Métodos de búsqueda (Búsqueda informada)

Ahora que nuestros compañeros han cubierto la búsqueda no informada, que busca a través de todo el espacio de búsqueda sin saber a qué distancia está del objetivo, podemos pasar a la búsqueda informada, que busca a través del espacio con información adicional, como la distancia del objetivo, el coste de moverse de un estado a otro, etc.

Para ello, utilizamos una función heurística, ya que ayuda a la búsqueda utilizando esta información adicional sobre el espacio de búsqueda. Toma el estado actual como entrada y determina lo lejos que está del objetivo como salida. Debido a esto, un algoritmo de búsqueda que utiliza heurísticas no recorre todo el espacio de búsqueda, encontrando una solución en un tiempo decente. Sin embargo, por esta razón, no siempre garantiza una solución óptima, sólo una buena, lo que hace que se utilice mejor en espacios de búsqueda muy grandes, en contraposición a la búsqueda no informada, que se utiliza mejor en espacios de búsqueda pequeños, ya que el tiempo no es una restricción tan grande.

Como ejemplos, exploraremos los algoritmos de escalada y A*.

Algoritmo hill-climbing

Este algoritmo tiene muchas variantes. Para este tutorial, utilizaremos el algoritmo de escalada Steepest Ascent. Evalúa cada estado vecino y elige el que está más cerca del objetivo, luego se mueve hacia él e itera.

El algoritmo es el siguiente:

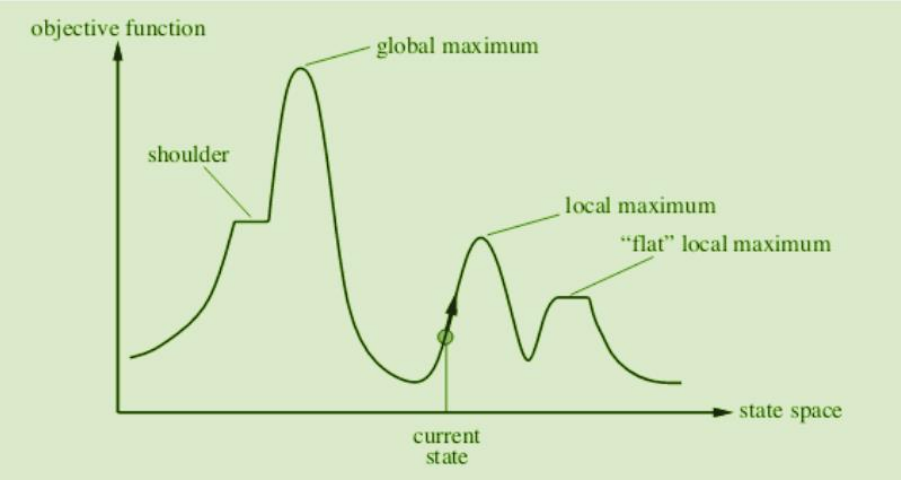
1. Evaluar el estado inicial. Si es un estado objetivo, detener el algoritmo y devolver los resultados.
2. Si no lo es, repetir los siguientes pasos hasta que se encuentre una solución o no haya cambios en los estados.
    a. Evaluar los estados vecinos que no han sido visitados.
    b. Encuentre el estado más cercano al objetivo.
    c. Si el estado vecino está más cerca que el estado actual, muévase a él. Si no, detenga el algoritmo y devuelva los resultados.

Hill-climbing en prolog:

```
% Define the graph as a list of edges (neighbors)

edge(a, b, 1).

edge(a, c, 2).

edge(b, d, 2).
```

```
edge(b, e, 3).
edge(c, d, 4).
edge(c, f, 1).
edge(d, g, 1).
edge(e, g, 5).
edge(f, g, 3).

% Define the steepest ascent hill climbing algorithm
hill_climbing(Start, Goal, Path, Cost) :-
    hill_climbing(Start, Goal, [Start], Path, Cost).
hill_climbing(Goal, Goal, Path, Path, 0).
hill_climbing(Current, Goal, Visited, Path, Cost) :-
    findall(C-Next, (edge(Current, Next, C), not(member(Next, Visited))),
Options),
    sort(Options, SortedOptions),
    reverse(SortedOptions, DescendingOptions),
    member(C-Next, DescendingOptions),
    hill_climbing(Next, Goal, [Next|Visited], Path, Cost2),
    Cost is Cost2 + C.

% Find the path to the goal with the steepest ascent hill climbing algorithm
find_path(Start, Goal) :-
    hill_climbing(Start, Goal, Path, Cost),
    write('Path: '), write(Path),
    write(', Cost: '), write(Cost).
```

Un problema de este tipo de algoritmos es que tienden a buscar óptimos locales en lugar de globales. Esto no ocurre en funciones cóncavas, pero como muchas funciones no son cóncavas, merece la pena tenerlo en cuenta. Un diagrama en GeeksforGeeks ayuda a ilustrar esto:



Algoritmo A*

Este método es una de las técnicas más populares para la búsqueda de rutas y el recorrido de grafos. Se considera particularmente inteligente, sin embargo, su complejidad espacial es un problema importante, ya que almacena todos los nodos generados en memoria.

Supongamos que se nos da una celda inicial y una celda objetivo en una cuadrícula cuadrada con numerosos obstáculos. Si es posible, queremos ir lo más rápido posible de la celda inicial a la celda objetivo. Aquí es donde brilla el algoritmo A*.

El algoritmo de búsqueda A* selecciona el nodo en cada etapa basándose en un valor, "f", que es un parámetro igual a la suma de dos factores adicionales, "g" y "h". Elige el nodo o celda con el valor "f" más bajo en cada etapa y procesa ese nodo o celda.

A continuación, definimos "g" y "h" de la forma más sencilla posible:

g = el coste de movimiento para desplazarse desde el punto de partida hasta una casilla determinada de la cuadrícula, siguiendo el camino generado para llegar hasta allí.

h = el coste de movimiento estimado para desplazarse desde esa casilla de la cuadrícula hasta el destino final. Esto suele denominarse heurística.

El algoritmo es el siguiente:

1. Inicializar la lista abierta
2. Inicializar la lista cerrada
3. Poner el nodo inicial en la lista abierta (se puede dejar su f a cero)
4. Mientras la lista abierta no esté vacía (while)
    a. Encontrar el nodo con la menor f en la lista abierta, llámelo "q"
    b. Sacar q de la lista abierta
    c. Generar 8 sucesores de q y establecer sus padres a q

d. Para cada sucesor (for)
    i. Si el sucesor es el objetivo, detener la búsqueda.
    ii. Si no, calcular g y h para el sucesor
    sucesor.g = q.g + distancia entre sucesor y q
    sucesor.h = distancia entre el objetivo y el sucesor
    iii. Si hay un nodo con la misma posición que el sucesor en la lista ABIERTA que tiene un f menor que el sucesor, omitir este sucesor
    iv. Si un nodo con la misma posición que el sucesor está en la lista CERRADA y tiene un f menor que el sucesor, omite este sucesor, de lo contrario, añade el nodo a la lista abierta.

    Fin del bucle for

e. empujar q en la lista cerrada

Fin del bucle while

A* en prolog:

```
% Define the graph as a list of edges (neighbors)
edge(a, b, 1).
edge(a, c, 2).
edge(b, d, 2).
edge(b, e, 3).
edge(c, d, 4).
edge(c, f, 1).
edge(d, g, 1).
edge(e, g, 5).
edge(f, g, 3).

% Define the heuristic function that estimates the cost to reach the goal
heuristic(g, 0).
heuristic(X, H) :- edge(X, Y, C), heuristic(Y, H2), H is H2 + C.

% Define the A* search algorithm
astar(Start, Goal, Path, Cost) :-
    astar(Start, Goal, [Start], Path, Cost).

astar(Goal, Goal, Path, Path, 0).
astar(Current, Goal, Visited, Path, Cost) :-
    edge(Current, Next, C),
    not(member(Next, Visited)),
    heuristic(Next, H),
    F is C + H,
```

```
    astar(Next, Goal, [Next|Visited], Path, Cost2),
    Cost is Cost2 + C.

% Find the shortest path from start to goal
find_path(Start, Goal) :-
    astar(Start, Goal, Path, Cost),
    write('Path: '), write(Path),
    write(', Cost: '), write(Cost).
```

Una de las principales deficiencias de este algoritmo es, como ya se ha dicho, su complejidad espacial, ya que utiliza grandes cantidades de memoria. También puede ser muy intensivo computacionalmente si el espacio de estados es grande. Además, si la función heurística no es muy precisa, puede volverse realmente lento.

Bibliografía

Adan, M. (n.d.). *What is the hill-climbing algorithm?* Educative: Interactive Courses for Software Developers. Retrieved February 5, 2023, from https://www.educative.io/answers/what-is-the-hill-climbing-algorithm

Belwariar, R. (2018, September 7). *A* Search Algorithm - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/a-search-algorithm/

Hill climbing. (2019, December 24). Wikipedia. https://en.wikipedia.org/wiki/Hill_climbing

Introduction to Hill Climbing | Artificial Intelligence. (2017, December 12). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

tutorialforbeginner.com. (n.d.). *Informed Search Algorithms | tutorialforbeginner.com*. Www.tutorialforbeginner.com. Retrieved February 1, 2023, from https://tutorialforbeginner.com/informed-search-algorithms-in-ai

What are informed search algorithms? (n.d.). Educative: Interactive Courses for Software Developers. Retrieved February 2, 2023, from https://www.educative.io/answers/what-are-informed-search-algorithms

Wikipedia Contributors. (2019, March 10). *A* search algorithm*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/A