Universidad de Oviedo

Intelligent Scheduling and Optimization

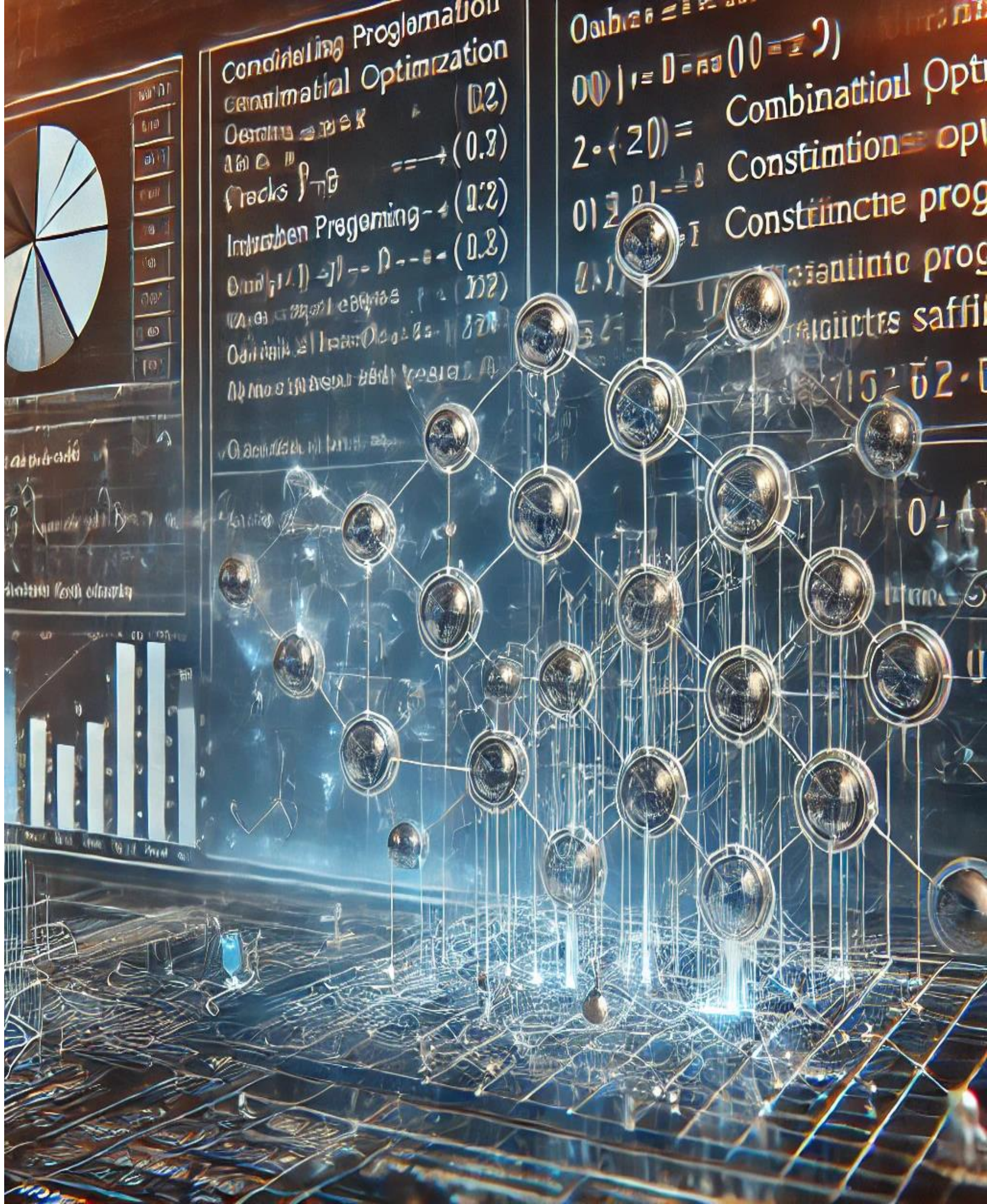
Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

Tema 3: Búsqueda heurística en espacios de estados

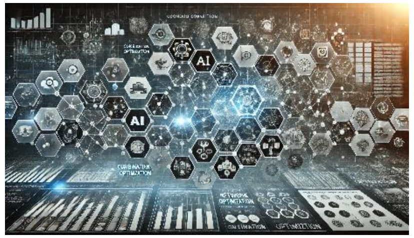
Ramiro Varela Arias
ramiro@uniovi.es

Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática

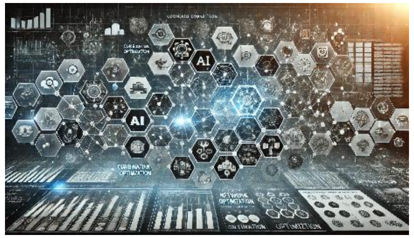
ceisIA Centro de Estudios sobre el Impacto Social de la Inteligencia Artificial | Universidad de Oviedo



Introducción



- La resolución de problemas con búsqueda heurística en espacios de estados consiste en
  - Modelar un espacio de soluciones mediante un grafo dirigido con costes positivos. Con un estado inicial y uno o varios objetivos. Se define de forma implícita por las operaciones
    - estadoInicial() genera el estado inicial
    - esObjetivo(s) comprueba si un estado s es objetivo
    - acciones(s) genera la lista de acciones aplicables al estado s
    - resultado(s,a) genera el estado sucesor de s al aplicarle la acción a
    - coste(s1,a,s2) calcula el coste de la acción a que lleva del estado s1 al estado s2
  - Elegir un algoritmo de búsqueda para calcular caminos entre el estado inicial y los objetivos
  - Definir funciones heurísticas de evaluación para guiar al algoritmo de búsqueda



Introducción

- Vamos a considerar tres problemas
  - Cálculo de rutas en mapas
  - El 8-puzzle
  - El problema del viajante de comercio (TSP)

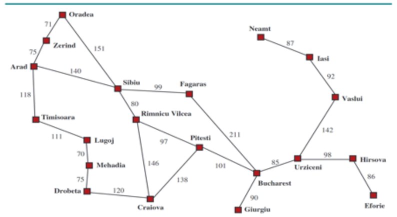
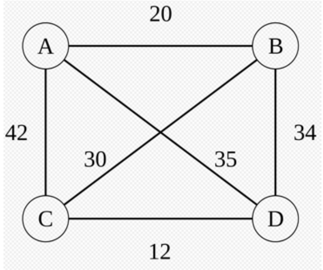
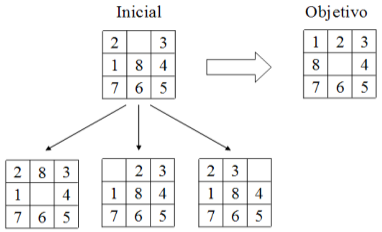


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

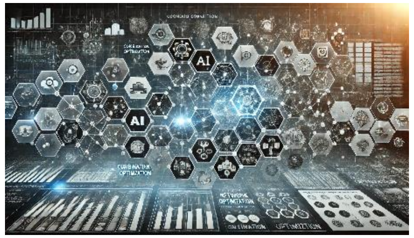




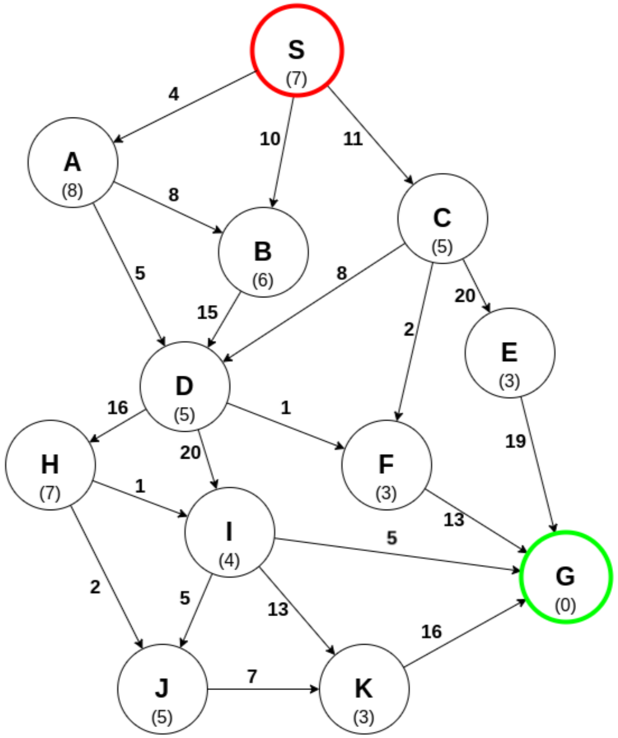
- Y el algoritmo genérico: Best First Search (BFS)
  - Incluida la versión concreta A* (A estrella)
  - Y una implementación en Python (aima-Python)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

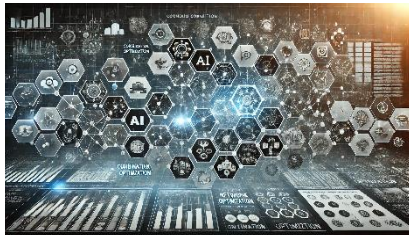
1.- Espacio de búsqueda



- El espacio de búsqueda es un grafo simple dirigido con arcos etiquetados con costes positivos
  - Nodos: estados que representan subproblemas
    - 1 estado inicial
    - 1 ó más estados objetivo
  - Arcos: representan acciones elementales
    - Permiten pasar de un estado n_1 a un estado sucesor n_2
    - Tienen asociado un coste no negativo $c(n_1, accion, n_2)$ ó $c(n_1, n_2)$
  - Solución del problema
    - Es cualquier camino desde el estado inicial a uno de los estados objetivo
    - El coste de la solución es la suma de los costes de los arcos del camino

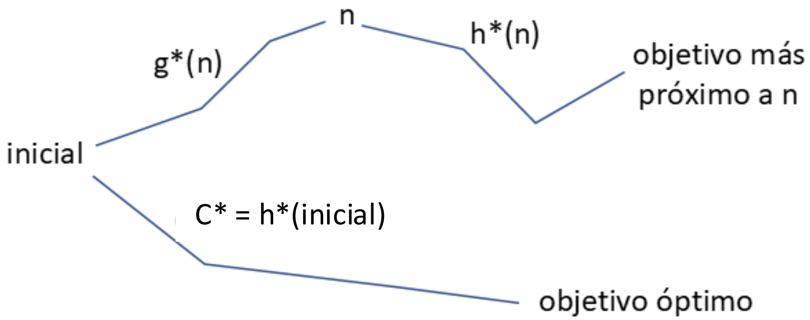


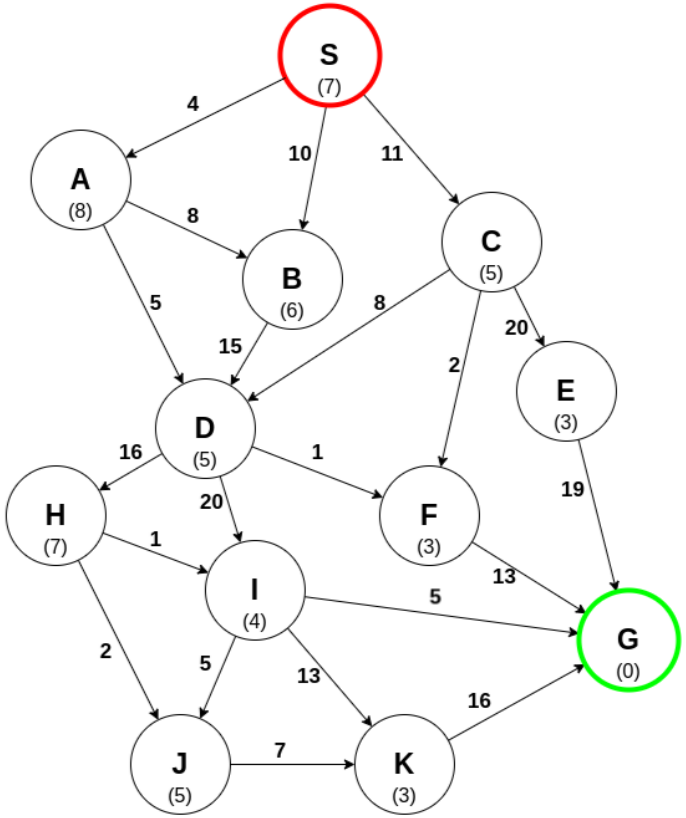
Espacio de búsqueda
Notación



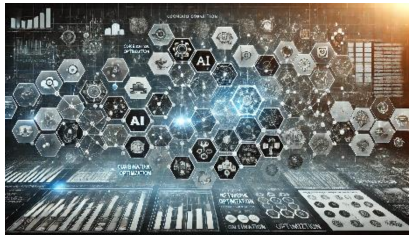
- Dado un espacio de búsqueda, definimos
  - $g^*(n)$ es el coste mínimo desde el inicial a n
  - $h^*(n)$ es el coste mínimo de n a los objetivos
  - $C^* = h^*(\textit{inicial})$, es el coste de la solución óptima

$g^*(n) + h^*(n)$, es el coste mínimo desde el inicial a los objetivos condicionado a pasar por n





Ejemplo: Calculo de rutas en mapas



- Se trata de calcular la mejor ruta entre dos ciudades en un mapa, por ejemplo entre Arad y Bucharest

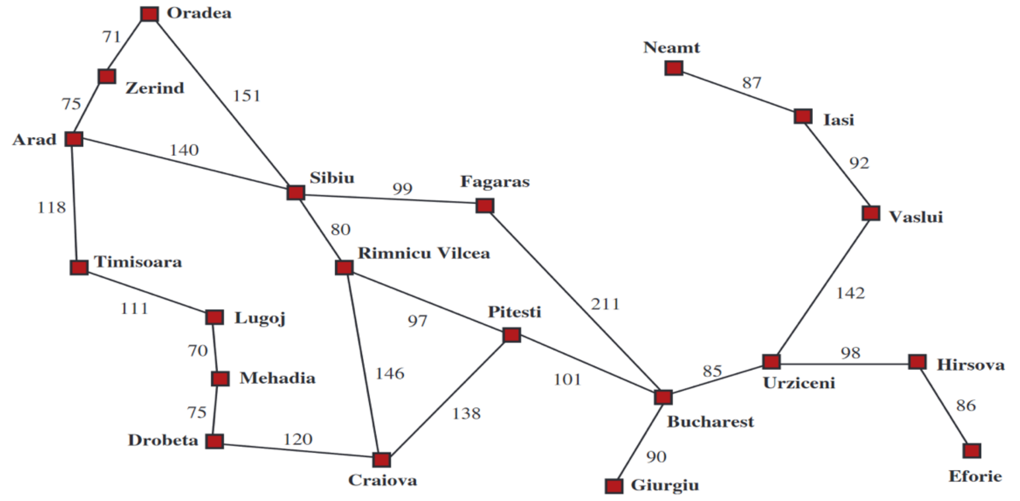
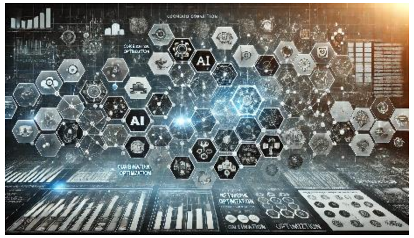


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

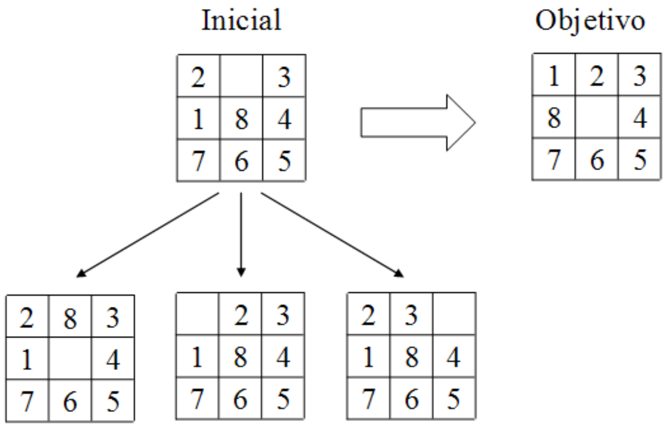
- Estados: ciudades en el mapa de Rumanía
  - estadoInicial(): Arad
  - esObjetivo(s): s es Bucarest?
- Acciones: conexiones entre las ciudades
  - acciones(s): lista de ciudades conectadas con s
  - resultado(s, a): estado definido por la ciudad a
  - c(s1,s2): distancia de la conexión entre s1 y s2

Intelligent Scheduling and Optimization

Ejemplo: El problema del 8-puzzle

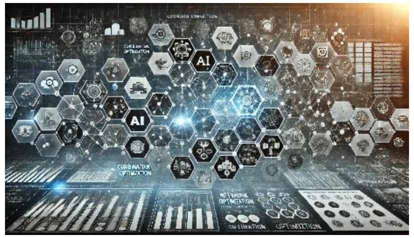


- Dado un tablero de 3 × 3 posiciones con 8 fichas y una casilla vacía,
  - Se trata de buscar la secuencia mínima de movimientos para llegar desde una situación inicial a otra, objetivo, en la que las fichas están ordenadas como se indica en la figura
  - Un movimiento elemental consiste en mover una ficha a la posición vacía si ésta es adyacente ortogonalmente

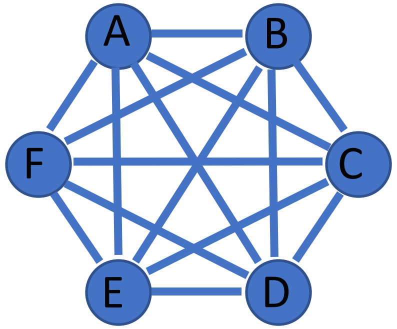


- Estados:
  - Situaciones posibles de las 8 fichas en el tablero
  - Hay un solo objetivo
- Acciones:
  - 2, 3 ó 4 para cada estado
  - El coste es 1 siempre ya que se trata de minimizar el número de movimientos

Ejemplo: El problema del viajante de comercio (TSP)



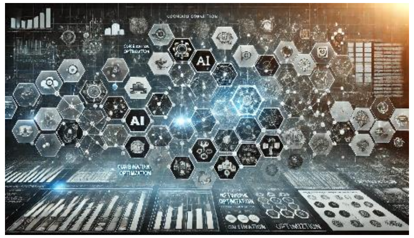
- En el TSP (*Traveling Salesman Problem*), se trata de calcular un recorrido sobre una serie de ciudades, con origen y destino en la ciudad A, visitando cada ciudad una sola vez, y con un coste mínimo
- Ejemplo (TSP simétrico con conexiones entre todas las ciudades)



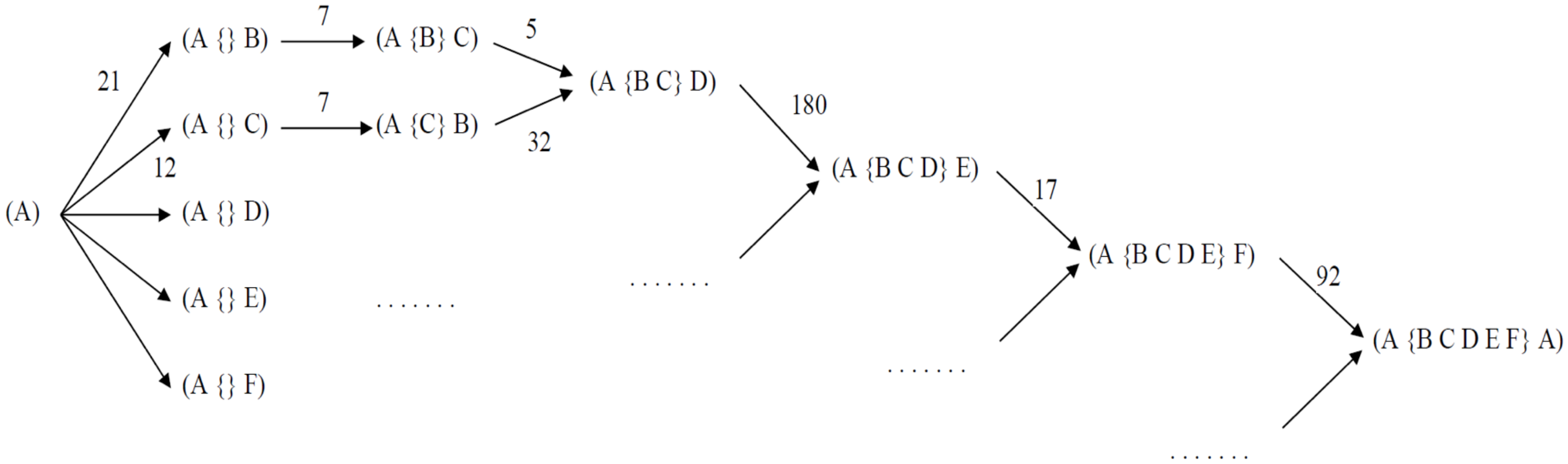
	A	B	C	D	E	F
A		21	12	15	113	92
B			7	32	25	9
C				5	18	20
D					180	39
E						17

Intelligent Scheduling and Optimization

Ejemplo: El problema del viajante de comercio (TSP)



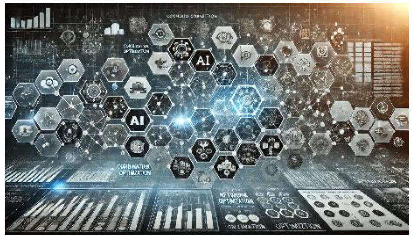
- Espacio de búsqueda
  - **Un estado es un par (conjunto de ciudades visitadas, la ciudad actual)**
  - Solo hay un estado objetivo



  - Una solución es un camino desde el inicial al objetivo
  - Cada estado representa un subproblema, pero no la forma de llegar desde el inicial hasta él

Intelligent Scheduling and Optimization

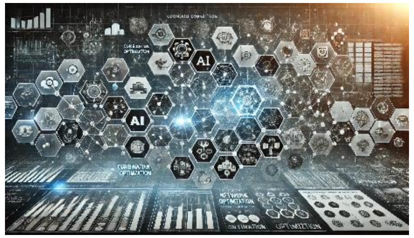
2.- Algoritmos de Búsqueda



- Algoritmo "Best First Search"

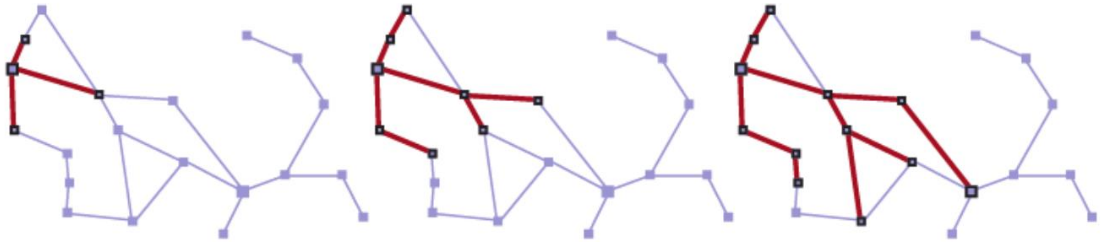
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Algoritmos de Búsqueda

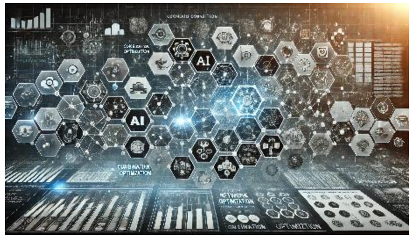


- Algoritmo ”Best First Search”
  - Desarrolla un árbol de búsqueda hasta que a través de una rama se encuentra una solución

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

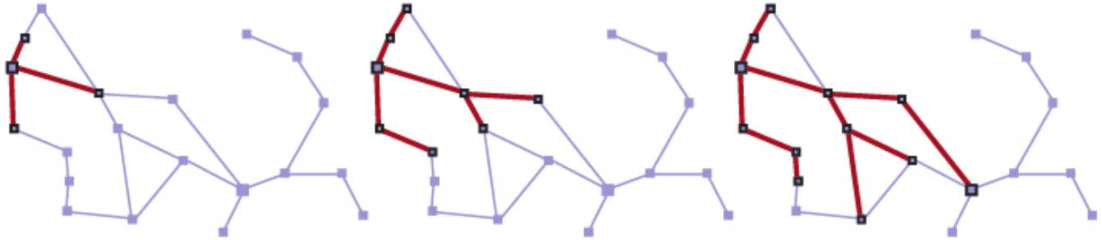


Algoritmos de Búsqueda

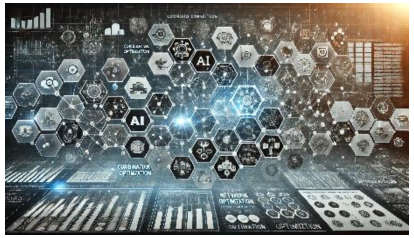


- Algoritmo "Best First Search"
  - Desarrolla un árbol de búsqueda hasta que a través de una rama se encuentra una solución
  - Permite introducir conocimiento sobre el dominio el problema a través de la función f (función heurística)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

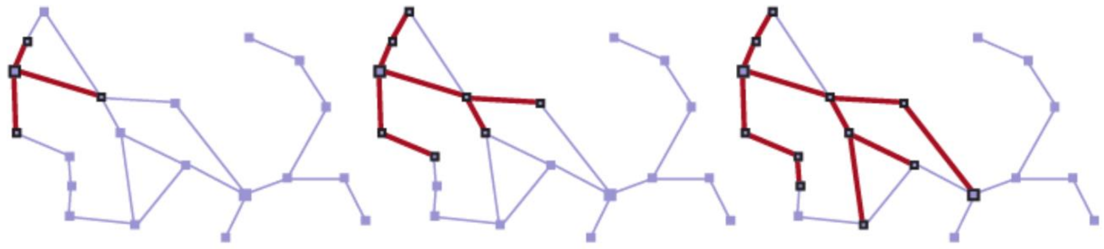


Algoritmos de Búsqueda

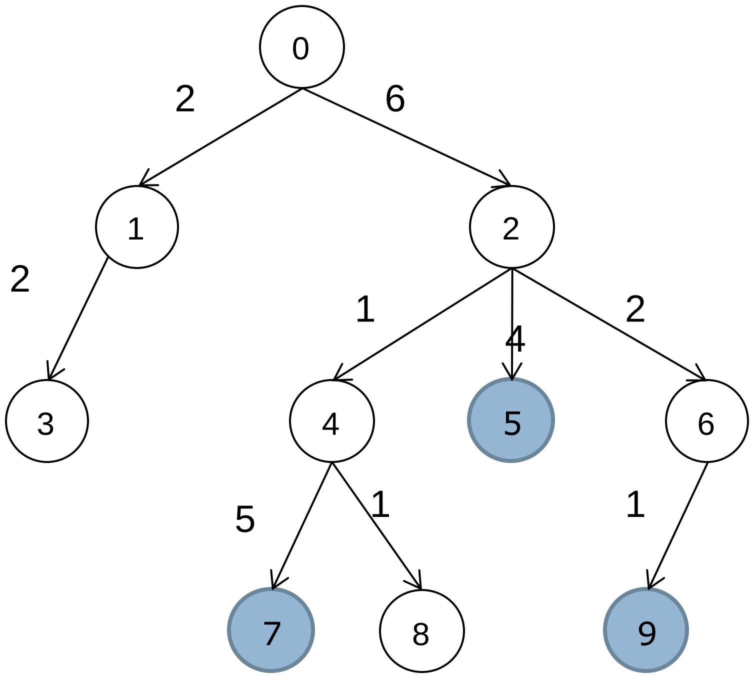


- Algoritmo ”Best First Search”
  - Desarrolla un árbol de búsqueda hasta que a través de una rama se encuentra una solución
  - Permite introducir conocimiento sobre el dominio el problema a través de la función f (función heurística)
  - Dependiendo de las características de la función f, el algoritmo puede ser exacto o aproximado, o ser más o menos eficiente

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

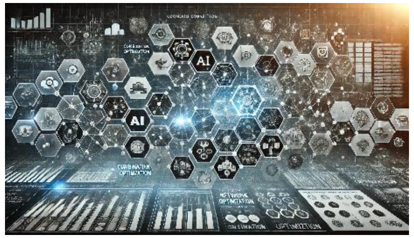


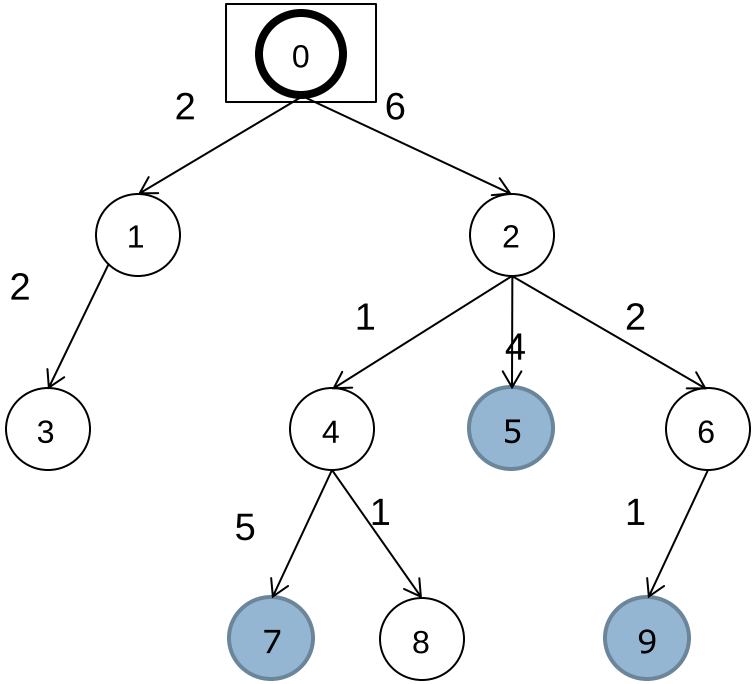
Algoritmos de Búsqueda
Ejemplo simple de BFS



Intelligent Scheduling and Optimization

Algoritmos de Búsqueda
Ejemplo simple de BFS

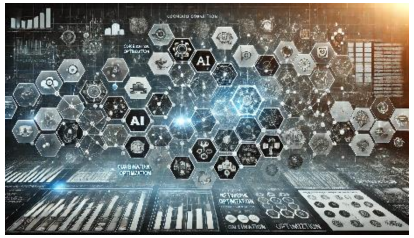


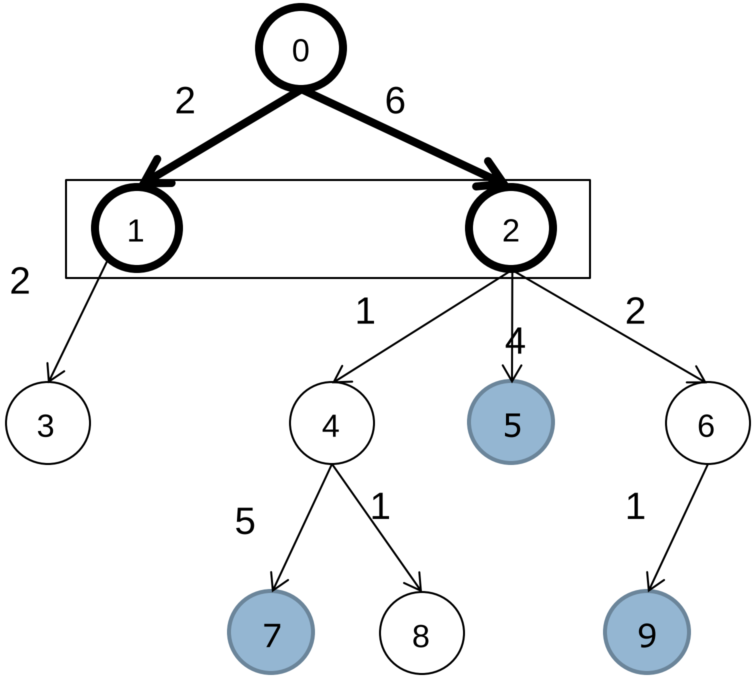


frontier = (0)

Intelligent Scheduling and Optimization

Algoritmos de Búsqueda
Ejemplo simple de BFS

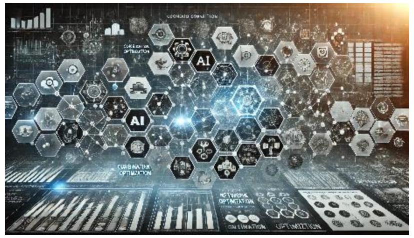


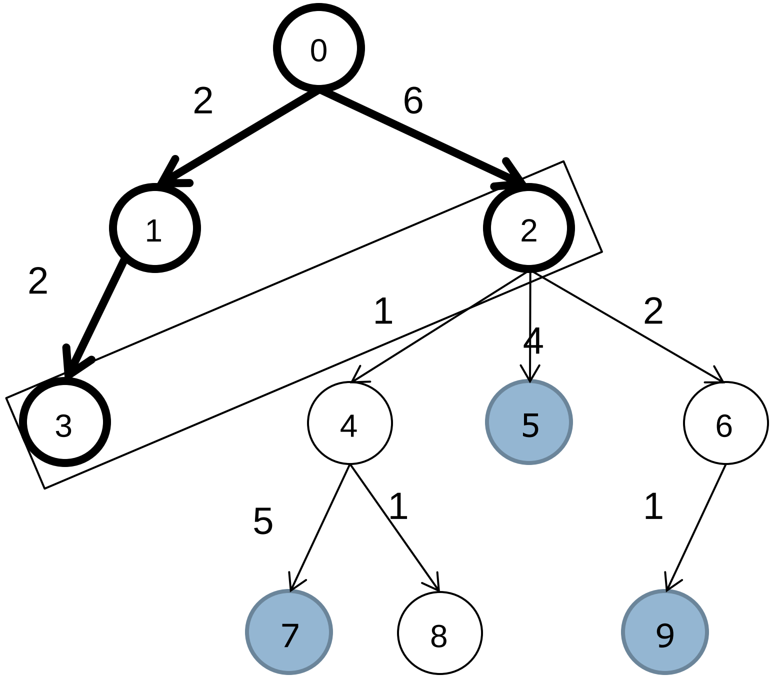


frontier = (0)
frontier = (1,2)

Intelligent Scheduling and Optimization

Algoritmos de Búsqueda
Ejemplo simple de BFS

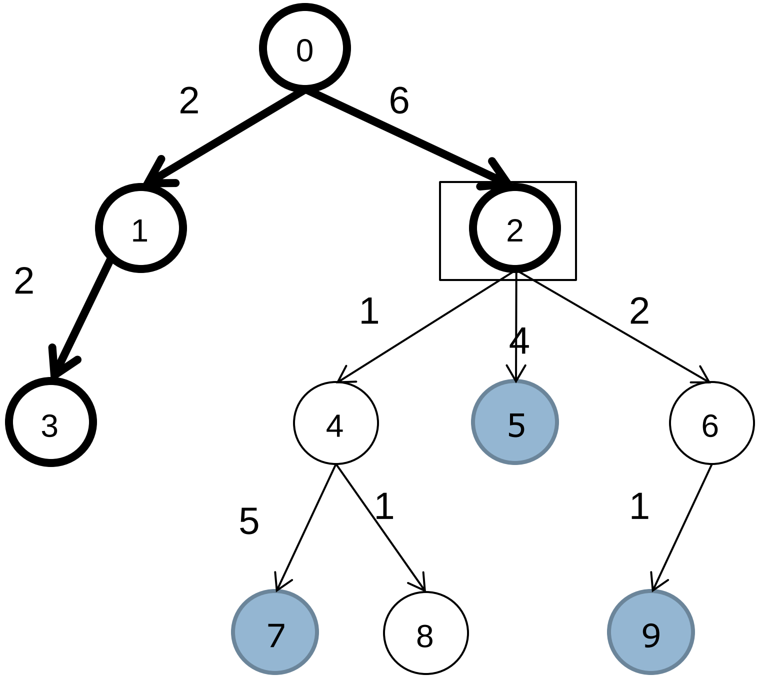




frontier = (0)
frontier = (1,2)
frontier = (3,2)

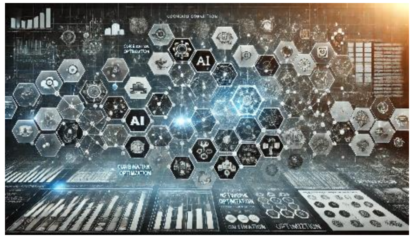
Intelligent Scheduling and Optimization

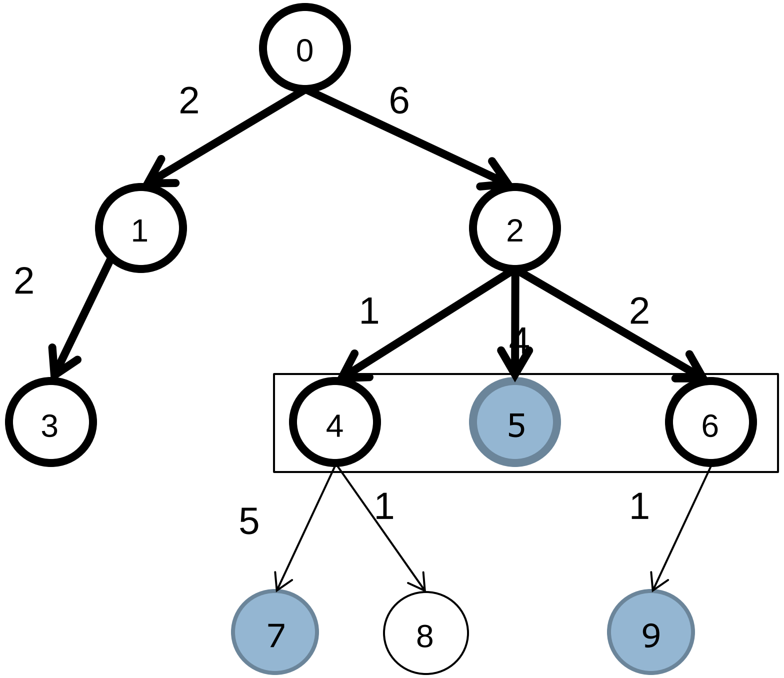
Algoritmos de Búsqueda
Ejemplo simple de BFS



frontier = (0)
frontier = (1,2)
frontier = (3,2)
frontier = (2)

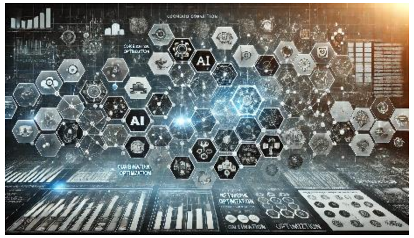
Algoritmos de Búsqueda
Ejemplo simple de BFS

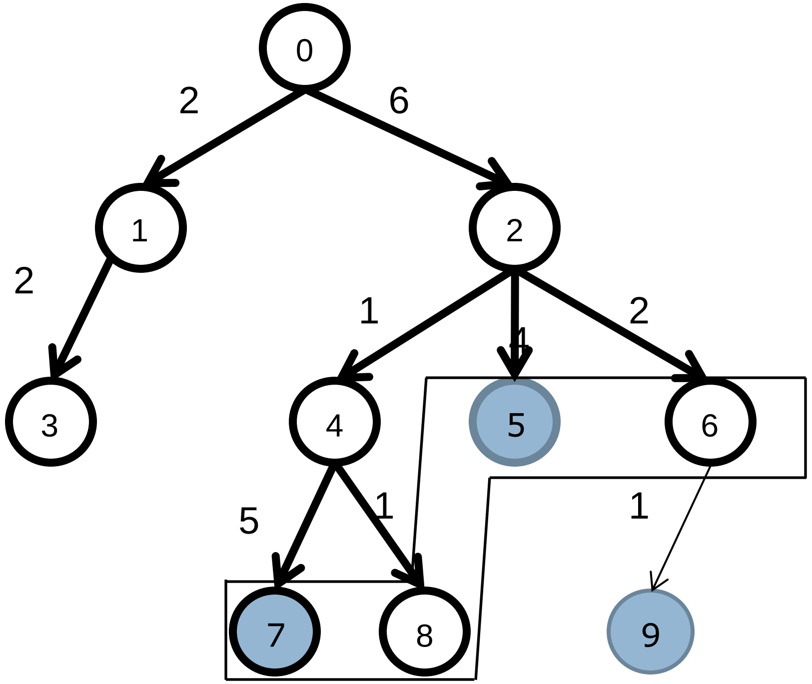




frontier = (0)
frontier = (1,2)
frontier = (3,2)
frontier = (2)
frontier = (4,5,6) **5 es solución pero no está el primero**

Algoritmos de Búsqueda
Ejemplo simple de BFS

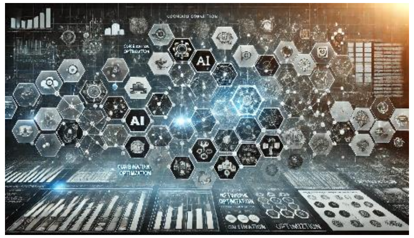


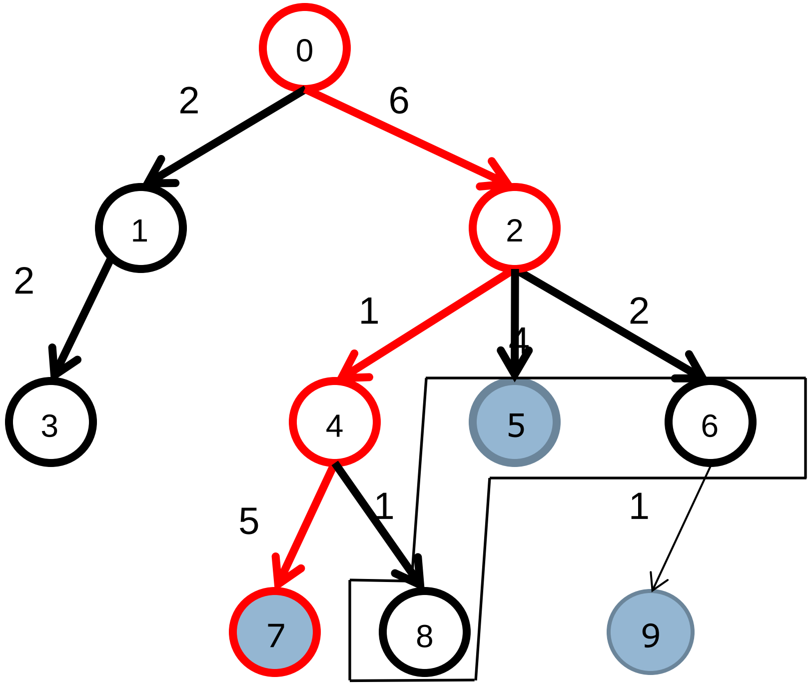


frontier = (0)
frontier = (1,2)
frontier = (3,2)
frontier = (2)
frontier = (4,5,6) 5 es solución pero no está el primero
frontier = (7,8,5,6)

Intelligent Scheduling and Optimization

Algoritmos de Búsqueda
Ejemplo simple de BFS



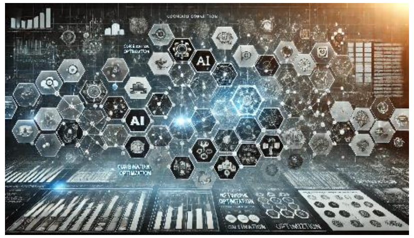


frontier = (0)
frontier = (1,2)
frontier = (3,2)
frontier = (2)
frontier = (4,5,6) 5 es solución pero no está el primero
frontier = (7,8,5,6)
frontier = (8,5,6)

Solución: 0-2-4-7
Coste: 6+1+5 = 11 (no óptima)

Intelligent Scheduling
and Optimization

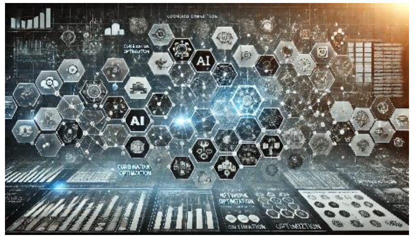
Algoritmos de Búsqueda
Casos particulares de “búsqueda a ciegas”

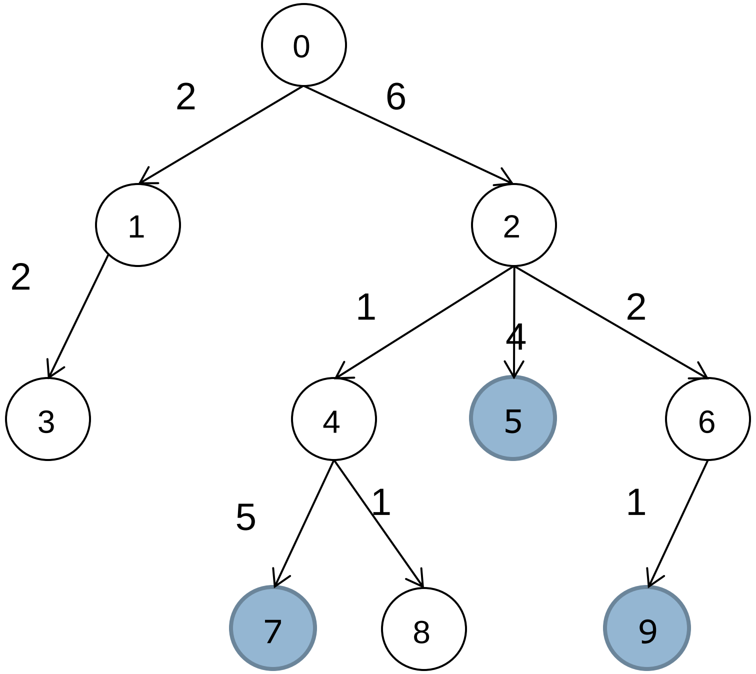


- Búsqueda en Profundidad
  - Insertar *child* al principio de *frontier* (f(n) = 1 / profundidad(n))
  - No es completo si hay ramas infinitas
- Búsqueda en Anchura
  - Insertar *child* al final de *frontier* (f(n) = profundidad(n))
  - Es una estrategia completa
- Coste Uniforme
  - Insertar *child* de forma ordenada en *frontier* según el coste al inicial (f(n) = n.PATH-COST)
  - Es una estrategia admisible

Algoritmos de Búsqueda

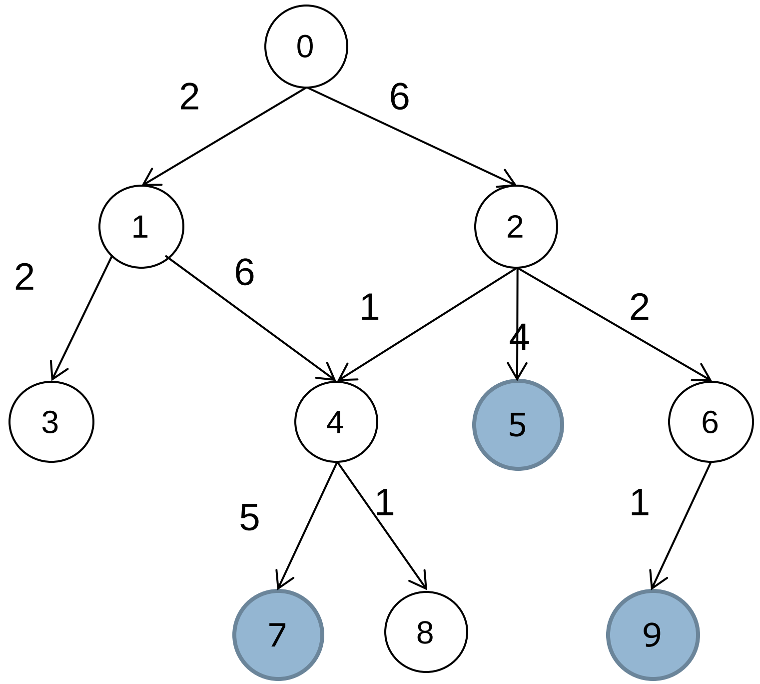
Resumen de las tres formas de búsqueda a ciegas





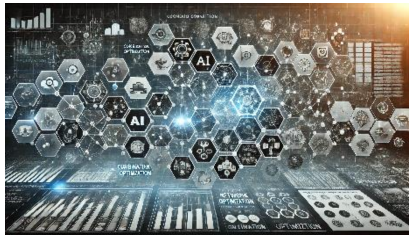
- **Búsqueda en Profundidad**
  - f(n) = 1 / profundidad(n)
  - Nodos expandidos: 0, 1, 3, 2, 4, 7
- **Búsqueda en Anchura**
  - f(n) = profundidad(n)
  - Nodos expandidos: 0, 1, 2, 3, 4, 5
- **Coste Uniforme**
  - f(n) = n.PATH-COST (coste inicial a n)
  - Nodos expandidos: 0, 1, 3, 2, 4, 6, 8, 9

Algoritmos de Búsqueda
Ejemplo menos simple de BFS

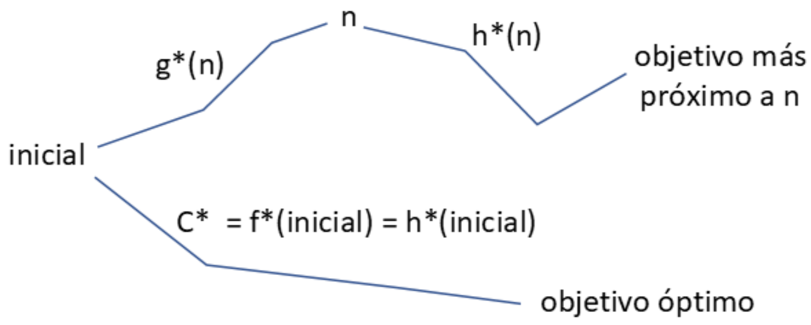


El Algoritmo A*

[Hart, P., Nilsson, N., Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans. Syst. Science and Cybernetics, SSC-4(2):100-107, 1968.]



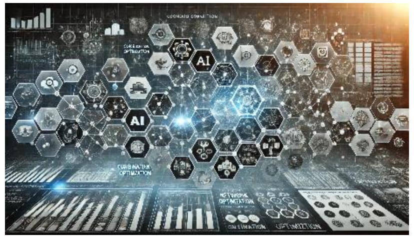
- Dado un espacio de búsqueda, definimos
  - $g^*(n)$ es el coste mínimo desde el inicial a n
  - $h^*(n)$ es el coste mínimo de n a los objetivos
  - $f^*(n) = g^*(n) + h^*(n)$, es el coste mínimo desde e inicial a los objetivos condicionado a pasar por n
  - $C^* = f^*(inicial) = h^*(inicial)$, es el coste de la solución óptima



- A* = BFS con f(n) = g(n) + h(n)
  - $f(n)$ es una estimación de $f^*(n)$
  - $g(n)$ = coste de inicial a *n*.STATE registrado en el Nodo *n*; es una variable de A* (*n*.PATH-COST)
  - $h(n)$ = estimación positiva de $h^*(n)$, con $h(n) = 0$, si n es objetivo. Es el HEURÍSTICO y se debe definir utilizando información sobre el problema, en particular sobre el estado *n*.STATE, consideraremos que $h(n) = h$(problema, *n*.STATE)

Aplicación de A*

Cálculo de rutas



$h_{SLD}(n)$ = distancia en línea recta, desde la ciudad de n a *Bucarest*

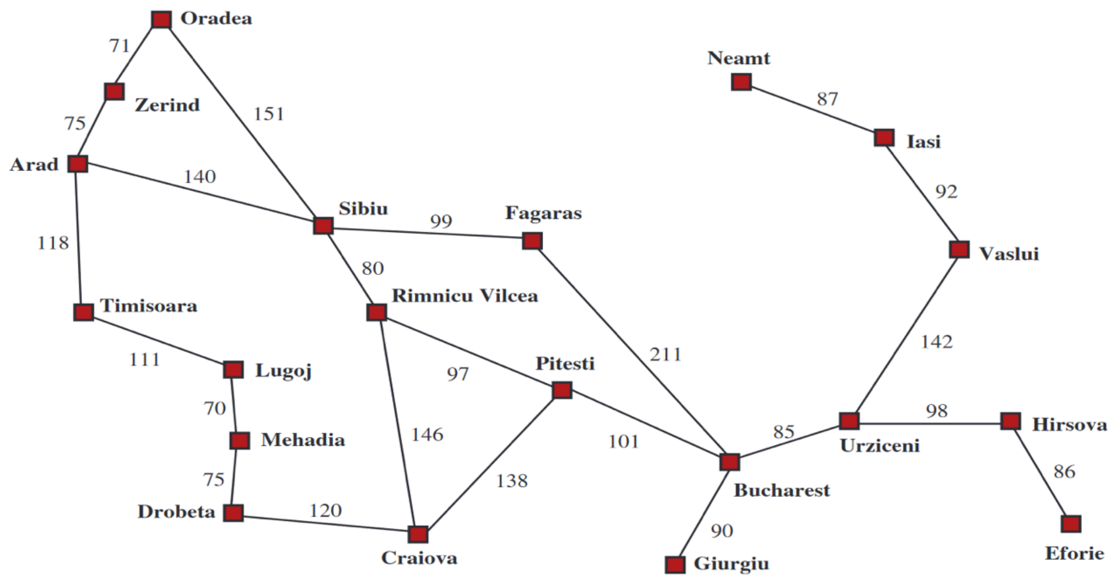
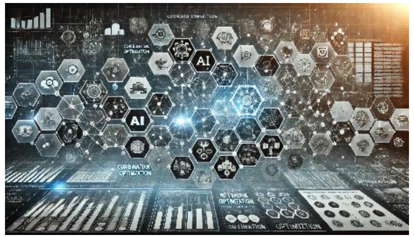


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Aplicación de A*
El 8-puzzle



- Diseño de heurísticos
  - En un estado podemos estimar el coste de llegar a la solución como el número de fichas que no están en la posición que les corresponde en el objetivo

h1(

2	8	3
1		4
7	6	5

) = 3

Objetivo

1	2	3
8		4
7	6	5

  - O mejor aún, como la suma de las distancias ortogonales de la posición actual de cada ficha a su posición en el objetivo

h2(

2	8	3
1		4
7	6	5

) = 1+1+2 = 4

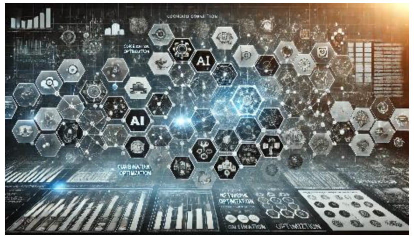
Objetivo

1	2	3
8		4
7	6	5

  - En este ejemplo tenemos $h1(s) < h2(s) = h^*(s) = 4$

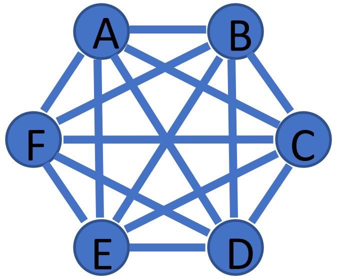
Intelligent Scheduling and Optimization

Aplicación de A*
El viajante de comercio



- Diseño de heurísticos
  - En un estado en el que quedan por recorrer x arcos, podemos calcular la suma de los x arcos más cortos del grafo residual

    h1(**(A{B}C)**) = 5 + 15 + 17 + 20 = 57



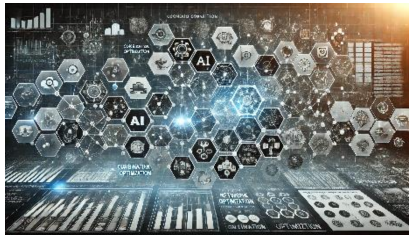
  - O bien, para cada ciudad por abandonar (alcanzar) sumar el coste del arco más corto que la toca

    h21(**(A{B}C)**) = 5 + 5 + 18 + 17 = 45

    h22(**(A{B}C)**) = 5 + 18 + 17 + 15 = 55

	A	B	C	D	E	F
A		21	[illegible]	15	113	92
~~B~~			~~7~~	~~32~~	~~25~~	~~9~~
C				5	18	20
D					180	39
E						17

Otros algoritmos de búsqueda



- A* es normalmente la mejor opción para grafos con muchos caminos alternativos entre pares de nodos
  - Ejemplos: el 8-puzzle, cálculos de rutas óptimas, planes de actuación, ….
- Si el tamaño del problema es muy grande hay que renunciar a la admisibilidad
  - Una opción son los algoritmos ε-*admisibles*
  - Ejemplo: Ponderación Estática (PEA*)

$$f(n) = g(n) + (1+\varepsilon)h(n),\ \varepsilon > 0$$

- Hay otras opciones que pueden ser más eficientes, particularmente cuando el espacio de búsqueda es un árbol
  - Algoritmos de Ramificación y Poda (*B&B*, Branch and Bound)
    - Combinan A* con búsqueda en árboles con cálculos de cotas superiores (con un algoritmo voraz). Si *f*(*n*) >= *cota_superior*, se descarta *n*
  - DF (Depth First) o backtracking parcialmente informado
    - Se ordenan los estados sucesores (o las reglas en backtracking) con un heurístico
  - Iterative Deepening A* (IDA*)
    - Parecido a DF iterativa, pero los sucesivos límites de profundidad se fijan con valores de f() de los nodos que se descartan en la iteración anterior
  - . . . . .

Intelligent Scheduling and Optimization