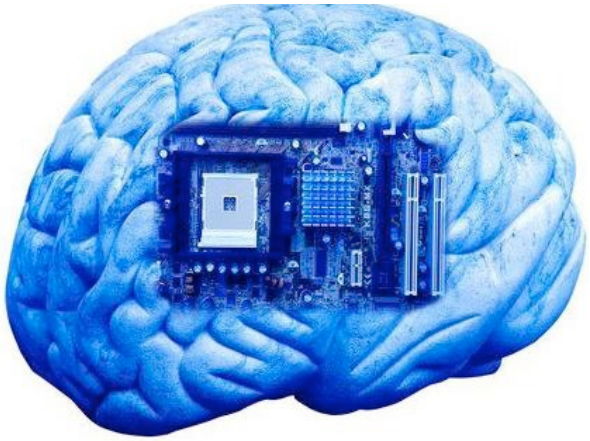
Práctica 1: Resolución de problemas mediante búsqueda



Grado Ingeniería Informática. Inteligencia Artificial 2020/2021

Escuela Universitaria
Politécnica - Teruel
Universidad Zaragoza

Tabla de Contenidos

Esta práctica se corresponde con el Tema 2 de la asignatura, que a su vez está basado en el Capítulo 3 de la bibliografía básica ("*Artificial Intelligence: A Modern Approach, Chapter 3: Solving Problems By Searching*").

Objetivos de la práctica

La primera práctica de la asignatura se centrará en la resolución básica de problemas mediante búsqueda por expansión de estados. Se estudiarán diferentes estrategias tanto informadas como no informadas y se estudiará su eficiencia a la hora de resolver problemas en entornos deterministas, completamente observables y estáticos. Gran parte de los juegos para un solo jugador (sudoku, puzles, etc.) podrían caer en esta categoría, pero también otros sistemas en los que se puedan definir de forma inequívoca los diferentes estados del sistema y las transiciones entre los mismos mediante una serie finita de acciones.

Búsqueda no informada e informada

Los problemas de búsqueda siempre comienzan con un estado inicial del sistema, y se centran en encontrar un estado final que satisface ciertas condiciones. Por tanto, debemos definir para cada problema los siguientes parámetros:

- El estado inicial del sistema.
- Las posibles acciones que pueden producirse desde cada uno de los posibles estados del sistema.
- El resultado de dichas acciones, es decir, a qué estado nos conduciría una acción dado un estado dado.
- Una función de coste, que asigna un coste a cada serie de acciones. Normalmente se representa como $g(n)$.
- Un test para determinar si un estado es el final (también llamado meta u objetivo).

El esquema general de un algoritmo de búsqueda aparece en la Figura 3.7 de la bibliografía, definida como *TREE-SEARCH* y *GRAPH-SEARCH*. La única diferencia entre ellos es que *GRAPH-SEARCH* incorpora una lista de estados visitados (explorados) para evitar repeticiones, por lo que es más eficiente especialmente en entornos con multitud de estados. Por ello, usaremos este algoritmo como base para la práctica:

```
function GRAPH-SEARCH(problem) returns a solution, or failure
        initialize the frontier using the initial state of problem
        initialize the explored set to be empty
        loop do
                if the frontier is empty then return failure
                choose a leaf node and remove it from the frontier
                if the node contains a goal state then return the corresponding solution
                add the node to the explored set
                expand the chosen node, adding the resulting nodes to the frontier
                   only if not in the frontier or explored set
```

Figura 1: Descripción informal del algoritmo general de búsqueda en grafos

Existen diferentes variantes de este algoritmo, todas ellas basadas en diferencias a la hora de elegir un nodo a expandir de la frontera de estados visitados. Podemos hacer una primera división entre algoritmos de búsqueda no informada, o informada:

- **Búsqueda no informada (ciega):** las estrategias no disponen de información adicional sobre estados más allá de la proporcionada en la definición del problema, por lo que sólo pueden expandir estados y distinguir entre estados objetivos y no-objetivos. Dentro de esta categoría se distinguen:
    - **Búsqueda primero en anchura (*breadth-first*):** se expanden primero los estados de un nivel antes de expandir los del estado siguiente.
    - **Búsqueda primero en profundidad (*depth-first*):** se expanden primero los estados de mayor nivel (los más profundos).
    - **Búsqueda en coste uniforme (*uniform cost*):** se expanden primero los estados con menor coste acumulado de acciones para llegar a él.
- **Búsqueda informada (heurística):** se utiliza información propia del problema más allá de la proporcionada en la definición. Se utiliza una función heurística, normalmente denominada $h(n)$, que estima el coste del camino más corto desde el estado actual hasta el objetivo. Para que la heurística sea admisible, nunca puede indicar un coste mayor que el real para algún estado. En esta categoría se incluirían:
    - **Búsqueda voraz, avariciosa o primero el mejor (*greedy or best-first*):** se expanden primero los estados con el menor valor de la función heurística, h.
    - **Búsqueda A*:** se expanden primero los estados con el menor valor de la suma del coste para llegar al estado más la estimación de alcanzar el estado objetivo, es decir, $f = g + h$.

Problema a resolver

Para aplicar las estrategias de búsqueda, resolveremos el problema de los misioneros y caníbales. En este problema tenemos 3 misioneros y 3 caníbales que desean cruzar un río empleando una barca en la que caben 2 personas. Sin embargo, los misioneros deben tener cuidado con cómo cruzan el río, pues si en alguna de las orillas hay un número mayor de caníbales que misioneros, se convertirán en la cena.

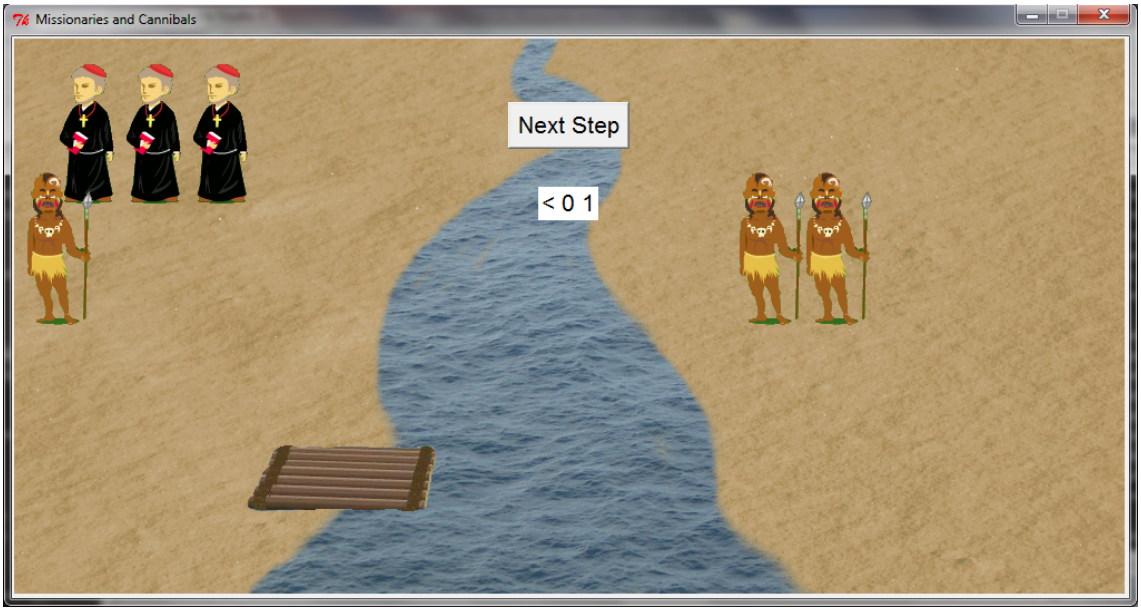


Figura 2: Ejemplo de estado en el problema de misioneros y caníbales

Se partirá del estado en el que las 6 personas y la barca están en el lado izquierdo del río, y el objetivo consiste en conseguir que las 6 personas y la barca finalicen en el lado derecho del río. Las acciones las representaremos mediante un string o un tupla en Python con la siguiente estructura: `accion = ('>', M, C)`, o `accion = '>MC'`, donde :

- `accion[0]` indica la dirección en que se mueve la barca: `'>'` corresponde a mover la barca de la orilla izquierda a la derecha, y `'<'` el movimiento de derecha a izquierda.
- `accion[1]` corresponde al número de misioneros que pasan en la barca.
- `accion[2]` corresponde al número de caníbales que pasan en la barca.

Por supuesto, algunas de las posibles acciones que podrían darse en el problema llevarían a un estado incorrecto y no podrían darse, como intentar pasar más personas de las disponibles en una orilla o dar lugar a una situación con menos misioneros que caníbales en alguna orilla.

El coste de una solución (función *g*) se considerará igual al número de viajes de la barca que han sido necesarios para alcanzar el objetivo.

Ficheros de la práctica

La práctica está compuesta por los siguientes ficheros:

- `datastructures.py`: contiene estructuras de datos útiles en Python para la realización de los algoritmos (`Stack`, `Queue`, `PriorityQueue`).
- `search.py`: contiene la estructura básica del proceso de búsqueda no informada e informada. Sin embargo, algunas de las funciones no tiene el código necesario para que el algoritmo funcione correctamente. Por tanto, cada alumno debe rellenar el código de las siguientes funciones:
  - `uninformed_search`
  - `breadth_first`
  - `depth_first`
  - `uniform_cost`

    - `informed_search`
    - `greedy`
    - `a_star`
    - `h1` u otras funciones heurísticas.
- `state.py`: contiene la clase que representa cada estado del problema. Cada alumno debe rellenar el código de las siguientes funciones:
    - `succ`
    - `next_states`
- `missionariesworld.py`: proporciona una interfaz gráfica para poder ver la solución encontrada con alguno de los algoritmos de búsqueda.
- `missionariesproblem.py`: contiene el programa principal de la práctica, que hace llamadas a los diferentes algoritmos de búsqueda.

Entrega de la práctica

Deberán completare las siguientes tareas para considerar la práctica entregada:

1. Completar el código Python necesario para que los algoritmos de búsqueda no informada e informada funcionen correctamente, siguiendo el algoritmo *GRAPH-SEARCH*. Además, deberán proponerse, al menos 2 funciones heurísticas diferentes y admisibles para ser utilizadas en los algoritmos de búsqueda informada. La eficiencia de las funciones heurísticas formará parte de la evaluación de la práctica.
2. Estudiar la optimalidad y eficiencia de los algoritmos en el problema básico con 3 misioneros y 3 caníbales, comparando la longitud de las soluciones encontradas y el número de nodos expandidos y generados por cada algoritmo. Discutir los resultados.
3. (Opcional) Ampliar el problema para que el número de misioneros (M) y caníbales (C) sea variable, así como la capacidad de la barca (B). No obstante, debe cumplirse en la definición del problema que M >= C, y C > B. Además, en la barca nunca podrá haber más caníbales que misioneros, al igual que en las orillas.
4. Estudiar el comportamiento de los algoritmos cuando se aumenta el número de misioneros y caníbales, y la capacidad de la barca. Probar al menos 2 configuraciones, con una de ellas con un número de misioneros superior a 50.

Todas estas tareas se llevarán a cabo en parejas o de manera individual.

La práctica se entregará a través del Anillo Digital Docente (ADD), para lo cual se deberá realizar una memoria indicando cómo se ha realizado la implementación y los resultados obtenidos por los diferentes algoritmos, y se deberá entregar un fichero comprimido con el código de la práctica y con la memoria realizada en formato PDF. La fecha límite de entrega será el **16/10/2020**.

Recursos adicionales

- Juego de misioneros y caníbales (Flash):
    - http://www.learn4good.com/games/puzzle/boat.htm
- Reglas del juego de misioneros y caníbales:
    - http://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem