Razonamiento Automatizado 2020-2
Proyecto 1

Alma Rocío Sánchez Salgado.

15 de abril de 2020

Planteamiento del problema

Sabemos que uno de los problemas más comunes es encontrar una ruta de un punto A a un punto B. Este tipo de problemas tienen diversas variantes, representado con gráficas o bien, representado como una cuadrícula de $m * n$ donde el punto A con coordenadas (i, j) con $0 < i < m$ y $0 < j < n$ tiene como objetivo llegar a la casilla B.

El problema a resolver es verificar si en un número de pasos k un agente puede llegar a la casilla con oro. Para esto tomaremos en cuenta lo siguiente:

1. El agente siempre inicia desde la casilla $(0, 0)$
2. En casillas aleatorias hay obstáculos y el agente debe evitarlos
3. En una sola casilla estará el oro
4. El agente solo puede visitar la casilla una vez
5. El agente puede moverse solo una casilla a la derecha, arriba o abajo.

Resolución del problema con satisfacibilidad

El problema de satisfacibilidad verifica si hay un modelo para la resolución de este o no. Es sabido que existe una amplia variedad de problemas que pueden ser codificados como instancia SAT, es decir, si un problema lo codificamos como instancia SAT podemos hacer uso de herramientas SAT-solvers para ver si son o no satisfacibles.

Ahora bien, el agente solo puede observar qué es lo que hay en sus casillas adyacentes, no sabe más y queremos ver si existe una ruta de longitud k que no pase por los obstáculos para llegar a la casilla que tiene oro. Por lo que es conveniente tener una serie de cláusulas en las que solamente intervengan las posibles casillas a las que se pueda mover teniendo en cuenta el objetivo ylos obstáculos para después verificar si es satisfacible o no. Entonces, en el paso k descartaremos en la fórmula a verificar los pasos anteriores,esto para hacer más comprensibles las fórmulas que se generen.

Nos podemos basar en lo se plantea en el libro *Artificial Intelligence: A modern approach* para codificar el problema de planeación en una instancia SAT

Algorithm 1: Algoritmo $SAT - PLAN$

input : init, transition, goal, $number - steps$
output: UNSAT o modelo si es SAT

foreach *step in steps* **do**
  $formula \longleftarrow$ **TRANSLATE-TO-SAT(init, transition, goal, step)**
  $model \longleftarrow$ **SAT-SOLVER(formula)**
  if *model is not null* **then**
    return *EXTRACT-SOLUTION(model)*
  end
end
return *UNSAT*

Para adaptarlo a nuestro problema consideraremos lo siguiente:

- Como el agente tiene tres posibles acciones (moverse una casilla hacia la derecha, hacia arriba o hacia abajo), entonces tenemos tomar en cuenta todas las acciones válidas de acuerdo a su posición en la cuadrícula, i.e, si se encuentra en la casilla más alta, el agente no podrá moverse hacia arriba.

- Tenemos que abarcar todas las posibles rutas del agente, para esto, por ejemplo en una cuadrícula de $3 * 3$, el agente en el paso 1 puede moverse hacia las casillas $(1, 0), (0, 1)$. Tenemos dos posibles rutas entonces para el siguiente paso del agente:

  1. El agente se mueve a la casilla $(0, 1)$: las siguientes acciones válidas son moverse hacia la casilla $(0, 2), (1, 1)$. Es importante mencionar que solo puede estar en un solo lugar
  2. El agente se mueve a la casilla $(1, 0)$: se puede mover hacia una casilla de: $(1, 1), (2, 0)$

  Por lo que las posibles ubicaciones del agente en el segundo paso son : $[(0, 2), (1, 1), (2, 0)]$. De esta manera, lo que hay que hay que verificar es si alguna de esas casillas tiene el oro, pero sin olvidar que hay obstáculos. Para esto vamos a formalizar el problema de la siguiente manera:

  - `Variables:`
    * La localización del agente en el tiempo t: $A - ijt$
    * La localización del obstáculo en el tiempo t: $P - ijt$
    * La localización del oro en el tiempo t : $G - ijt$
  - `Estado inicial:` Será la localización del agente en el paso en el que esté (tiempo), recordemos que el agente no sabe nada más que información de sus casillas adyacentes y su actual.
  - `Acciones: :`
    * $mover - arriba(i, j, time) = A - i - 1, j, time$
    * $mover_abajo(i, j, time) = A - i + 1, j, time$
    * $mover_derecha(i, j, time) = A - i, j + 1, time$

- Para codificarlo podemos tener un arreglo de fórmulas que se tienen que cumplir así que vamos añadiendo las siguientes fórmulas a este:

  * Restricciones: Sabemos que exactamente hay una casilla con oro, entonces a lo más va a existir una casilla al que el agente pueda moverse que tenga oro, obtenemos las literales correspondientes y aplicamos la función *exactly-one* que es el equivalente a la función $MaxUno$
  * Siempre el agente empieza en $A - 000$ en el tiempo 0, esa casilla nunca es obstáculo y tampoco tiene oro, entonces puede verse como $AND(A - 000, Not(P - 000), Not(G - 000))$
  * Para moverse: codificamos las posibles casillas a las que se puede mover y sabemos que el agente puede estar en una sola casilla al mismo tiempo, por lo que también le aplicamos la función *exactly-one*. Sabemos que el agente se puede mover si no hay pozo en esa casilla, así que se tiene que cumplir que $AND(A - ijt, Not(P - ijt))$. Finalmente, que para que se cumpla el objetivo el agente debe estar en la misma posición que el oro pero solamente se puede cumplir uno, así que se codifica como $AND(A - ijt, G - ijt)$ y se pasa por la función *exactly-one-goal*
  * Finalmente se agrega el hecho de que el oro esté o no en una casilla (por lo que ve el agente) y lo mismo con los obstáculos

- Como ya tenemos la fórmula, revisamos si es satisfacible, si lo es, podemos observar el valor de cada variable

Resultados

El programa sí verifica correctamente si en k pasos el agente puede llegar al objetivo, el modelo que arroja es el estado de las casillas adyacentes y deduce en cuál casilla está el agente (siempre se pone or en la fórmula), sin embargo, no puede dar el plan por la manera en cómo codificamos el problema. Para que esto último pueda suceder, tenemos que considerar a las casillas ya visitadas para toda la fórmula. Algo bueno de esto es que de manera muy sencilla podemos implementar la verificación para n robots que quieren llegar al mismo objetivo, solo es cuestión de iterar el proceso para cada uno y añadir la restricción de que dos robots no pueden estar al mismo tiempo en la misma casilla.

Para correr el proyecto solo basta con ejecutar *solve.py* en el directorio donde está el archivo, se pueden modificar los parámetros del número de pasos, así como el ancho y alto de la cuadrícula o del mundo.