Práctica Inteligencia Artificial

Autores:

Juan José Garzón Luján (100363861) → 100363861@alumnos.uc3m.es
Carlos García Cañibano (100363813) → 100363813@alumnos.uc3m.es

Introducción

En este documento especificaremos los detalles relativos a la implementación en Python del problema de control de un dron de reconocimiento. En nuestro caso, el enfoque a la hora de solucionar el enunciado ha sido el de programar una alternativa lo más modular y “trazable” posible.

Descripción de la tarea realizada

En este apartado describiremos la funcionalidad de los métodos que permiten al dron de reconocimiento obtener las acciones a realizar, el resultado de las mismas, comprobar si ya ha terminado, como varían los costes de la solución final dependiendo del terreno y la definición de una heurística admisible. Todo este proceso de decisión está apoyado en la librería simpleai que nos aporta los diferentes algoritmos de búsqueda utilizados.

- **def actions(self, state):**

Esta función utiliza una lista de posibles acciones que el dron puede ejecutar codificadas con los cuatro puntos cardinales dentro de un eje espacial de dos dimensiones. En nuestro caso, [Norte, Sur, Este, Oeste].

Partiendo de un estado inicial en el que todas las acciones son posibles, vamos restringiendo la lista de acciones finalmente aceptables por nuestra máquina en función de las limitaciones que nos impone el enunciado o de las propias dimensiones del mapa. Por ejemplo, las casillas de tipo mar no se pueden atravesar. Utilizamos el atributo config de la propia clase y otro tipo de atributo definido en el archivo config.py denominado ‘water’ para las casillas de tipo ‘sea’, de modo que son reconocibles utilizando la función dada ‘getAttribute(...)’.

Después de filtrar las acciones, se retorna una lista con las acciones aceptables que puede realizar el dron.

- **def result(self, state, action):**

Cuando el algoritmo de búsqueda elegido ha decidido cual es la acción que quiere debe realizar el dron, utiliza esta función para saber cual es el resultado de la misma. Básicamente actualizamos nuestro estado actual para que represente el siguiente dentro de nuestra búsqueda de la solución. Actualizamos las coordenadas y realizamos una tarea adicional que tiene que ver con el cumplimiento del objetivo del dron.

Comprobamos con un atributo definido en el archivo de configuración config.py llamado ‘isGoal’ si la casilla actual donde nos encontramos es un lugar de interés donde realizar una fotografía. También comprobamos, para evitar que se dupliquen las adiciones, que la casilla actual no pertenezca al conjunto de datos que ya hemos recolectado con anterioridad.
Para actualizar nuestra lista de casillas fotografiadas de cara a saber cuando hemos terminado, tenemos que realizar una transformación entre tupla/lista en ambos sentidos dado el carácter inmutable de las tuplas en Python. El retorno final es un estado construido que representa el siguiente paso para que el algoritmo de búsqueda pueda seguir trabajando.

- **def is_goal(self, state):**

Aquí nos encargamos de comprobar si el estado actual en el que nos encontramos es el estado final para saber cuando tenemos que terminar.

Para realizar esta comprobación, nos valemos de la tuplas de casillas fotografiadas que contienen el estado actual y el final (definido en la clase como un atributo). Para comprobar dos tuplas, al ser tipos inmutables, necesitamos ordenarlas puesto que la posición de los datos es relevante. Una vez hecho esto, solo hay que comprobar si contienen exactamente los mismos elementos y si las coordenadas de ambos estados descritos anteriormente coinciden.

- **def cost(self, state, action, state2):**

Función encargada de retornar los costes de atravesar cada tipo de terreno que puede existir dentro de nuestro mapa. Los costes son elegidos según nuestro criterio y están definidos como atributo de cada casilla (‘cost’) en el fichero config.py:

1. Desert → Coste de 3 unidades de trabajo.
2. Plains → Coste de 1 unidad de trabajo.
3. Hills → Coste de 2 unidades de trabajo.
4. Forest → Coste de 5 unidades de trabajo.
5. Goal → Coste de 1 unidad de trabajo.
6. Drone-base → Cose de 0 unidades de trabajo.

Tengamos en cuenta que sus variantes ‘ya atravesadas’ cuestan el mismo trabajo.

- **def setup (self):**

Usada para la definición de parámetros iniciales que utilizará nuestro agente. Un estado inicial, un estado final y un algoritmo extraído de la librería simpleai.

Hemos decidido diseñar la solución del problema estableciendo el estado inicial y final de la siguiente forma:

initial_state = ((x, y), ())

El primer campo de la tupla que conforma el estado inicial es otra tupla que contiene las posiciones x e y dentro de los ejes cartesianos y codifican el punto exacto donde nos encontramos. El segundo campo es una tupla vacía donde almacenaremos todas las casillas fotografiadas hasta el momento. Inicialmente, está vacía.

final_state = ((x, y), (todas las casillas goal))

El primer campo se corresponde con las mismas coordenadas en las que el agente empieza a moverse porque el objetivo es realizar todas las fotografías y volver al punto de partida.

- **Relativo al diseño de las heurísticas:**

Hemos intentado implementar una variante ‘vitaminada’ de la distancia de Manhattan convencional.

Nuestros primeros pensamientos a la hora de diseñar la heurística fueron que debía existir un compromiso entre la distancia a la que el dron se alejaba de su punto de retorno y a la vez se acercaba a sus casillas objetivo.

Por otra parte, también caímos en cuenta de que nuestro mapa está compuesto por los ejes cartesianos. Esto nos planteó el problema de intentar aproximar una distancia ‘absoluta’ o ‘relativa’ a las casillas objetivo respecto de la posición actual de nuestro dron. En primera instancia pensamos que una aproximación del teorema de Pitágoras sería lo más acertado. Sin embargo, una media clásica entre la distancia medida en ambos ejes, también proporciona una medición fiable y aporta información a la búsqueda de la solución.

Por último, también creímos adecuado incluir los costes de las casillas candidatas a ser atravesadas por nuestra máquina para que tuviese ‘cierto poder de decisión’ también a la hora de elegir el terreno.

En definitiva, nuestra heurística que, seguro mejorable, pero en nuestro punto de vista razonable, tiene este aspecto:

$$h = (distanciaObjetivo/2) + (distanciaBase/2) + costeCasilla$$

Sin embargo, como somos curiosos, hemos también implementado una variante de la primera y sencilla heurística que calcularía “tal cual” una solución basada en el teorema de Pitágoras. La apariencia de la fórmula es parecida, pero en esta ocasión no utilizamos una aproximación de la distancia diviendo entre 2 por ser este el número de ejes, si no que intentamos calcular con mayor fidelidad cual sería la distancia.

$$h1 = (distanciaObjetivo) + (distanciaBase) + costeCasilla$$

En la etapa de experimentación enfrentaremos nuestras dos heurísticas y trataremos de extraer alguna conclusión final.

Experimentación

En esta primera tabla de experimentación hemos intentado hacer una representación general de cual es el funcionamiento del dron en diferentes mapas teniendo en cuenta distintos parámetros de su ejecución. Hemos variado dimensiones, cantidad de casillas objetivo y semilla generadora del mapa.

El objetivo de la comparación ha sido intentar percibir las distintas ventajas, desventajas y propiedades de los diferentes algoritmos de búsqueda que nos proporciona la librería utilizada.

Tabla 1. (Escenarios divididos por las líneas de mayor grosor)

Primer escenario con un mapa con semilla = 777: Comparación del coste en tiempo (iterations)

de la solución para distintos escenarios y algoritmos.

Segundo escenario con un mapa con semilla = 666: Comparación de la longitud y el coste de la solución para los distintos algoritmos.

Tercer escenario con un mapa con semilla = 555: Comparación del coste en memoria de los diferentes algoritmos.

Mapa	***Número de Objetivos***	***Amplitud***	***Profundidad***	***Greedy***	***A****
Pequeño (5x5 casillas)	5	Iteraciones: 1552	Iteraciones: 58	Iteraciones: 28	Iteraciones: 595
Mediano (10x10 casillas)	5	Iteraciones: 10446	Iteraciones: 308	Iteraciones: 126	Iteraciones: 7429
Grande (12x12 casillas)	5	Iteraciones: 18061	Iteraciones: 467	Iteraciones: 254	Iteraciones: 13144
Pequeño (5x5 casillas)	6	Longitud: 15 Coste: 33	Longitud: 61 Coste: 177	Longitud: 17 Coste: 33	Longitud: 15 Coste: 29
Mediano (6x8 casillas)	6	Longitud: 23 Coste: 57	Longitud: 89 Coste: 257	Longitud: 23 Coste: 53	Longitud: 23 Coste: 53
Grande (7x8 casillas)	6	Longitud: 25 Coste: 61	Longitud: 89 Coste: 259	Longitud: 49 Coste: 133	Longitud: 25 Coste: 59
Pequeño (4x8 casillas)	6	Nodos lista abierta: 1784	Nodos lista abierta: 48	Nodos lista abierta: 63	Nodos lista abierta: 1419
Mediano (7x5 casillas)	6	Nodos lista abierta: 4043	Nodos lista abierta: 72	Nodos lista abierta: 232	Nodos lista abierta: 3551
Grande (6x6 casillas)	6	Nodos lista abierta: 3224	Nodos lista abierta: 73	Nodos lista abierta: 114	Nodos lista abierta: 2149

Para nuestra segunda tabla de experimentación relativa a los problemas avanzados, hemos comparado en dos casos diferentes con el mismo mapa la eficacia de las dos heurísticas desarrolladas dentro del contexto de un problema complejo que implica una batería limitada con una estación de recarga en una localización específica del mapa.

Intentar observar el comportamiento de los algoritmos informados y su eficiencia ha sido la principal prioridad. Para diferentes localizaciones de las estación/es de carga y niveles de batería no hemos realizado pruebas detalladas porque nuestra percepción ha sido parecida en las ejecuciones que hemos realizado de forma aislada. Esta comparación nos parecía la más interesante.

Hay que tener en cuenta que el espectro de pruebas se encuentra limitado por la potencia de la máquina donde se realiza el trabajo. En contextos donde las dimensiones crecen más de las observadas en la tabla o se añaden objetivos/estaciones el problema deja de tener solución al agotar los recursos disponibles del computador.

Tabla 2. (Escenarios divididos por las líneas de mayor grosor)

Primer escenario con un mapa con semilla = 222: Comparación del coste en tiempo (iterations), longitud, coste de la solución y coste en memoria de la solución para los distintos escenarios y algoritmos. Utilizaremos la función heurística h.

Segundo escenario con un mapa con semilla = 222: Comparación del coste en tiempo (iterations), longitud, coste de la solución y coste en memoria de la solución para los distintos escenarios y algoritmos. Utilizaremos la función heurística h1.

Mapa	***Número de Objetivos***	***Amplitud***	***Profundidad***	***Greedy***	***A****
Pequeño (8x3 casillas)	4 (1 batería)	Iteraciones: 21310 Longitud: 19 Coste: 83 Nodos lista abierta: 4342	Iteraciones: 1195 Longitud: 119 Coste: 483 Nodos lista abierta: 149	Iteraciones: 268 Longitud: 33 Coste: 137 Nodos lista abierta: 190	Iteraciones: 10511 Longitud: 21 Coste: 82 Nodos lista abierta: 3776
Mediano (9x4 casillas)	4 (1 batería)	Iteraciones: 38246 Longitud: 21 Coste: 93 Nodos lista abierta: 4660	Iteraciones: 3193 Longitud: 81 Coste: 392 Nodos lista abierta: 94	Iteraciones: 436 Longitud: 95 Coste: 189 Nodos lista abierta: 720	Iteraciones: 26035 Longitud: 23 Coste: 88 Nodos lista abierta: 5434
Grande (7x6 casillas)	4 (1 batería)	Iteraciones: 45636 Longitud: 21 Coste: 94 Nodos lista abierta: 5380	Iteraciones: 2938 Longitud: 101 Coste: 416 Nodos lista abierta: 135	Iteraciones: 1075 Longitud: 61 Coste: 189 Nodos lista abierta: 1391	Iteraciones: 36582 Longitud: 23 Coste: 94 Nodos lista abierta: 7168
Pequeño (8x3 casillas)	4 (1 batería)	Mismo resultado que en el caso anterior.	Mismo resultado que en el caso anterior.	Iteraciones: 313 Longitud: 33 Coste: 124 Nodos lista abierta: 334	Iteraciones: 7714 Longitud: 21 Coste: 82 Nodos lista abierta: 3422
Mediano (9x4 casillas)	4 (1 batería)	Mismo resultado que en el caso anterior.	Mismo resultado que en el caso anterior.	Iteraciones: 652 Longitud: 83 Coste: 154 Nodos lista abierta: 706	Iteraciones: 20531 Longitud: 23 Coste: 88 Nodos lista abierta: 5147
Grande (7x6 casillas)	4 (1 batería)	Mismo resultado que en el caso anterior.	Mismo resultado que en el caso anterior.	Iteraciones: 654 Longitud: 85 Coste: 163 Nodos lista abierta: 1108	Iteraciones: 32333 Longitud: 21 Coste: 93 Nodos lista abierta: 7493

Conclusiones

En este apartado final de la memoria analizaremos los resultados de la etapa de experimentación anteriormente descrita:

- Análisis de los datos para la primera tabla:

Hemos decidido variar el número de casillas y objetivos en este tipo de pruebas así como su geometría (más o menos cuadrados) para percatarnos de peculiaridades de los diferentes algoritmos a la hora de encontrar soluciones. Más adelante hablaremos de ellas.

Como consideraciones básicas acerca de los algoritmos de búsqueda tenemos que **profundidad** es el algoritmo que más tarda en encontrar una solución. Además el tiempo en encontrar una solución escala enormemente rápido cuando la complejidad del problema aumenta. El algoritmo que menos tiempo tarda es **greedy** por ser informado y basarse en **profundidad**.

Relativo a la longitud y al coste de la solución tenemos que los algoritmos de **amplitud** y **A*** suelen encontrar las mejores y más optimizadas soluciones en cuanto a estos términos. En ocasiones, los algoritmos de **profundidad** y **greedy** encuentran también una solución extremadamente eficiente en cuestiones de longitud, coste y uso de recursos, pero hemos percibido que su rendimiento es irregular y depende en gran medida de la forma que tenga el mapa y de la distribución de sus objetivos. La linealidad en la ubicación de los objetivos así como la geometría más irregular del posible mapa de trabajo puede hacer que sean mucho mejores para según que situaciones o bastante mediocres hasta el punto de no encontrar la solución a problemas no tan complejos en caso contrario.

En cuestión de consumo de memoria **amplitud** y **A*** son los que más consumen recursos en varios órdenes de magnitud superiores en comparación con **profundidad** y **greedy**. Esto tiene sentido por la forma de actuar que tienen unos y otros. La capacidad de escalar de ambos grupos de algoritmos no es la misma y entra dentro de la inteligencia de las personas que analicen el problema que tengan entre manos cual es la mejor aplicación posible. Los primeros siempre encontrarán una solución y generalmente será eficiente en cualquier situación, pero el consumo de recursos físicos será muy elevado e insostenible en según que casos. Sin embargo, con un buen análisis del problema, los segundos pueden resultar increíblemente rápidos e igual de válidos. Podríamos decir que su uso sería más enfocado y específico. **Greedy** nos ha parecido muy interesante porque creemos que, acompañado de una heurística potente, es bastante competente en muy diversas situaciones.

- Análisis de los datos para la segunda tabla:

En este caso de experimentación, el análisis se ha vuelto mucho más complejo. Los tiempos de ejecución en busca de una solución han aumentado mucho al intervenir una nueva variable como es la batería y un nuevo agente como es la estación de repostaje.

A la hora de probar si el funcionamiento es el correcto implementando batería, nuestros resultados han sido satisfactorios dentro de la lógica, claro está. Por ejemplo, no tiene mucho sentido establecer una batería muy baja inicialmente y un punto de recarga muy alejado, puesto que el problema no tendría solución. En casos de uso intermedios con una batería de carga completa, pensamos que ha funcionado todo correctamente.

Comparando las heurísticas, hemos caído en la cuenta de que h1 consume menos recursos en la máquina de ejecución a cambio de un tiempo de solución ligeramente mayor. Esto pensamos que se debe a la mayor complejidad en los cálculos presente en dicha heurística. Sin embargo, la

optimización en longitud y coste suele ser mejor con esta heurística, lo que nos hace pensar que está mejor definida en cuanto primar el bajo coste de la solución encontrada. En algunos casos la longitud de la solución ha sido ligeramente mayor pero su coste menor.

Para la heurística h, hemos percibido que es más equilibrada en cuanto a longitud y coste de la solución, puesto que solo ha sido capaz de ser superada su solución en términos de coste por los algoritmos informados que utilizaban h1. En este caso, parece que las soluciones tienden a obtener mejores valores en términos de longitud a cambio de elevar ligeramente el coste de la solución.

En resumidas cuentas, parece cabal pensar que depende de las consideraciones acerca de que tipo de solución es mejor, una heurística trabaja mejor que la otra. Si nuestra prioridad es minimizar la longitud, aprendemos que en IA no siempre es la mejor idea intentar realizar cálculos muy precisos sobre aquello que no conocemos y que las aproximaciones pueden ser nuestras aliadas a la hora de encontrar soluciones aceptables, como es el caso de h. En contrapartida, si nuestra prioridad es minimizar el coste por encima de la longitud, h1 ofrece mejores resultados al realizar un cálculo más preciso de las distancias basado en un teorema matemático conocido como es el de Pitágoras.

Como conclusión, pensamos que lo mejor es no perder la perspectiva a la hora de resolver problemas relacionados con la IA. Comprender la naturaleza del problema que tenemos delante tiene mucho más sentido que intentar calcular una solución con las variables numéricas que presenta. Los problemas cambian y evolucionan, nunca son estáticos. De este razonamiento depende si nuestras soluciones serán momentáneas o, por el contrario, perdurarán a lo largo del tiempo.