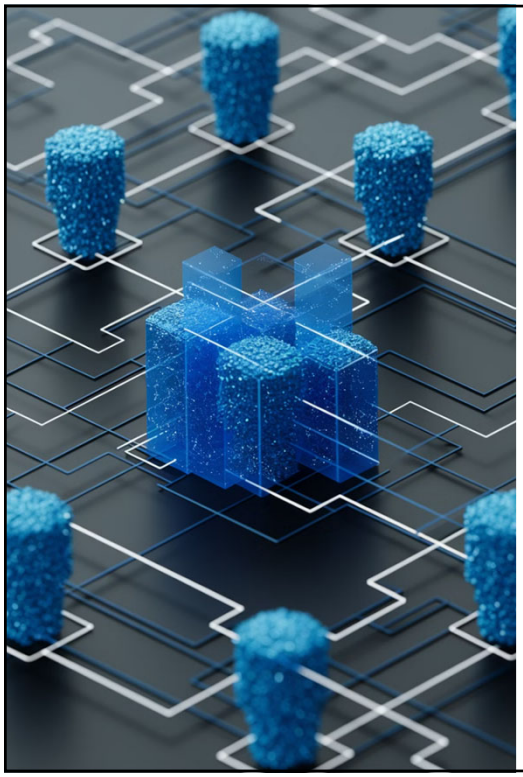


Resolución de problemas IA con algoritmos de búsquedas

Se exploran la resolución de problemas en IA centrada en los algoritmos de búsqueda. Se enfoca en cómo la IA encuentra soluciones óptimas de manera eficiente.

Jaime Alberto Guzmán Luna, Ph.D

1

Espacio de problemas

La representación formal de problemas para permitir su resolución mediante algoritmos.

Componentes del problema

- Espacio de Estados: Conjunto de todos los estados alcanzables.
- Acciones Posibles: Operadores que transforman un estado.
- Estado Inicial: El punto de partida de la búsqueda.
- Meta: Estado objetivo que se desean alcanzar.

Representación

Representable por un grafo donde los nodos son estados y las aristas son operadores.

Resolución de problemas

Búsqueda en el grafo. Normalmente, la búsqueda genera un árbol de exploración.

Parámetros importantes

- Factor de ramificación
- Profundidad del árbol de búsqueda determinan la complejidad.

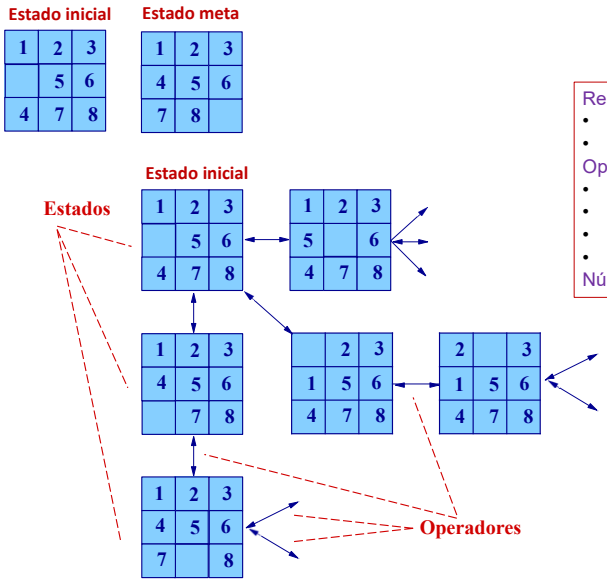
2

Visualización problema de búsqueda

Problema del 8-Puzzle

Componentes del problema

- Para el 8–puzzle se usa un cajón cuadrado en el que hay situados 8 bloques cuadrados. El cuadrado restante está sin rellenar.
- Cada bloque tiene un número. Un bloque adyacente al hueco puede deslizarse hacia él.
- El juego consiste en transformar la posición inicial en la posición final mediante el deslizamiento de los bloques.



Representación:
- Estado inicial: [[1,2,3],[h,5,6],[4,7,8]]
- Estado final: [[1,2,3],[4,5,6],[7,8,h]]

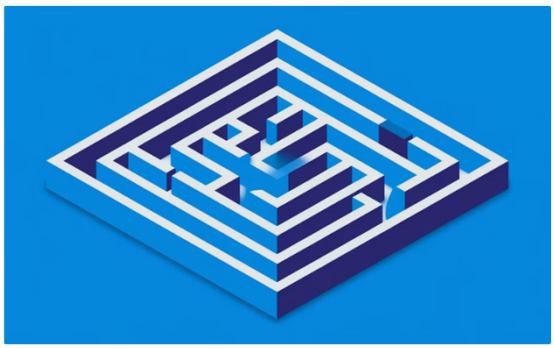
Operadores:
- Mover el hueco a la izquierda
- Mover el hueco arriba
- Mover el hueco a la derecha
- Mover el hueco abajo

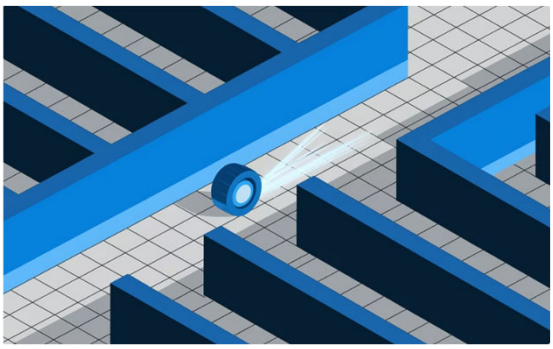
Número de estados = 9! = 362.880.

3

Estrategias de Búsqueda

Las estrategias de búsqueda en inteligencia artificial se dividen en dos categorías principales:





No informados (ciegos)

No utilizan información específica del problema.

Incluyen búsqueda en **amplitud** y **profundidad**.

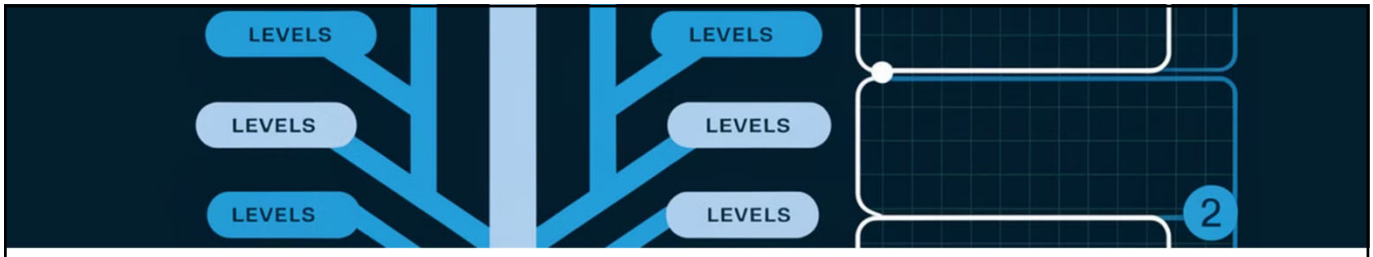
Informados (heurísticos)

Utilizan conocimiento específico del problema.

Incluyen búsqueda **primero mejor** y **A***.

2

4



Búsqueda en Amplitud (BFS)

Características

Explora todos los nodos a una profundidad antes de avanzar al siguiente nivel. Utiliza estructura FIFO (cola).
Eficiencia: buena si las metas están cercanas.
Problema: consume memoria exponencial

Propiedades

Completo: Sí (con factor de ramificación finito).

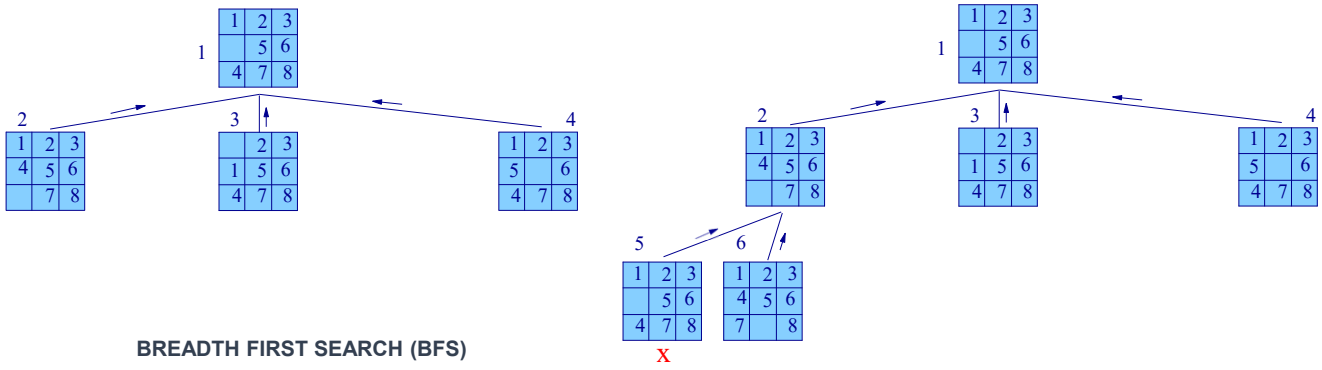
Óptimo: Sí (con costos iguales).

Complejidad

Temporal: O(b^d) donde b=factor de ramificación, d=profundidad.

5

Búsqueda en Amplitud -Ejemplo

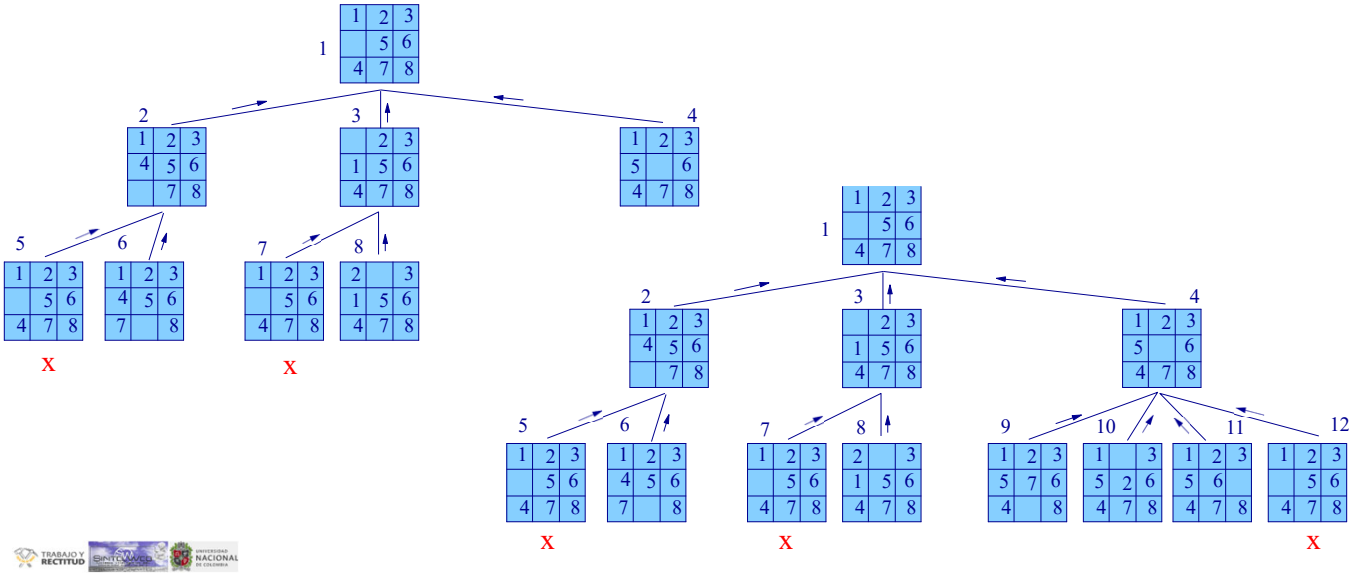


BREADTH FIRST SEARCH (BFS)

3

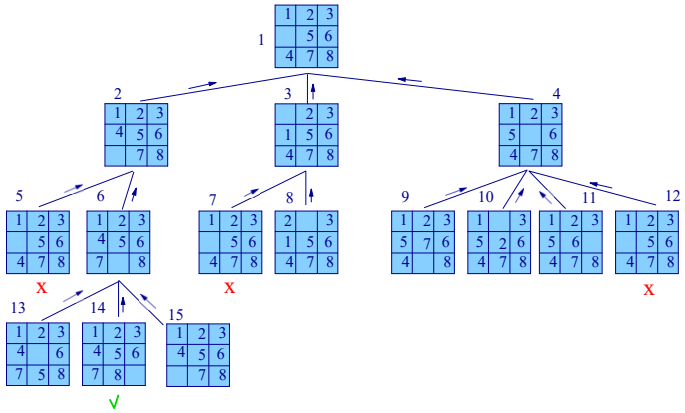
6

Búsqueda en Amplitud -Ejemplo



7

Búsqueda en Amplitud -Ejemplo



8

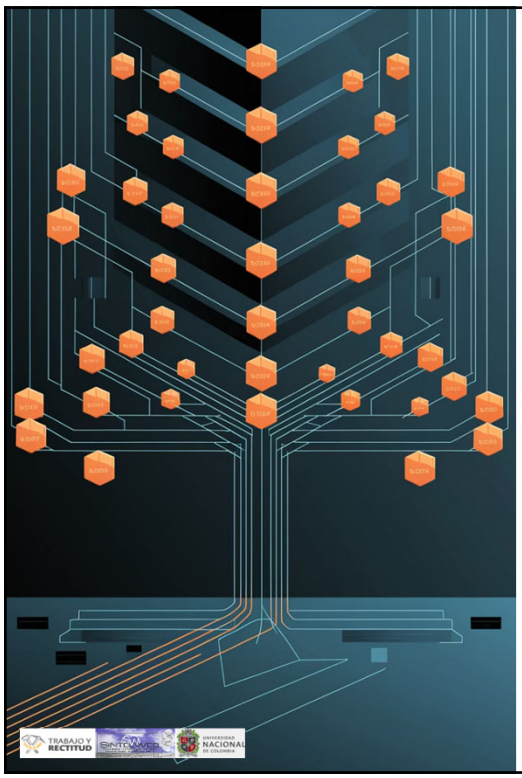
Algoritmo Búsqueda en amplitud

El algoritmo de búsqueda en amplitud (BFS) explora sistemáticamente todos los nodos de un nivel antes de pasar al siguiente nivel.

ALGORITMO Búsqueda-En-Amplitud(Estado-inicial Estado-Final):

1. Crear lista ABIERTA con el nodo inicial, I, (estado-inicial)
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO
   Quitar de ABIERTA el primer nodo, N
   Si N tiene sucesores
   Entonces Generar los sucesores de N
   Crear punteros desde los sucesores hacia N
   Si algún sucesor es nodo meta
   Entonces EXITO=Verdadero
   Si no Añadir **los sucesores al final** de ABIERTA
4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

9



Búsqueda en Profundidad (DFS)

Estrategia que explora una rama hasta el final antes de retroceder. Utiliza estructura LIFO (pila).

Características

Explora tan profundo como sea posible por una rama antes de retroceder y explorar otras alternativas. Prioriza la profundización sobre la amplitud.

Propiedades

Completo: No (riesgo de ciclos).
Óptimo: No (soluciones potencialmente subóptimas).
Eficiencia: bueno cuando metas alejadas de estado inicial, o problemas de memoria.
Problema: No es bueno cuando hay ciclos

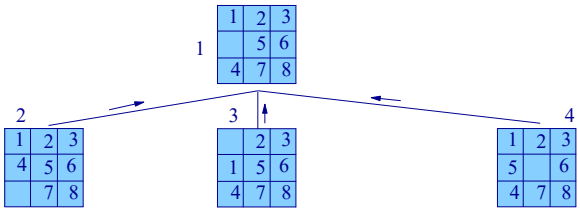
Complejidad

Temporal: O(b^m) donde b=factor de ramificación, m=profundidad máxima.

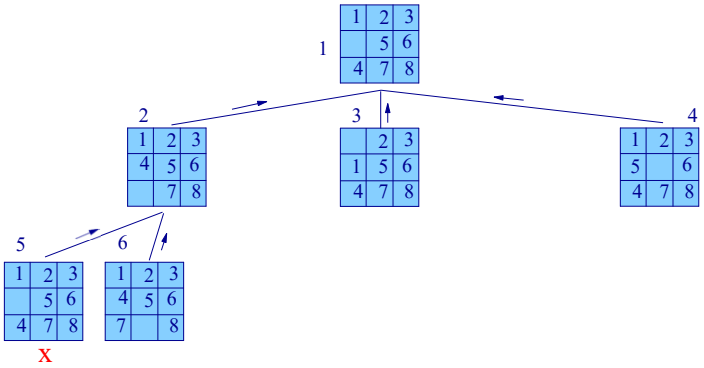
10

Búsqueda en Profundidad - Ejemplo

La búsqueda en profundidad (DFS) explora completamente una rama antes de retroceder.



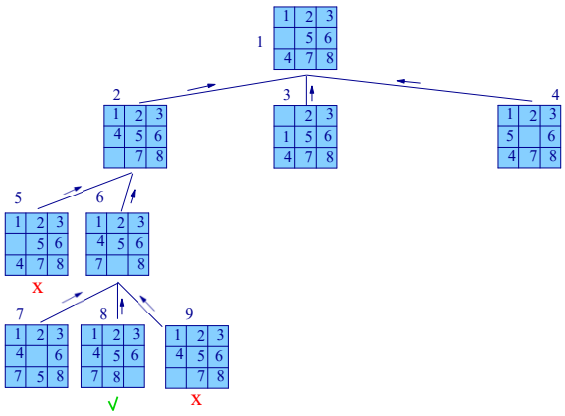
Depth-First Search-DFS



11

Búsqueda en Profundidad - Ejemplo

La búsqueda en profundidad (DFS) explora completamente una rama antes de retroceder.



12

Algoritmo Búsqueda en Profundidad

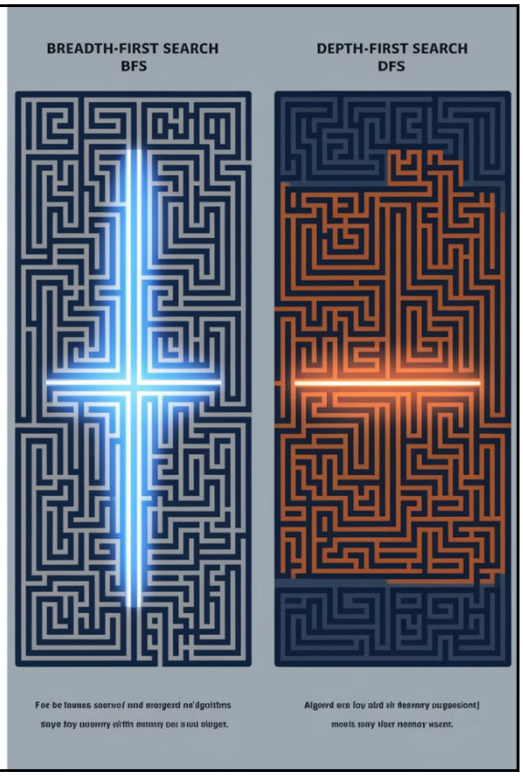
ALGORITMO Búsqueda-En-Profundidad(Estado-inicial Estado-Final):

1. Crear lista ABIERTA (pila) con el nodo inicial, I (estado-inicial),
   y su profundidad P=0
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO
   Quitar de ABIERTA el primer nodo
   Lo llamaremos N y a su profundidad P
   Si P < Profundidad-máxima y N tiene sucesores
   Entonces Generar los sucesores de N
   Crear punteros desde los sucesores hacia N
   Si algún sucesor es el Estado-Final
   Entonces EXITO=Verdadero
   Si no, Añadir los sucesores al inicio de ABIERTA
   y asignarles profundidad P+1
4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

13

Comparación: BFS vs DFS

Característica	Búsqueda en Amplitud	Búsqueda en Profundidad
Optimalidad	Sí (camino más corto)	No garantizada
Memoria	Mayor consumo	Menor consumo
Uso ideal	Soluciones a poca profundidad	Espacios grandes con múltiples soluciones



14

Búsqueda Informada (Heurística)

Definición de Heurística

Origen

Del griego *heuriskein*, descubrir: ¡Eureka!

Según el DRAE

"Técnica de la indagación y del descubrimiento".

En Inteligencia Artificial

Método para resolver problemas que no garantiza la solución, pero que en general funciona bien.

15

Búsqueda Informada (Heurística)

1. **Función heurística óptima**
   Estimación precisa
2. **Heurística admisible**
   No sobreestima
3. **Función h(n)**
   Costo estimado al objetivo

La heurística proporciona información adicional para guiar la búsqueda hacia los caminos más prometedores, mejorando la eficiencia.

16

Problema de referencia: Viaje a Romania

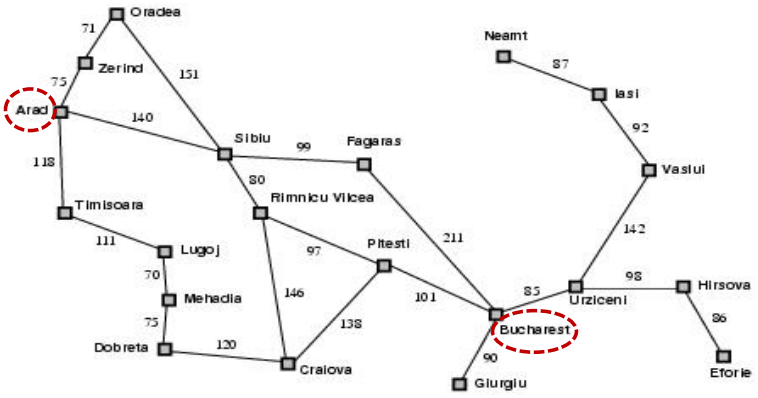
Componentes del problema

- Vacaciones en Romania: Actualmente en Arad.
- El carro sale mañana a Bucharest
- Formulación del objetivo:
  - Estar en Bucharest
- Formulación del problema:
  - Estados: varias ciudades
  - Acciones: conducir entre ciudades (ej. Ir a Arad, Ir a Fagaras, etc)
- Buscar solución:
  - Secuencia de ciudades, ej. Arad, Sibiu, Fagaras, Bucharest

Función Heurística

- Distancia en línea recta
- heurística(estado) = distancia(coordenadas(estado), coordenadas(Bucharest))

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



17

Búsqueda Primero el Mejor

1 **Selección de nodos**

Elige el nodo más prometedor primero

2 **Función de evaluación**

Usa directamente h(n)

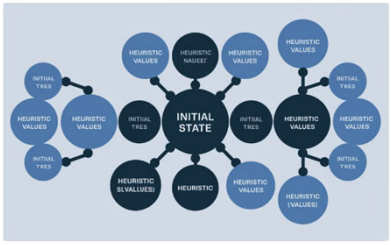
3 **Implementación**

Cola de prioridad: Menor valor primero

- Este algoritmo selecciona el nodo más prometedor según su heurística.
  - Elige siempre el nodo con menor valor h(n).

18

Búsqueda Primero el Mejor - Implementación



Nodo de búsqueda
estado + camino + heurística

Funciones de acceso
estado(nodo), camino(nodo) y heuristica-del-nodo(nodo)

Sucesores de un nodo heurístico

FUNCION SUCESOR(NODO,OPERADOR)
1. Hacer ESTADO-SUCESOR
   igual a APLICA(OPERADOR,ESTADO(NODO))
2. Si ESTADO-SUCESOR=NO-APLICABLE
   devolver NO-APLICABLE
   en caso contrario,
      devolver un nodo cuyo estado es ESTADO-SUCESOR, cuyo
      camino es el resultado de añadir OPERADOR a CAMINO(NODO)
      y cuya heurística es la de ESTADO-SUCESOR

FUNCION SUCESORES(NODO)
1. Hacer SUCESORES vacío
2. Para cada OPERADOR en *OPERADORES*,
   si SUCESOR(NODO,OPERADOR) ≠ NO-APLICABLE,
      incluir SUCESOR(NODO,OPERADOR) en SUCESORES
3. Devolver SUCESORES

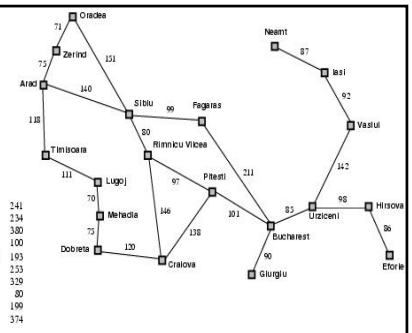
19

Búsqueda Primero el Mejor - Implementación

FUNCION BUSQUEDA-POR-PRIMERO-EL-MEJOR()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*, cuyo camino es vacío y cuya heurística es la de *ESTADO-INICIAL*);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   1. Hacer ACTUAL el primer nodo de ABIERTOS
   2. Hacer ABIERTOS el resto de ABIERTOS
   3. Poner el nodo ACTUAL en CERRADOS.
   4. Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
      1. devolver el nodo ACTUAL y terminar.
      2. en caso contrario,
         1. Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
         2. Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES en ABIERTOS y ordenar en orden creciente de sus heurísticas
3. Devolver FALLO.

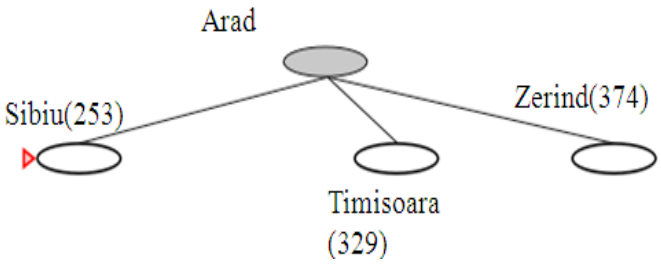
20

Primero Mejor – Problema del viaje



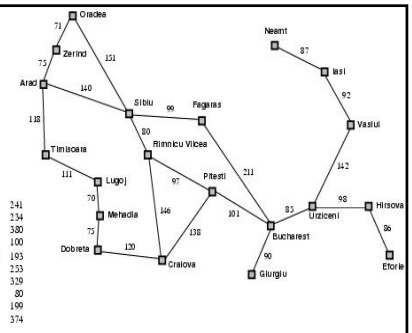
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Supongamos que queremos utilizar la búsqueda primero el mejor para resolver el problema de los viajes de Arad a Bucharest.
- El estado inicial = Arad

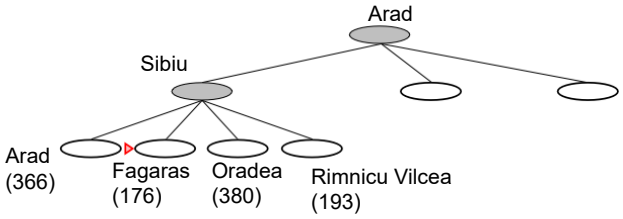


- El primer paso de expansión producido:
  - Sibiu, Timisoara and ZerindP
  - Primero el mejor seleccionará a Sibiu

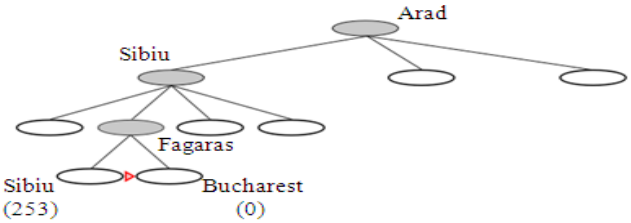
Primero Mejor – Problema del viaje



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

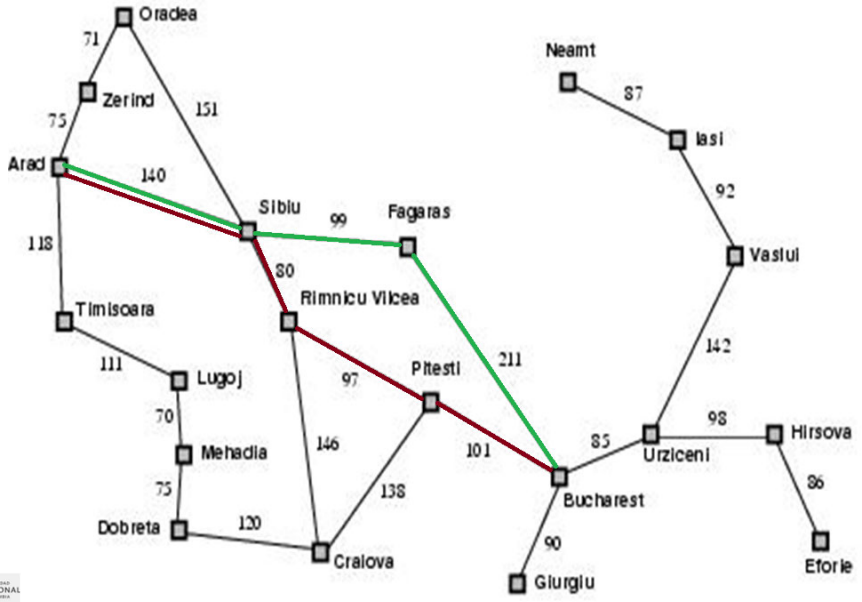


- Si Sibiu se expande se obtiene:
  - Arad, Fagaras, Oradea y Rimnicu Vilcea
- La búsqueda primero el mejor seleccionará: Fagaras



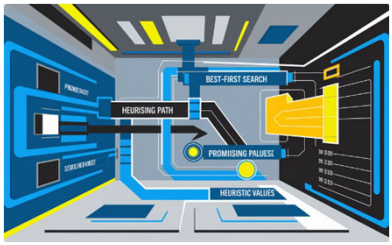
- Sí se expande Faragas, se obtiene:
  - Sibiu y Bucarest
- **Objetivo alcanzado!!**
- Sin embargo, **no es óptimo** (ver Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucarest)

Primero Mejor – Problema del viaje



23

Búsqueda Primero el Mejor - Propiedades



Complejidad

- Complejidad en tiempo $O(b^m)$, donde:
  - b : factor de ramificación.
  - m: profundidad máxima del árbol de búsqueda.
- Complejidad en espacio: $O(b^m)$.
  - Guarda todos los nodos en memoria.
- En la práctica, tiempo y espacio necesario dependen del problema concreto y de la calidad de la heurística usada.

Resultado

- No es completa, en general.
  - Mala heurística podría llevar a bucles.
- No es optima (no garantiza soluciones con el menor número de operadores).
  - La heurística podría guiar hacia una solución no mínima
  - Camino encontrado puede ser subóptimo pero se exploran menos nodos.

24

Algoritmo A*

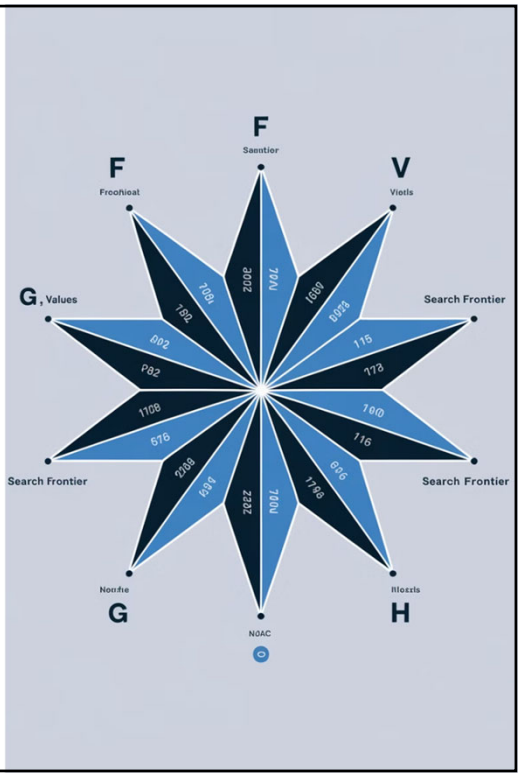
1 **Función de evaluación**

Combina costo acumulado y heurística:

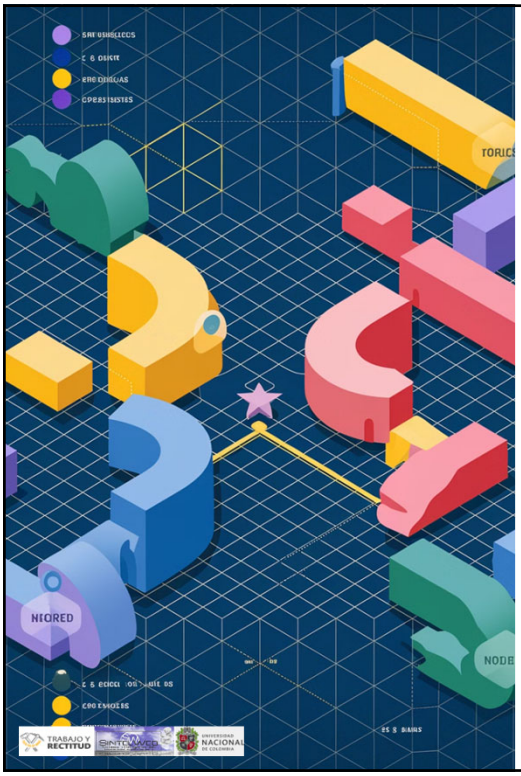
- f(n) = g(n) + h(n)

2 **Componentes**

g(n): costo desde inicio hasta n.
h(n): costo estimado desde n hasta objetivo.



25



Algoritmo A* - Funcionamiento

1 **Inicialización**

Comenzar con el nodo inicial y calcular:
f(n) = g(n) + h(n) para cada sucesor.

2 **Expansión**

Seleccionar el nodo con menor valor f(n) de la lista abierta.

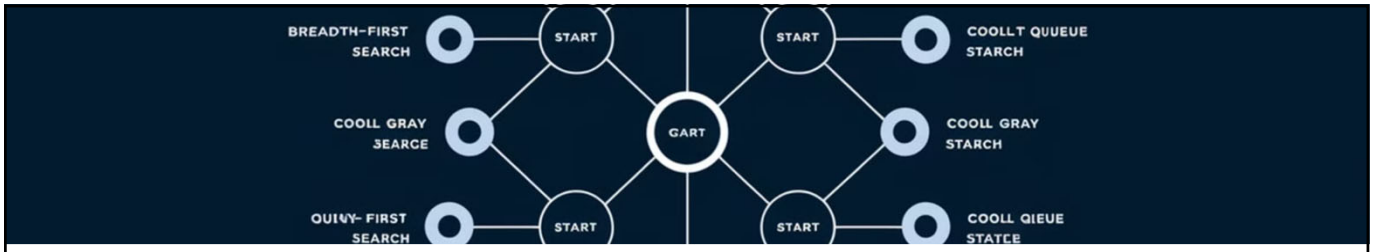
3 **Actualización**

Recalcular valores f(n) cuando se encuentre un camino mejor a un nodo.

4 **Finalización**

Alcanzar el objetivo con la garantía del camino óptimo.

26



Algoritmo A* - Implementación

Sucesores de un Nodo

- Sucesores de un nodo con costo y heurística se describe en el pseudocodigo de al lado

FUNCION SUCESOR(NODO,OPERADOR)

1. Hacer ESTADO-SUCESOR igual a APLICA(OPERADOR,ESTADO(NODO))
2. Si ESTADO-SUCESOR=NO-APLICABLE devolver NO-APLICABLE en caso contrario, devolver un nodo cuyo estado es ESTADO-SUCESOR, cuyo camino es el resultado de añadir OPERADOR a CAMINO(NODO) y cuyo costo es COSTO-CAMINO(CAMINO(NODO)) más COSTO-DE-APLICAR-OPERADOR(ESTADO(NODO),OPERADOR) y cuyo costo-más-heurística es el costo anterior más HEURISTICA(ESTADO(NODO))

27

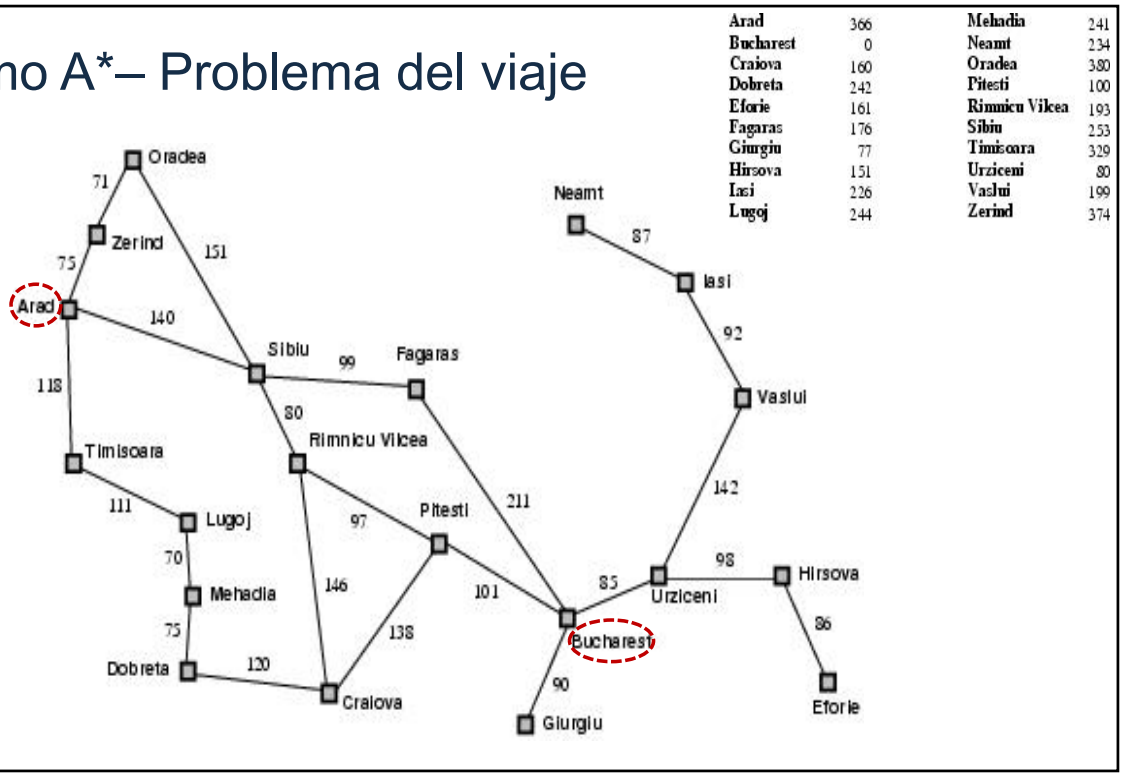
Algoritmo A* - Implementación

FUNCION BUSQUEDA-A-ESTRELLA()

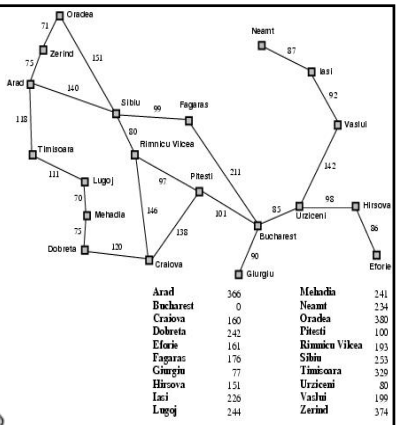
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo cuyo estado es *ESTADO-INICIAL*, cuyo camino es vacío, cuyo costo es 0, y cuyo costo-más-heurística es HEURISTICA(*ESTADO-INICIAL*))
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   1. Hacer ACTUAL el primer nodo de ABIERTOS Hacer ABIERTOS el resto de ABIERTOS
   2. Poner el nodo ACTUAL en CERRADOS.
   3. Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
      1. devolver el nodo ACTUAL y terminar.
      2. en caso contrario,
         1. Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) que o bien tienen un estado que no aparece en los nodos de ABIERTOS ni de CERRADOS, o bien su costo es menor que cualquier otro nodo con el mismo estado que apareciera en ABIERTOS o en CERRADOS
         2. Hacer ABIERTOS el resultado de incluir NUEVOS-SUC en ABIERTOS y ordenar todo en orden creciente de del costo-mas-heurística de los nodos
3. Devolver FALLO.

28

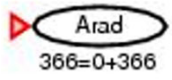
Algoritmo A*– Problema del viaje



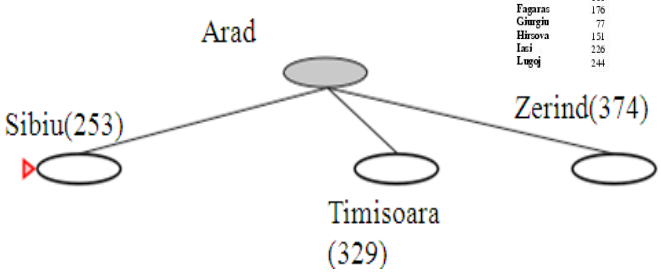
Algoritmo A* – Problema del viaje



Estado inicial

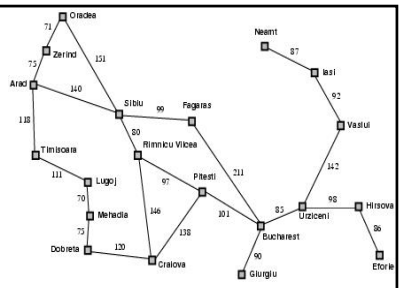


- Buscar Bucharest empezando en Arad
- f(Arad) = c(??,Arad)+h(Arad)=0+366=366

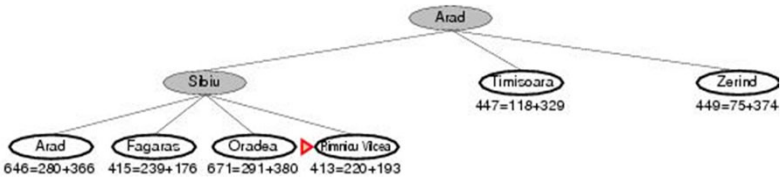


- El primer paso de expansión producido:
  - Sibiu, Timisoara and ZerindP
  - Primero el mejor seleccionará a Sibiu

Algoritmo A* – Problema del viaje



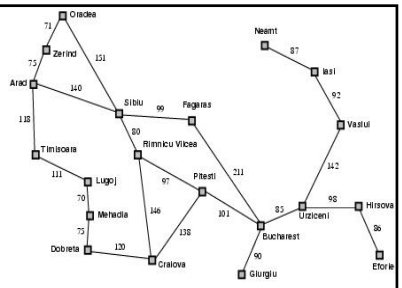
Después de expandir Sibiu



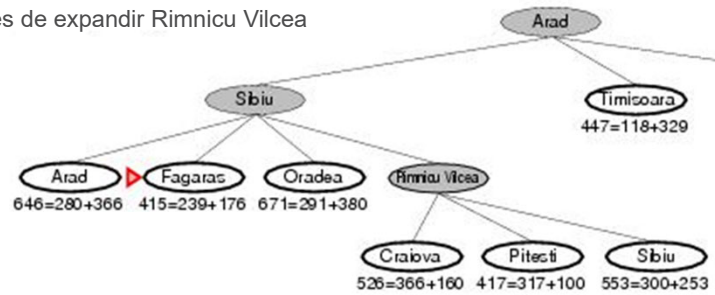
- Expandir Sibiu y determinar f(n) para cada nodo
  - f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
  - f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415
  - f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
  - f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413
- Mejor selección es Rimnicu Vilcea

31

Algoritmo A* – Problema del viaje



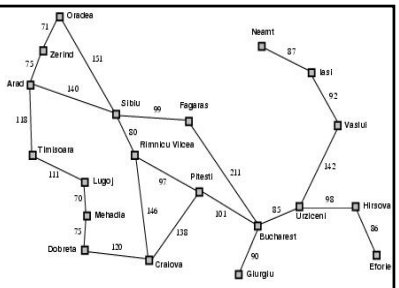
Después de expandir Rimnicu Vilcea



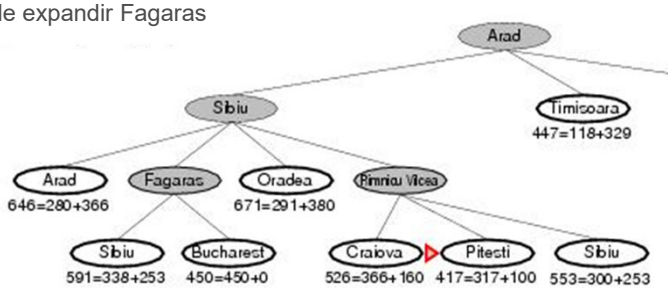
- Expandir Rimnicu Vilcea y determinar f(n) para cada nodo
  - f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
  - f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
  - f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553
- Mejor selección es Fagaras

32

Algoritmo A* – Problema del viaje



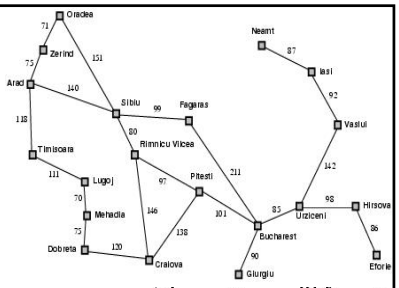
Después de expandir Fagaras



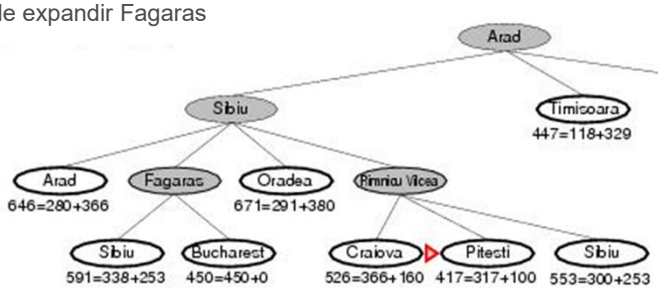
- Expandir Fagaras y determinar f(n) para cada nodo
  - f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591
  - f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450
- Mejor selección es Pitesti !!!

33

Algoritmo A* – Problema del viaje



Después de expandir Fagaras



- Expandir Pitesti y determinar f(n) para cada nodo
  - f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418
- Mejor selección es Bucharest !!!
  - Solución óptima (solo si h(n) es admisible)
- Note los valores a lo largo del camino óptimo !!

- Solución:
  - Arad
  - Sibiu
  - Rimnicu
  - Pitesti
  - Bucharest

34

Algoritmo A*- Propiedades

1 Heurística Admisible

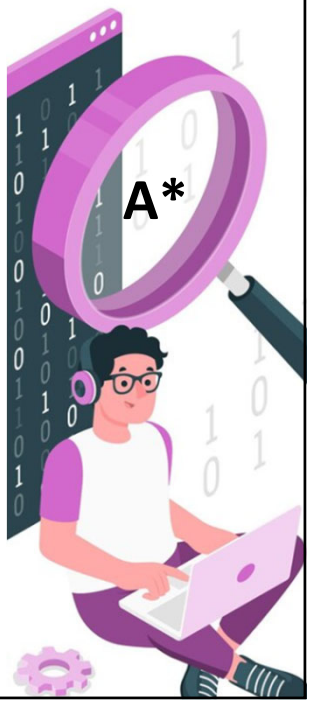
- Es completa.
- Encuentra siempre solución óptima.

2 Complejidad

- En tiempo: En el caso peor, con heurística de pésima calidad, la complejidad será exponencial, mientras en el caso mejor, con una buena h'(n), el algoritmo se ejecutará en tiempo lineal. O(b^d).
- En espacio: Guarda todos los nodos en memoria. Es Exponencial. O(b^d)
- b=factor de ramificación, d= profundidad de la solución óptima.

3 Práctica

En la práctica, el costo en tiempo y espacio depende del problema particular y de la calidad de la heurística usada



35

Comparativa de Algoritmos de Búsqueda

Los diferentes algoritmos de búsqueda ofrecen distintos balances entre eficiencia, completitud y optimalidad.

Algoritmo	Eficiencia Espacial	Eficiencia Temporal	Completitud	Optimalidad
BFS (Búsqueda en Amplitud)	25% - O(b^d)	50% - O(b^d)	Garantiza encontrar una solución si existe.	Óptimo solo cuando todos los pasos tienen igual costo.
DFS (Búsqueda en Profundidad)	75% - O(bm)	50% - O(b^m)	No completo en espacios infinitos.	No óptimo, puede encontrar soluciones subóptimas.
Best-First (Primero el Mejor)	50% - O(b^m)	75% - O(b^m)	No completo.	No óptimo, puede encontrar soluciones subóptimas.
A*	50% - O(b^d)	90% - O(b^d)	Garantiza encontrar una solución si existe.	Siempre óptimo con heurística admisible.

A* ofrece mejor balance entre optimalidad y eficiencia.

- b=factor de ramificación (número de hijos por nodo)
- m=profundidad máxima
- d= profundidad de la solución óptima.

36

Aplicaciones Prácticas

Navegación y rutas

Mapas digitales y GPS utilizan A* para encontrar rutas óptimas. Robots móviles planifican trayectorias evitando obstáculos.

Video juegos

Ajedrez y Go emplean búsquedas con Minimax. Videojuegos implementan pathfinding con A* para movimientos de personajes no jugadores (NPC).

IA en Web y E-Commerce

Mejorar las capacidades de búsqueda en sitios web y plataformas de comercio electrónico

37

Conclusiones

Selección de algoritmo

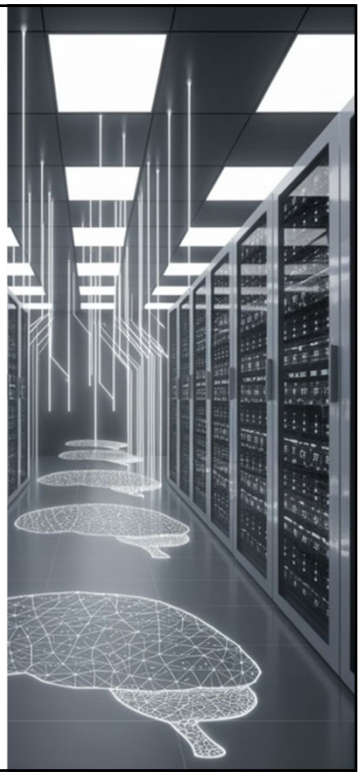
Considerar características del espacio de estados y disponibilidad de heurísticas informativas.

Tendencias actuales

Combinación con técnicas de aprendizaje y heurísticas generadas por redes neuronales.

Eficiencia

Balancear requisitos de optimalidad con restricciones de memoria y tiempo.



38