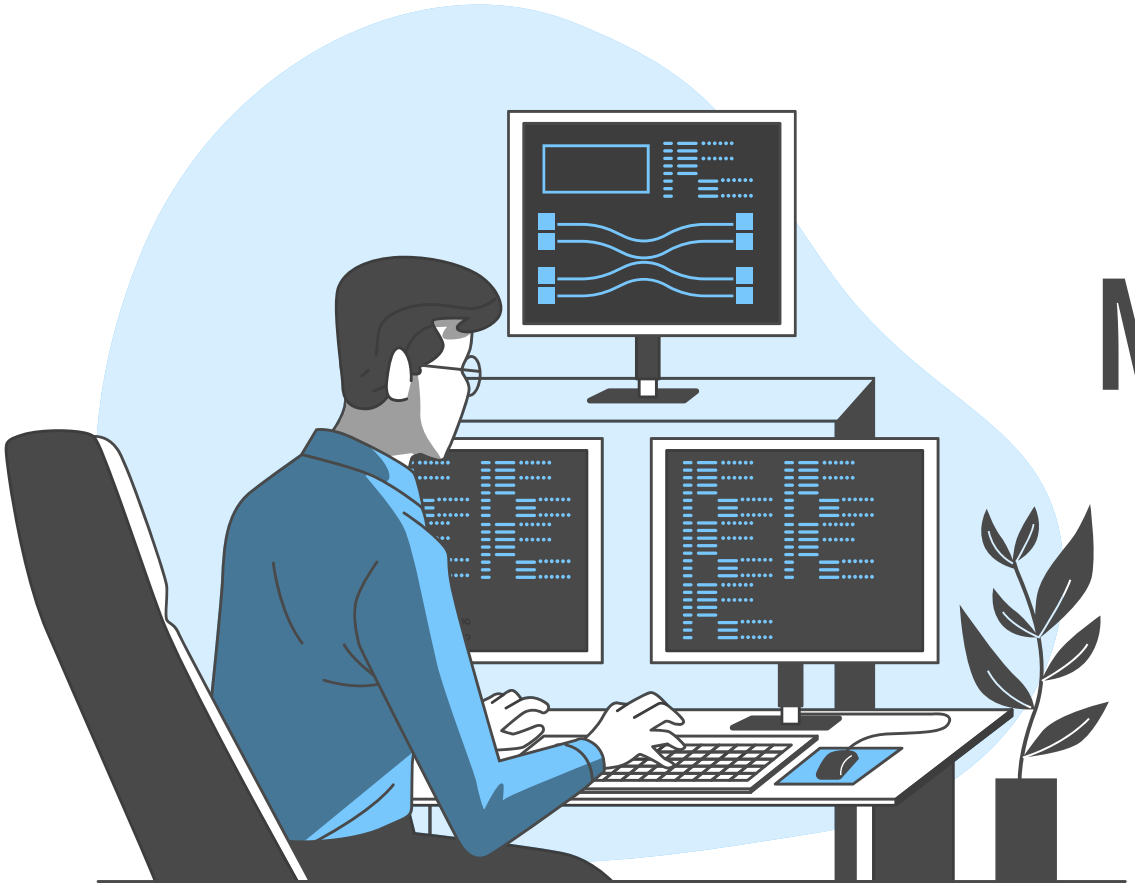


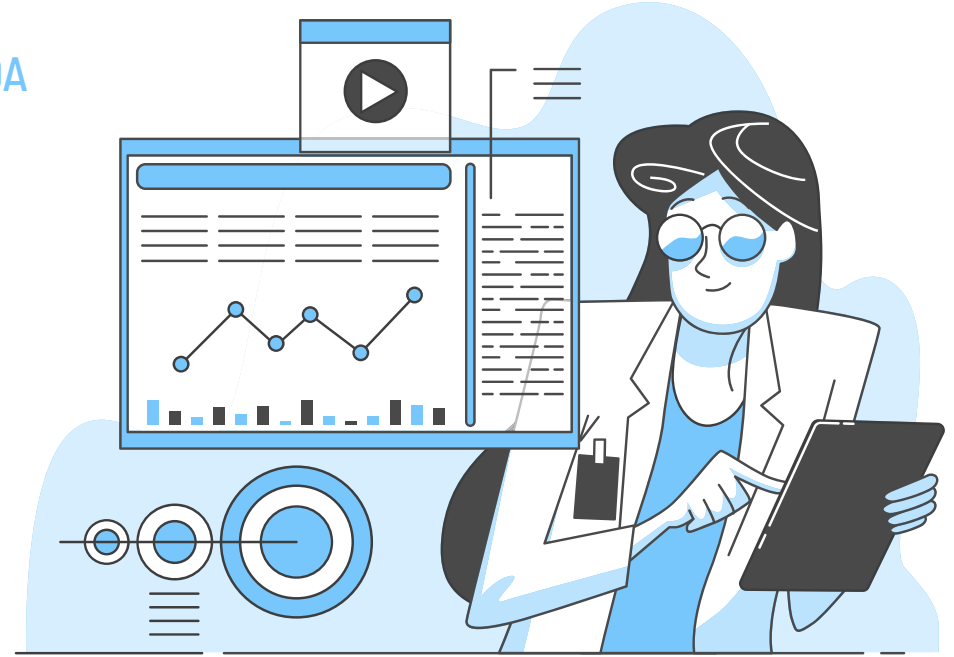
MODELOS DE BÚSQUEDA

Andres Felipe Quebrada Agudelo

Contenido

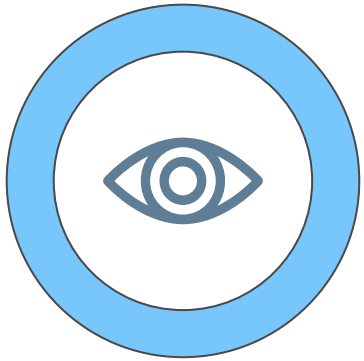
01 BÚSQUEDA NO INFORMADA O BÚSQUEDA CIEGA

02 BÚSQUEDA INFORMADA O BÚSQUEDA GUIADA



01

Búsqueda no informada o ciega



Las computadoras son ciegas, no ven, son invidentes.

No se conoce información del espacio de búsqueda.

Recursivo, no recursivo.

...

Métodos

Amplitud

Breadth First Search (BFS)

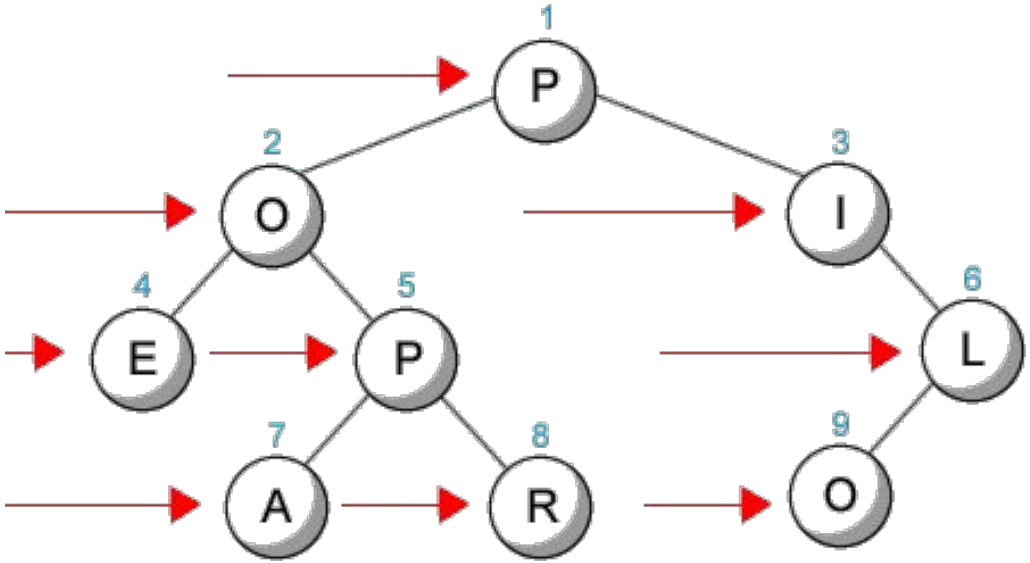
...

Profundidad

Depth First Search (DFS)

...

AMPLITUD



Algoritmo de *búsqueda en amplitud*:

1. Crear una lista de nodos llamada ABIERTA y asignarle el nodo raíz, que representa el estado inicial del problema planteado.

2. Hasta que ABIERTA esté vacía o se encuentre una meta, realizar las siguientes acciones:

2.1 Extraer el primer nodo de ABIERTA y llamarlo *m*.

2.2 Expandir *m* (generar todos sus sucesores). Para cada operador aplicable y cada forma de aplicación:

(1) Aplicar el operador a *m*, obtener un nuevo estado y crear un puntero que permita saber que su predecesor es *m*.

(2) Si el nuevo estado generado es meta, salir del proceso iterativo iniciado en **2.2** y devolver dicho estado.

(3) Incluir el nuevo estado al final de ABIERTA. (Una vez completado este proceso para todos los sucesores de *m* —cuando no se haya encontrado antes una meta— se continúa el proceso iterativo en el paso **2**.)

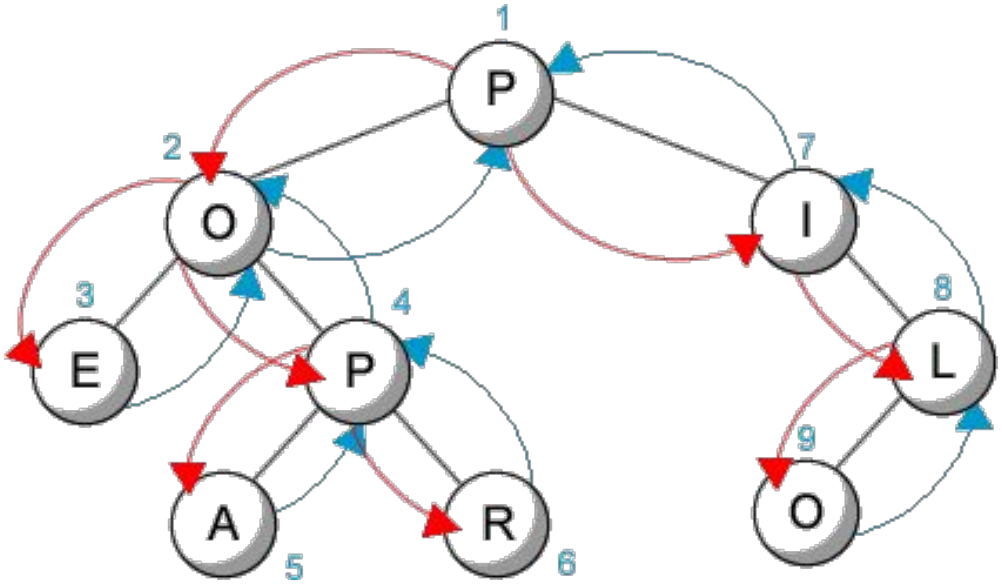
Observación: si el algoritmo termina con una meta, el camino de la solución puede obtenerse recorriendo los punteros creados desde el nodo meta al nodo raíz. En caso

Problemas Resueltos de Inteligencia Artificial Aplicada, Búsqueda y Representación.
Severino Fernandez, Jesus Gonzalez, Jose Mira.
Universidad Nacional de Educación a Distancia, Madrid, España

Ver pseudocódigo y código en github

[Búsqueda en Amplitud]

PROFUNDIDAD



Algoritmo de *búsqueda en profundidad*:

1. Crear una lista de nodos llamada ABIERTA y asignarle el nodo raíz, que representa el estado inicial del problema planteado.

2. Hasta que se encuentre una meta o se devuelva fallo, realizar las siguientes acciones:

(1) Si ABIERTA está vacía, terminar con fallo; en caso contrario, continuar.

(2) Extraer el primer nodo de ABIERTA y denominarlo *m*.

(3) Si la profundidad de *m* es igual a *lp* (límite de profundidad), regresar a **2**; en caso contrario, continuar.

(4) Expandir *m* creando punteros hacia este nodo desde todos sus sucesores, de forma que pueda saberse cuál es su predecesor. Introducir dichos sucesores al principio de ABIERTA siguiendo un orden arbitrario. (La "falta de orden" refleja el carácter no informado de este procedimiento.)

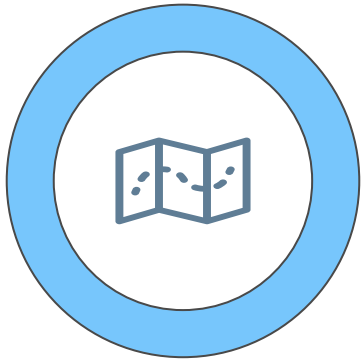
Problemas Resueltos de Inteligencia Artificial Aplicada, Búsqueda y Representación.
Severino Fernandez, Jesus Gonzalez, Jose Mira.
Universidad Nacional de Educación a Distancia, Madrid, España

código en github

Búsqueda en profundidad

02

Búsqueda informada o guiada



Explorar en primer lugar aquellas trayectorias que parecen más prometedoras a la hora de conducir a una solución (Heuristica).

Ahora, a diferencia de la búsqueda ciega, se le asignará a cada nodo un valor que dará una idea de lo cerca que está de la meta.

Métodos

Primero el mejor

...

Método de gradiente

...

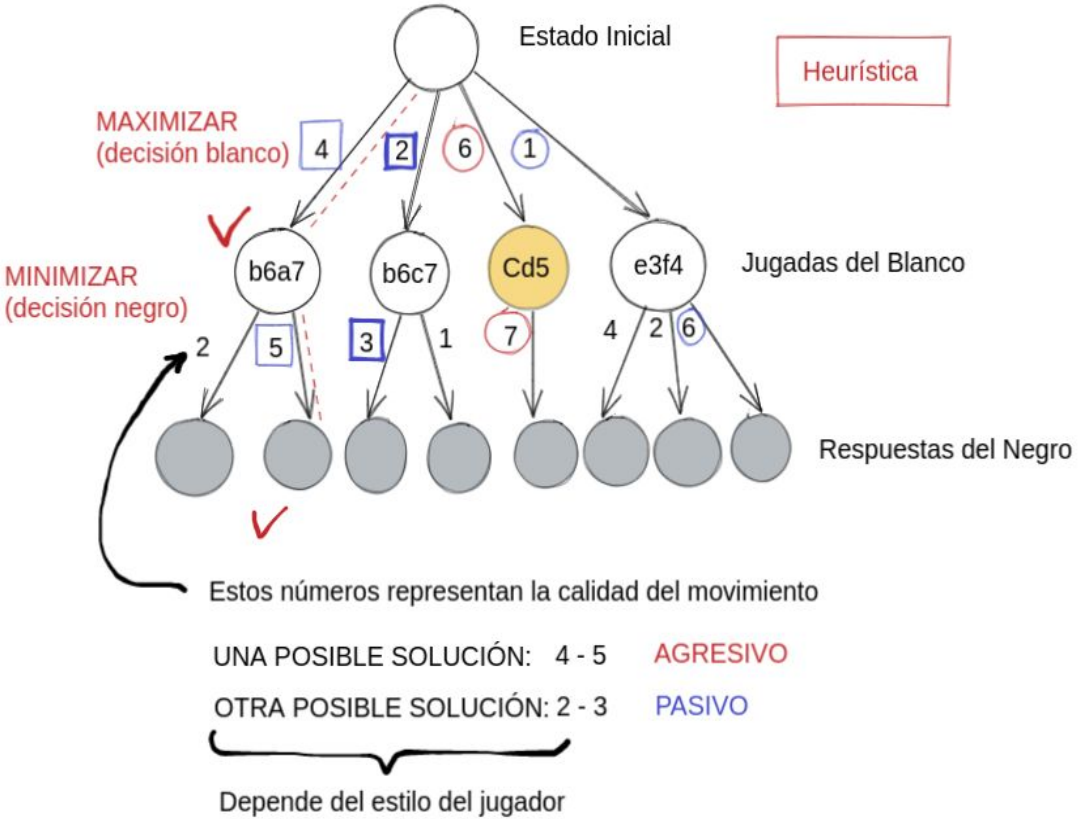
Algoritmo A*

...

Búsqueda con adversarios

Metodo Minimax
Método de poda $\propto - \beta$

...


Estado Inicial
Heurística
MAXIMIZAR
(decisión blanco)
4
2
6
1
b6a7
b6c7
Cd5
e3f4
Jugadas del Blanco
MINIMIZAR
(decisión negro)
2
5
3
1
7
4
2
6
Respuestas del Negro
Estos números representan la calidad del movimiento
UNA POSIBLE SOLUCIÓN: 4 - 5 AGRESIVO
OTRA POSIBLE SOLUCIÓN: 2 - 3 PASIVO
Depende del estilo del jugador

Procedimiento MINIMAX:
MINIMAX (*m*, *profundidad*, *jugador*)

1. Si *m* no tiene sucesores o si se considera que *m* ha alcanzado el límite de profundidad en *profundidad*, devolver *fev*(*m*) si *m* es un nodo MAX. En las mismas condiciones anteriores, devolver –*fev*(*m*) si *m* es un nodo MIN (Recordamos aquí que *fev*(*m*) representa lo prometedor que es el nodo *m* para MAX, independientemente de si *m* es un nodo MAX o MIN.).

2. Generar los sucesores de *m*:

2.1. Asignar a la variable *mejor* el valor mínimo que *fev* pueda tener (puede ser un valor de referencia previamente establecido).

$$mejor = \min_j\{fev(j)\ \forall j\}$$

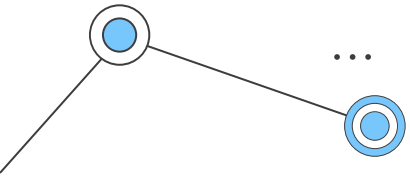
2.2. Para cada sucesor *n* de *m*:

(1) M(*n*) = MINIMAX(*n*, *profundidad* + 1, *C*(*jugador*))
C(*jugador*) es una función que cambia de jugador.

(2) *mejor* = max(–M(*n*), *mejor*)

3. Una vez que se han analizado recursivamente todos los sucesores de un determinado nivel (indicado por la variable profundidad), se devuelve el valor *mejor*.

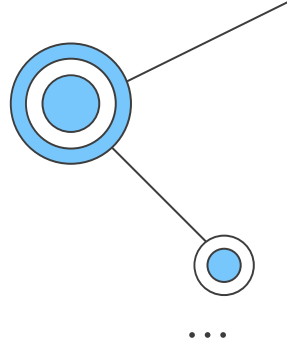
Observación: la primera llamada al procedimiento sería MINIMAX(*nodo-inicial*, 0, MAX).



```
graph = {0: [4, 2.1, 6, 1],
         4: [2, 5],
         2.1: [3, 1.1],
         6: [7],
         1: [4.1, 2, 6]
         }
```

```
def Minimax (m, profundidad, graph):
  if graph.get(m) == None:
    return m

  sucesores = graph.get(m)
  print(m)
  mejor = min(sucesores)

  for n in sucesores:
    M = Minimax(n, profundidad + 1, graph)

    mejor = max(M, mejor)

    return mejor

mejor = Minimax(0, 0, graph)
print(mejor)
```

```
0
4
2
```



código en github

Heuristica

Gracias