqueda, por lo que **se pueden producir soluciones que quedan a costes muy profundos, y que son inadmisibles. Es el que menor coste en memoria supone, ya que solo guarda el camino que explora.**

- En cuanto a la búsqueda con **profundidad limitada**, el límite puesto ayuda a encontrar soluciones óptimas, pero es mucho mas rentable utilizar la búsqueda AEstrella, **ya que en muchos casos, la profundidad limitada tiende a comportarse como la búsqueda en anchura en cuanto al tiempo empleado** (incluso no siendo posible en ocasiones encontrar la solución), explorando todo el árbol hasta encontrar la solución. Esta es la razón por la que las soluciones óptimas se alejan de la fórmula del coste ideal.

$$\big((Tamaño\ del\ problema - 1) * \ Nº\ de\ coches\big) + \sum_{k=0}^{Nºcoches-1} k$$

  Ya que esta supone que los coches se encuentran todos seguidos, y la salida a los obstáculos está bajo alguno de ellos. En muchas instancias del problema, el camino se encuentra alejado de los coches, y los propios coches están dispersos, por lo que la solución óptima está a una profundidad algo mayor de la que obtenemos mediante la fórmula.

- La **búsqueda con coste uniforme** es **equivalente a la búsqueda en anchura** debido a que el coste de las acciones es el mismo, y quedan ordenadas en la lista de elegibles por orden de generación, lo que hace que el recorrido de la lista de abiertos sea igual que en anchura. La evaluación del coste de cada nodo puede hacer aumentar el tiempo de ejecución

- **Primero Mejor** es un algoritmo de carácter voraz que mejora los resultados de la búsqueda en profundidad en cuanto a profundidad de la solución, con tiempos solamente por debajo de la búsqueda en profundidad, por lo que **es la mejor opción frente a la búsqueda en profundidad si queremos un tiempo rápido y soluciones más cercanas al óptimo.** Las soluciones no son óptimas, pero son aceptables si esperamos soluciones que no se desvíen mucho en un pequeño periodo de tiempo.

- **AEstrella es la mejor opción en cuanto a optimalidad de la solución y tiempo empleado en la búsqueda**. En este problema nos ha ofrecido **soluciones iguales, o muy similares** (variando el orden de algunos movimientos) **a un coste en tiempo memoria muy reducido,** y además **nos permite explorar tamaños y cantidades de coches que la búsqueda en anchura no nos permite debido al tiempo y al uso de memoria**, por lo que es el más adecuado para usar en este tipo de problemas, gracias a la función de evaluación basada en el coste del camino recorrido y el coste esperado de llegar a la solución.