Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Tema 1. Búsquedas en Inteligencia Artificial

Resumen y Códigos

Autor: Alberto Fernández Merchán
Asignatura: Sistemas Inteligentes

1. Formulación

La Inteligencia Artificial Clásica trata entornos con las siguientes características:

- Estático: El entorno no varía.
- Observable: Conocemos todas las variables que nos interesan.
- Discreto: Los valores tienen límite de división.
- Determinístico: El estado siguiente es consecuencia del estado actual y de la acción realizada.

2. Características

Se puede construir un grafo con la representación del problema. Las características de este grafo serán:

- Cada estado se representará mediante un nodo.
- Deberá tener un estado inicial y, al menos, un estado final.
- Las aristas representan las acciones entre los estados.
- Puede ser cíclico.

3. Espacio de Estados

3.1. Árbol de Búsqueda

El árbol de búsqueda se construye a partir del espacio de estados. En el árbol:

- Cada nodo es un estado.
- El nodo raíz es el estado inicial.
- Cada arista es la aplicación de una acción.
- Los nodos hoja son los estados finales.

Los elementos básicos para definir un problema en el espacio de estados son:

- Estado inicial: Estado con el que arranca el agente.
- Operadores: Posibles acciones que el agente puede emprender.
- Prueba de meta: Es una condición para saber si el problema ha terminado. Puede ser un estado concreto o un conjunto de propiedades ciertas en algún momento.
- Costo de Ruta: Función que asigna un costo a una acción determinada. El costo total es la suma de todos los costos de las acciones a lo largo de la ruta.

4. Solución

La solución será el **conjunto de acciones** que debe realizar un agente para llegar desde el estado inicial a cualquiera de los estados finales.

5. Ejemplos

5.1. Mundo de la aspiradora

- Estado Inicial: Cualquier estado puede ser un estado inicial.
- Operadores: Las acciones posibles serán: Izquierda, Derecha y Aspirar.
- Prueba de Meta: Revisar si las ubicaciones están limpias.
- Costo de Ruta: El costo será el número de pasos en la ruta (1 por cada acción).

5.2. 8-Puzzle

- Estado Inicial: Cualquier combinación de las 8 piezas y el espacio en blanco en el tablero.
- Operadores: Mover el hueco en blanco (Arriba, Abajo, Derecha, Izquierda).
- Prueba de Meta: Que las fichas estén ordenadas.
- Costo de Ruta: 1 por cada acción.

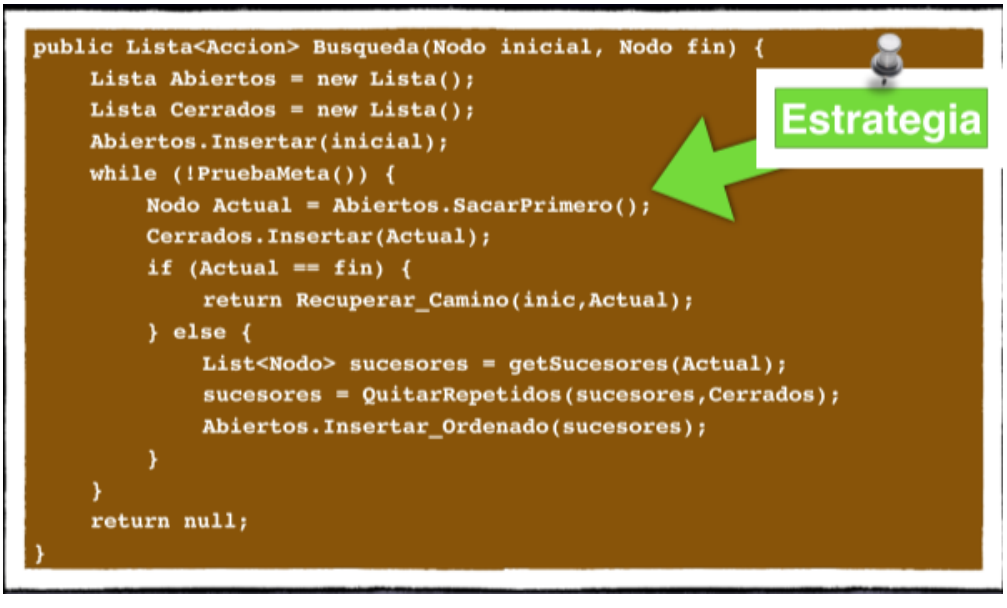
6. Búsqueda de soluciones

Se busca generar una **Secuencia de Acciones** que lleven desde el estado inicial hasta el estado de meta. Esta búsqueda se realiza en el espacio de estados. El algoritmo sería el siguiente:

- Evaluar si el estado es un estado de meta:
  - No es meta:
    - Aplicar operadores al estado actual y generar estados siguientes.
    - Elegir el estado siguiente por el que continuar.
    - Aplicar el operador.
    - Volver a comprobar si es un estado de meta.

El proceso de búsqueda es como construir un árbol de búsqueda sobrepuesto al espacio de estados.

7. Algoritmo GENERAL de Búsqueda



8. Estrategias de Búsquedas

8.1. Búsqueda No-Informada (A ciegas)

No contiene información de cuanto de cerca estamos de la solución. Solo podemos saber si hemos llegado al objetivo o no.

Son estrategias costosas, pero a veces son la única solución.

8.1.1. Primero en Anchura

Consiste en no expandir los nodos de nivel n hasta que todos los del nivel n-1 hayan sido expandidos.

La estructura para los nodos abiertos es una FIFO.

- Completitud: Siempre encuentra una solución.
- Complejidad Temporal: Exponencial respecto a la profundidad de la solución.
- Complejidad Espacial: Exponencial respecto a la profundidad de la solución.
- Optimización: La solución es óptima

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
    Lista Abiertos = new Lista();
    Lista Cerrados = new Lista();
    Abiertos.Insertar(inicial);
    while (Abiertos.Tamano()>0) {
          Nodo Actual = Abiertos.SacarPrimero();
          Cerrados.Insertar(Actual);
          if (Actual == fin) {
                return Recuperar_Camino(inic,Actual);
          } else {
                List<Nodo> sucesores = getSucesores(Actual);
                sucesores = QuitarRepetidos(sucesores,Cerrados);
                Abiertos.Insertar_AL_FINAL(sucesores);
          }
    }
    return null;
}
```

8.1.2. Primero en Profundidad

No expande los nodos del nivel n si hay algún hijo de nivel $> n$ pendiente de considerar.
La estrutura para los nodos abiertos es una LIFO.

- Completitud: Encuentra una solución si no hay ramas infinitas antes de la solución.
- Complejidad Temporal: Exponencial respecto a la profundidad del límite de exploración.
- Complejidad Espacial: Sin control de repetidos es lineal.
- Optimización: No se garantiza la solución óptima.

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
    Lista Abiertos = new Lista();
    Lista Cerrados = new Lista();
    Abiertos.Insertar(inicial);
    while (Abiertos.Tamano()>0) {
          Nodo Actual = Abiertos.SacarPrimero();
          Cerrados.Insertar(Actual);
          if (Actual == fin) {
                return Recuperar_Camino(inic,Actual);
          } else {
                List<Nodo> sucesores = getSucesores(Actual);
                sucesores = QuitarRepetidos(sucesores,Cerrados);
                Abiertos.Insertar_PRINCIPIO(sucesores);
          }
    }
    return null;
}
```

8.1.3. Profundidad Limitada

Se dejan de generar sucesores cuando llega al límite de profundidad. De esta forma se garantiza que el algoritmo pueda terminar.

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
    Lista Abiertos = new Lista();
    Lista Cerrados = new Lista();
    Abiertos.Insertar(inicial);
    while (Abiertos.Tamano()>0) {
          Nodo Actual = Abiertos.SacarPrimero();
          Cerrados.Insertar(Actual);
          if (Actual == fin) {
                return Recuperar_Camino(inic,Actual);
          } else {
                List<Nodo> sucesores = getSucesores(Actual);
                sucesores = QuitarRepetidos(sucesores,Cerrados);
                Quitar_Fuera_de_Nivel(sucesores);
                Abiertos.Insertar_PRINCIPIO(sucesores);
          }
    }
    return null;
}
```

8.1.4. Primero en Profundidad Iterativa

Realiza búsquedas en profundidad limitada sucesivas con un nivel de profundidad máximo y creciente en cada iteración.
Permite evitar los casos en el que existen ramas infinitas (Profundidad) definiendo una cota máxima de profundidad en la exploración.

- Completitud: Siempre encuentra la solución.
- Complejidad Temporal: Igual que la búsqueda en anchura. Al regenerar el árbol solo añade una constante a la función de coste.
- Complejidad Espacial: Igual que en la búsqueda en profundidad.
- Optimización: La solución es óptima.

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
    int nivel = 0;
    while (!encontrado){
          Lista Abiertos = new Lista();
          Lista Cerrados = new Lista();
          Abiertos.Insertar(inicial);
          while (Abiertos.Tamano()>0) {
                Nodo Actual = Abiertos.SacarPrimero();
                Cerrados.Insertar(Actual);
                if (Actual == fin) {
                      return Recuperar_Camino(inic,Actual);
                } else {
                      List<Nodo> sucesores = getSucesores(Actual);
                      sucesores = QuitarRepetidos(sucesores,Cerrados);
                      Quitar_Fuera_de_Nivel(sucesores);
                      Abiertos.Insertar_PRINCIPIO(sucesores);
                }
          }
          nivel = nivel + 1
    }
    return null;
}
```

8.2. Búsqueda Informada (heurística)

Existe una estimación de cómo de cerca está la solución. Tenemos que medir cuanto creemos que falta para llegar al final.
Utiliza una función heurística $h'(n)$ que mide el coste estimado más barato desde el nodo n hasta el nodo objetivo.
La función de evaluación ($f'(n)$)tiene dos componentes:

$$f'(n) = g(n) + h'(n) \tag{1}$$

Donde $g(n)$ es el coste mínimo desde el inicio hasta el nodo actual y $h'(n)$ es el coste mínimo estimado para ir desde el nodo actual a una solución.

A veces puede resultar peor que la estrategia ciega. ya que no se garantiza la completitud ni la optimalidad.

8.2.1. Primero el mejor (Voraz)

Expande el nodo más cercano al objetivo. Conduce rápidamente a una solución.

Evalúa los nodos utilizando la función heurística.

$$f'(n) = h'(n). \tag{2}$$

La estructura es una cola con prioridad. Esta es marcada por la función heurística. En cada iteración se escoge el primero de la cola.

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
    Lista Abiertos = new Lista();
    Lista Cerrados = new Lista();
    Abiertos.Insertar(inicial);
    while (Abiertos.Tamano()>0) {
          Nodo Actual = Abiertos.SacarPrimero();
          Cerrados.Insertar(Actual);
          if (Actual == fin) {
                return Recuperar_Camino(inic,Actual);
          } else {
                List<Nodo> sucesores = getSucesores(Actual);
                sucesores = QuitarRepetidos(sucesores,Cerrados);
                Abiertos.Insertar_PRINCIPIO(sucesores);
          }
    }
    return null;
}
```

8.3. Algoritmos de clase A

La función de evaluación ($f'(n)$)tiene dos componentes:

$$f'(n) = g(n) + h'(n) \tag{3}$$

f' es un valor estimado del coste total.
h' es un valor estimado del coste de alcanzar el objetivo.
g es un coste real. Representa el coste del camino más corto conocido.
Escoge el nodo con menor f' y, en caso de empate, el que tenga menor h'.

8.3.1. A*

En cada iteración se escoge el mejor camino estimado. Es completo cuando el factor de ramificación es finito y cada operador tiene un coste positivo fijo.
Dependiendo de la función heurística podrá encontrar o no la solución óptima.

```
public Lista<Accion> Busqueda(Nodo inicial,fin) {
  Lista Abiertos = new Lista();
  Lista Cerrados = new Lista();
  Abiertos.Insertar(inicial);
  while (Abiertos.Tamano()>0) {
    Nodo Actual = Abiertos.SacarPrimero();
    Cerrados.Insertar(Actual);
    if (Actual == fin) {
      return Recuperar_Camino(inic,Actual);
    } else {
      List<Nodo> sucesores = getSucesores(Actual);
      sucesores = QuitarRepetidos(sucesores,Cerrados);
      Abiertos.Insertar_Ordenado(sucesores);
    }
  }
  return null;
}
```

9. Criterios para evaluar las estrategias

- Correción y Completitud: Garantiza encontrar una solución correcta y todas las soluciones.
- Complejidad: Cuanto tiempo y memoria se necesita para encontrar la solución.
- Optimización: Si se encuentra una solución óptima.

10. Búsquedas con adversario

Objetivo: Encontrar un conjunto de movimientos accesible que de como ganador a MAX.

10.1. Algoritmo MiniMax

Genera todo el árbol de jugadas. Se etiquetas las jugadas terminales dependiendo del ganador (MIN o MAX).
Se propagan los valores de las jugadas terminales de las hojas hasta la raíz.

Desventajas Cada nueva decisión por parte del adversario implica repetir parte de la búsqueda.
Para evitar dibujar el árbol de juegos entero, se utiliza una aproximación heurística (búsqueda con profundidad limitada).

10.2. Poda α-β

El problema de minimax es que el número de estados a examinar es exponencial con el número de movimientos. La poda $\alpha - \beta$ permite eliminar partes del árbol sin influir en la decisión final.

Los dos parámetros describen los límites sobre los valores que aparecen a lo largo del camino.

- α: El valor de la mejor opción encontrada hasta el momento para MAX.
- β: El valor de la mejor opción encontrada hasta el momento para MIN.

Esta búsqueda actualiza el valor de α y de β según se recorre el árbol y termina cuando encuentra un nodo peor que el actual valor de α o β