Primera Tarea Académica – Aplicaciones de Ciencias de la Computación (INF265-0781)

Búsqueda en el juego de Pacman

Introducción

La presente tarea académica se elaborará sobre el proyecto del juego PACMAN desarrollado en la Universidad UC Berkeley (http://ai.berkeley.edu/search.html). Sera necesario entrar a ese sitio web y bajar el archivo search.zip que contiene todos los archivos Python necesarios para desarrollar el trabajo. La descripción de cada archivo se encuentra en el mismo sitio web. Para este trabajo sólo se necesitará editar los archivos `searchAgents.py` y `search.py` donde residen las implementaciones de agentes, problemas y algoritmos de búsqueda. Las implementaciones del ambiente del Pacman, estados, direcciones, grid y estructuras de datos útiles se encuentran en los otros archivos. Después de bajar y descompactar `search.zip` es posible jugar Pacman digitando: `python pacman.py`
Pacman tiene varias opciones (para verlas tipear: `python pacman.py -h`). Por ejemplo, la opción –l especifica el layout del pacman; –p especifica el agente de búsqueda, -a especifica el algoritmo de búsqueda. Un agente básico ya implementado en `searchAgents.py` es el agente GoWestAgent, que siempre va para el oeste (es un agente reactivo trivial). Para probarlo, digitar: `python pacman.py -l testMaze -p GoWestAgent`

Obviamente este agente se queda atascado cuando ya no puede ir más a la derecha. Para salir del juego tipear CTRL-c en el terminal.

Desafíos:

1) Definir problema de búsqueda de rutas óptimas para que el Pacman visite sus esquinas

Se debe habilitar un agente de búsqueda para que encuentre una ruta mínima que comience en la posición inicial del Pacman, visite las cuatro esquinas del layout y regrese a la posición inicial. Para ello deberá definir un espacio de estado que codifique toda la información necesaria que permita detectar si se ha alcanzado el objetivo (las cuatro esquinas han sido visitadas y ha regresado a su posición inicial). Lo ideal sería un espacio de estados que facilite la búsqueda de caminos óptimos. Se recomienda que los estados no codifiquen información irrelevante para el problema (posición de fantasmas, comida extra, etc.). No se recomienda usar `GameState` como estado (sería muy lento).

La definición de este problema de búsqueda debe ser implementada en la clase `CornersProblem`, que está en el archivo `searchAgents.py`. Se debe implementar la función que genera estados sucesores (`getSuccessors`), la función de test de objetivo (`isGoalState`) y la función que retorna el estado inicial en el espacio de estados escogido (`getStartState`).

2) Implementar y comparar desempeños de algoritmos de búsqueda

Una vez implementado `CornersProblem` se pide implementar los siguientes métodos de búsqueda en `search.py`:

- Depth First Search – dfs (función `depthFirstSearch`)
- Breadth First Search – bfs (función `breadthFirstSearch`)

- Iterative deepening search – ids (función `iDeepeningSearch`)
- Bidirectional search – bs (función `bidirectionalSearch`)
- A* Search – astar (función `aStarSearch`)

Para el caso de `aStarSearch`, se debe suministrar como argumento una función heurística. La función heurística trivial `nullHeuristic`, que devuelve siempre cero (implementada en search.py), podría ser pasada a `aStarSearch`, implicando una búsqueda de costo uniforme. Se pide implementar una heurística admisible y consistente (en `cornersHeuristic`) que ayude a A* a encontrar una solución a `CornersProblem` con menos nodos explorados que la heurística nullHeuristic. Probar las implementaciones en los layouts: `mediumCorners` y `bigCorners`. Por ejemplo, los comandos siguientes prueban bfs y A* (con `cornersHeuristic`) en `mediumCorners`:

```
> python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
> python pacman.py -l mediumCorners -p SearchAgent -a fn=
aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic -z 0.5
```

Llenar la tabla abajo, analizar los resultados y explicar cuál sería el método (o métodos) más adecuados y menos adecuados para el presente problema y por qué.

		DFS	BFS	IDS	BS	A* (cornersHeuristic)
mediumCorners	# nodos visitados					
	# nodos en memoria					
	Costo de la Solución					
bigCorners	# nodos visitados					
	# nodos en memoria					
	Costo de la Solución					

3) Implementar un agente de búsqueda codiciosa

En este desafío se pide implementar un agente de búsqueda que construya codiciosamente la solución al problema `CornersProblem`. Esto significa que, empezando en la posición inicial del Pacman se encuentre la ruta a la esquina más cercana, luego desde allí, a la siguiente esquina más cercana y así sucesivamente hasta regresar a la posición inicial. Un agente que sigue dicha idea se encuentra implementado en `search.py` en la clase `ClosestDotSearchAgent`, el cual busca codiciosamente todas las comidas del layout (no regresa a la posición inicial). El método `findPathClosestDot` en dicho agente es usado para trazar la ruta de cada trecho (no está implementado).
Basado en `ClosestDotSearchAgent` implemente el agente solicitado (`CornersGreedySearchAgent`). Escoja algún método de búsqueda apropiado para buscar el camino de cada trecho. Pruebe su implementación en los layouts: `mediumCorners` y `bigCorners` y compare los resultados con los obtenidos en el Desafío 2. Responda:

- Vale la pena hacer dicha búsqueda codiciosa en los layouts testados?
- La solución entregada será siempre la de menor costo?

Entregables:

1) Scripts con todo el código Python que permita reproducir los resultados
2) Informe conteniendo:
   I. Descripción de la representación de estados escogida en el Desafío 1
   II. Descripción de la heurística `cornersHeuristic` con prueba teórica o computacional de su adimisibilidad y consistencia (Desafío 2)
   III. Tabla comparativa del desempeño de los algoritmos junto con su análisis (Desafío 2)
   IV. Análisis de los resultados del Desafío 3 en relación al Desafío 2, junto con las respuestas a las preguntas levantadas.
   V. Conclusiones.

Indicaciones Generales:

- Fecha de entrega: **9 de mayo de 2018 (23:59 hs)**
- Grupos de hasta **3 personas**
- Se evaluará la creatividad, correctitud de las implementaciones, el análisis de los resultados y la redacción del informe
- Se contrastará las implementaciones entre los grupos. La nota puede ser afectada negativamente en los grupos que tengan implementaciones idénticas o muy similares.