Inteligencia Artificial

Búsqueda informada

Edgar Andrade, Ph.D.

Matemáticas Aplicadas y Ciencias de la computación

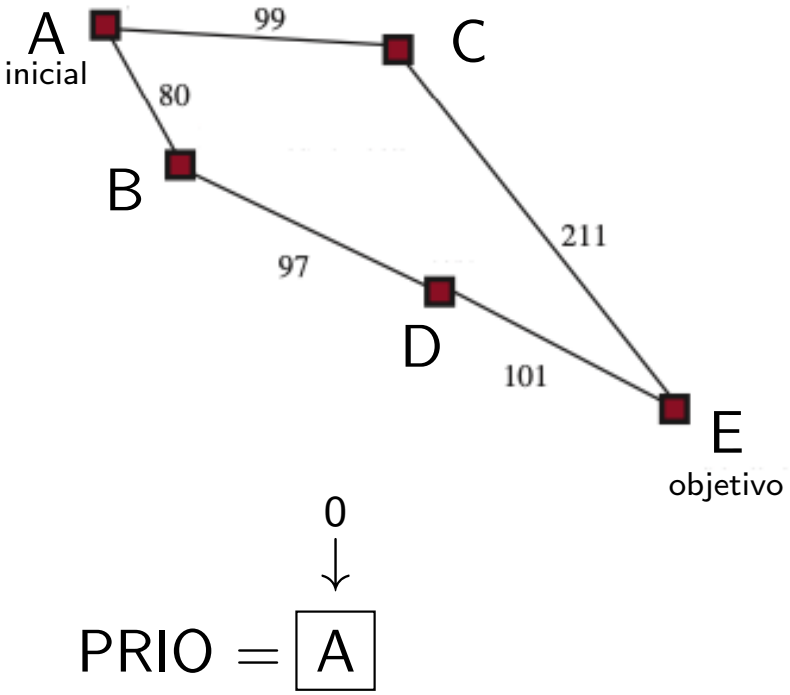
Última revisión: Agosto de 2023

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

Contenido

Algoritmo best first search

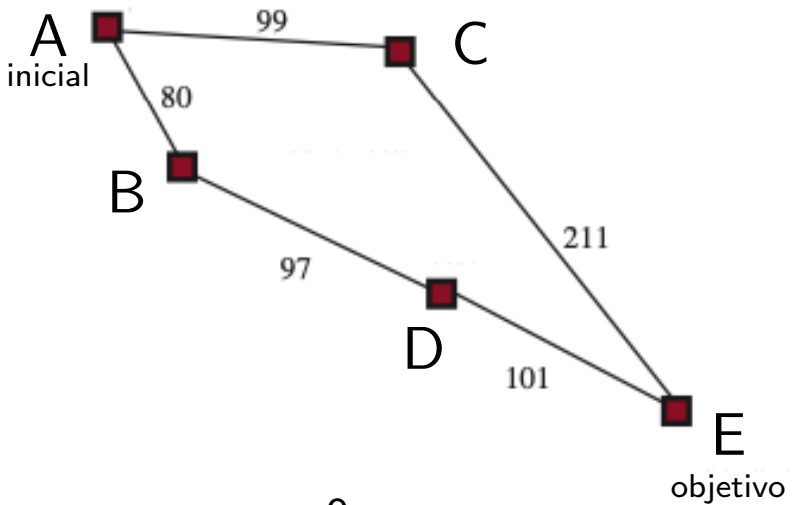


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



0

$$\text{PRIO} = \boxed{\text{A}}$$

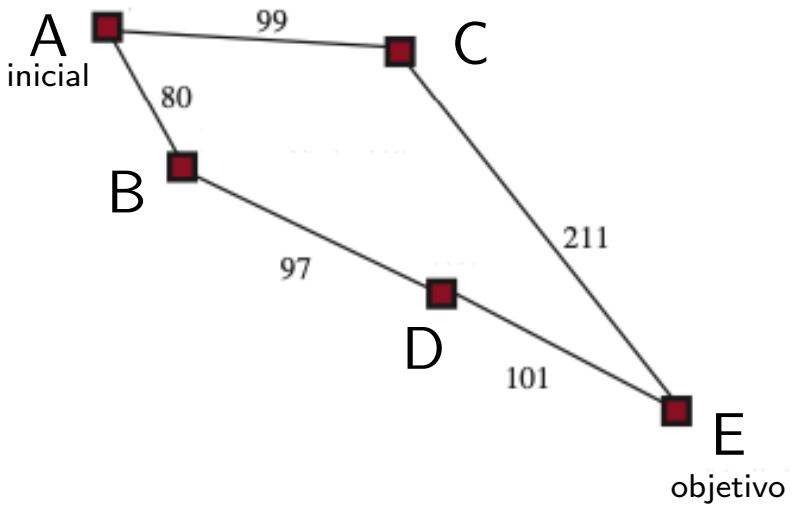
$$\textit{nodo} \leftarrow \text{POP(Frontera)}$$

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



$$\text{PRIO} =$$

$$nodo \leftarrow \text{A (test} = \text{false)}$$

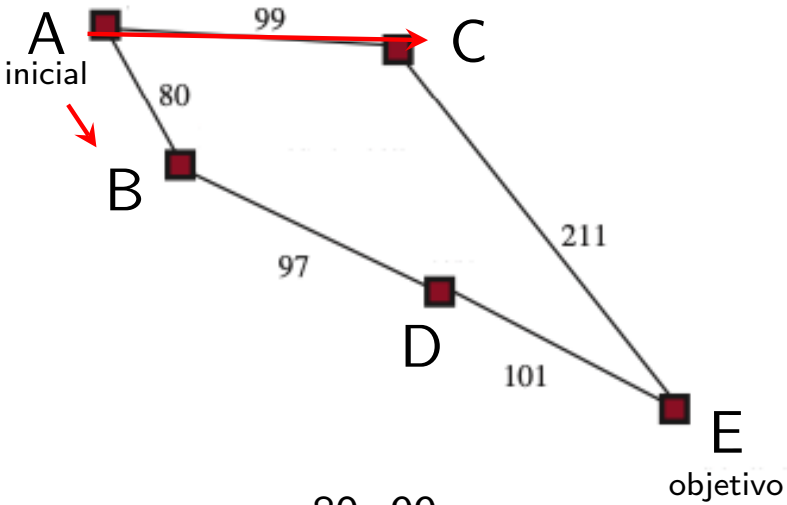
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

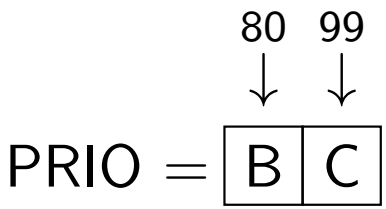
Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



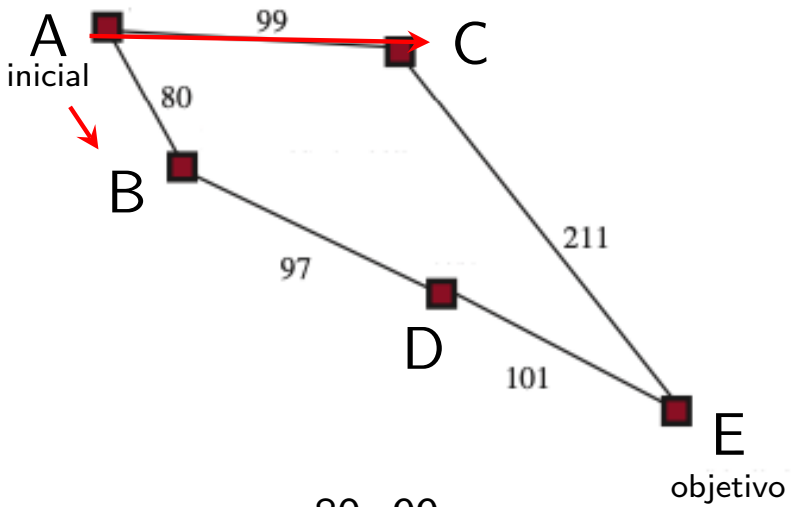


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



80 99

PRIO = | B | C |

nodo ← POP(Frontera)

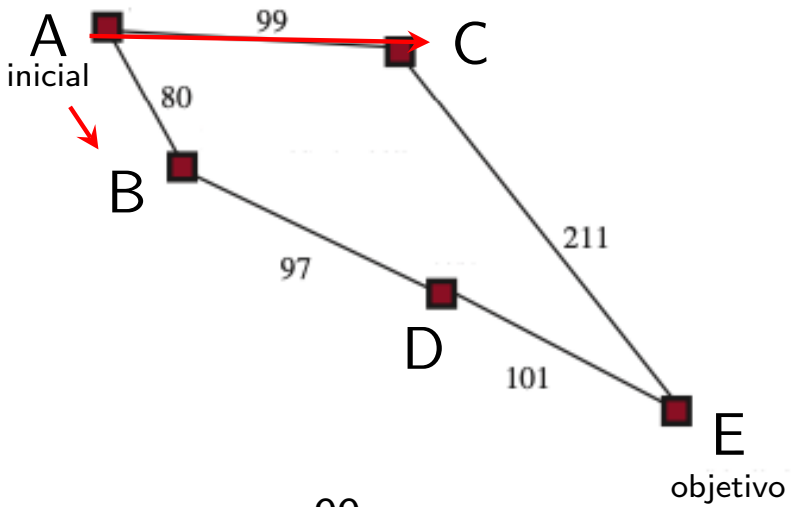
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



99
↓

PRIO = [C]

nodo ← B (test = false)

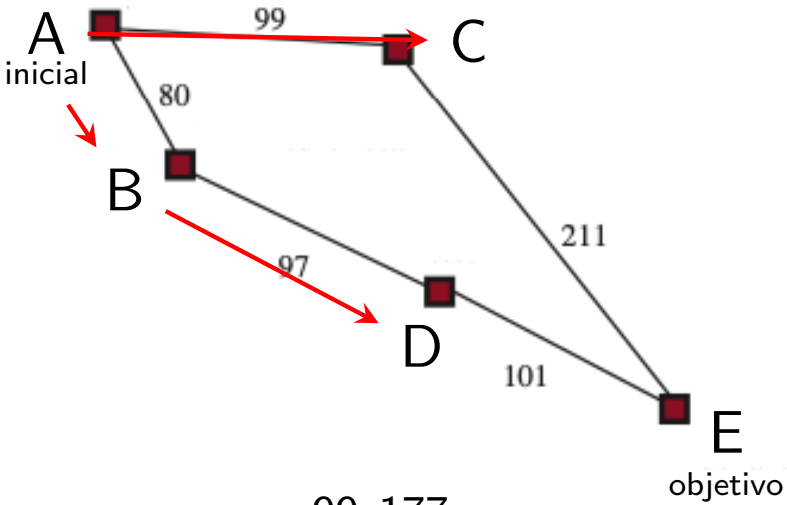
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

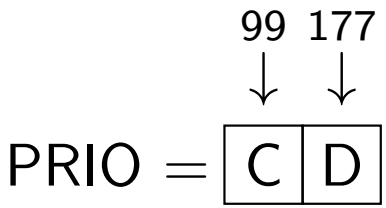
Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



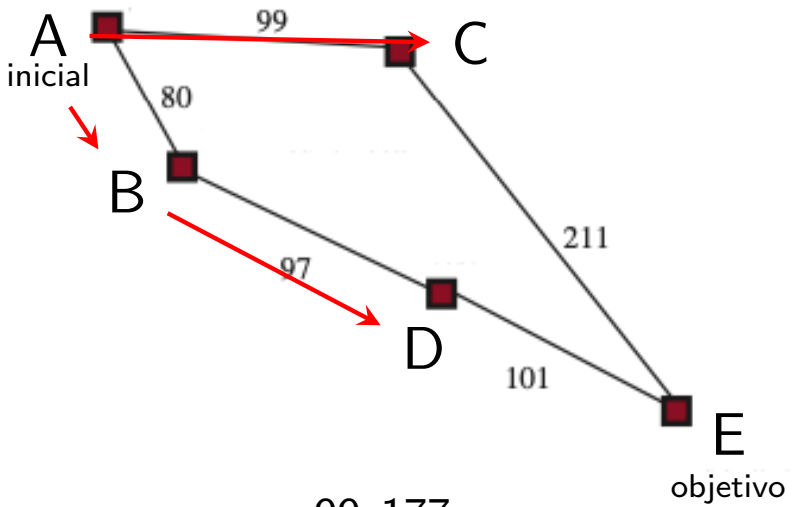


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



99 177

PRIO = | C | D |

nodo ← POP(Frontera)

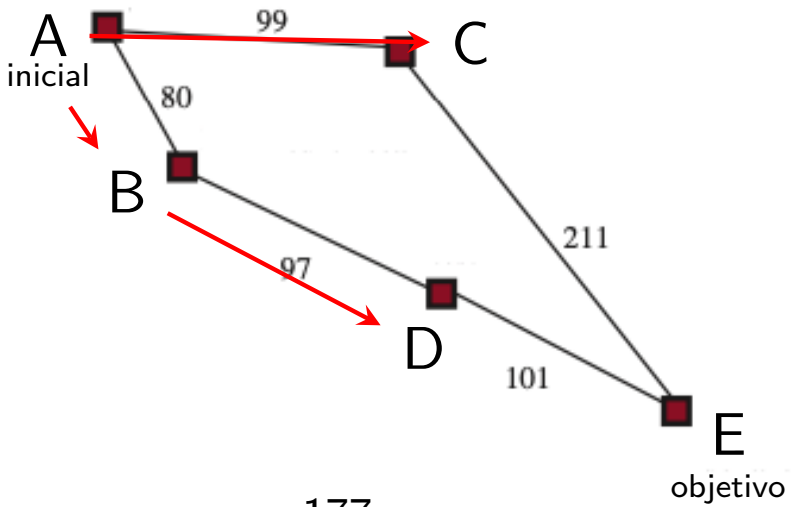
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



177

$$PRIO = \boxed{D}$$

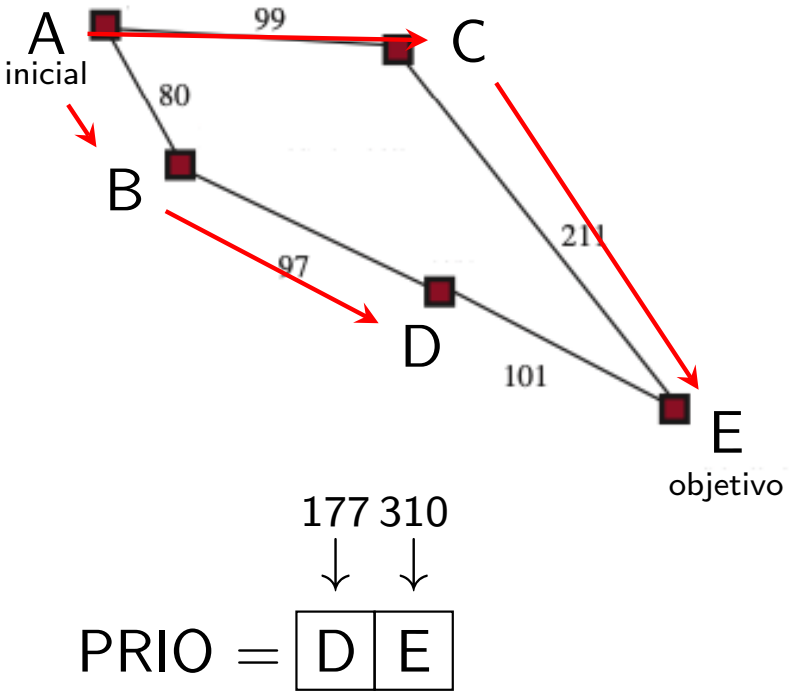
$$nodo \leftarrow C \ (test = false)$$

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search

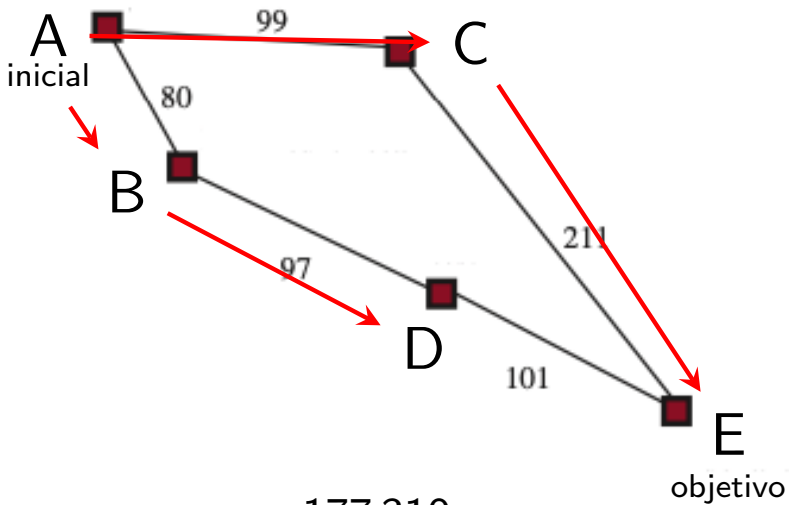


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



177 310

PRIO = | D | E |

nodo ← POP(Frontera)

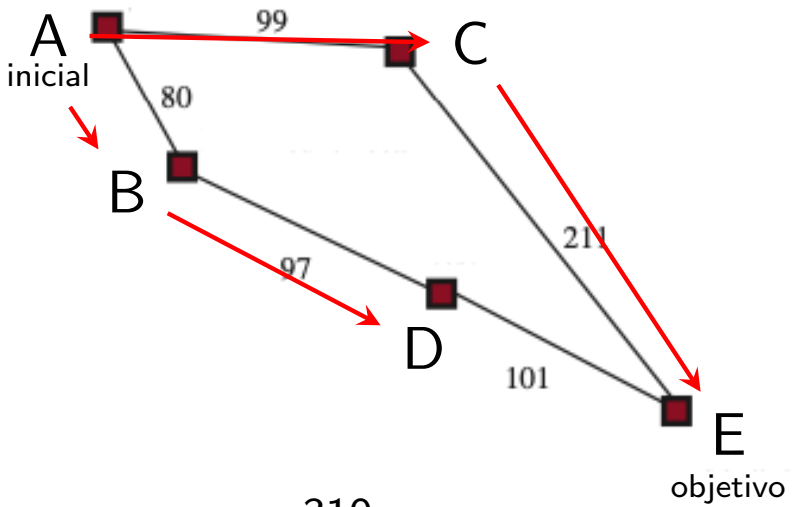
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



310

$\downarrow$

PRIO = $\boxed{\text{E}}$

nodo $\leftarrow$ D (test = false)

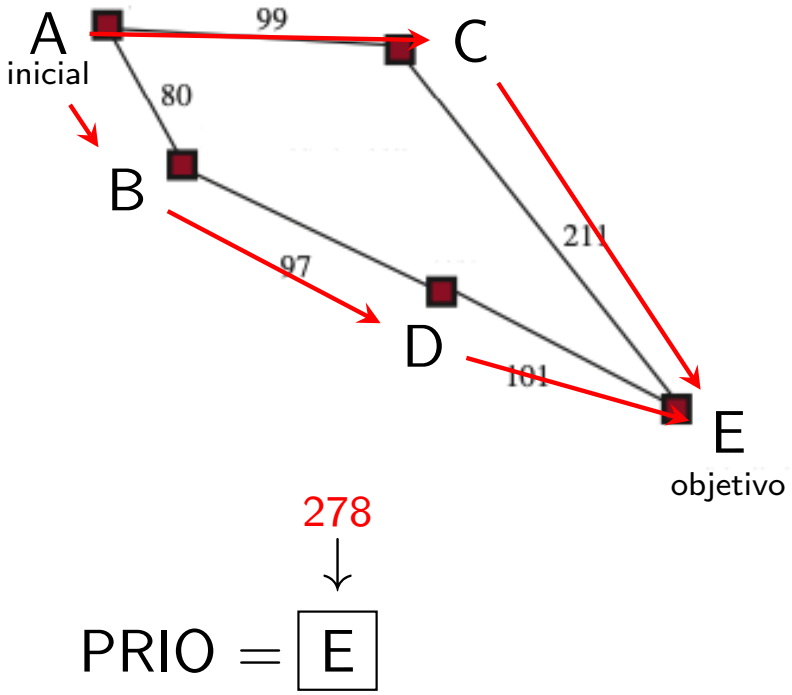
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search

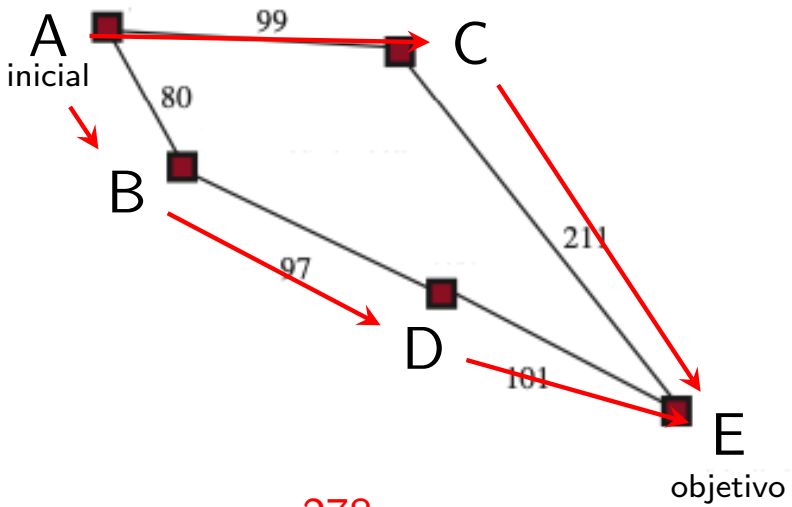


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



278

$$PRIO = \boxed{E}$$

$$nodo \leftarrow \text{POP}(\text{Frontera})$$

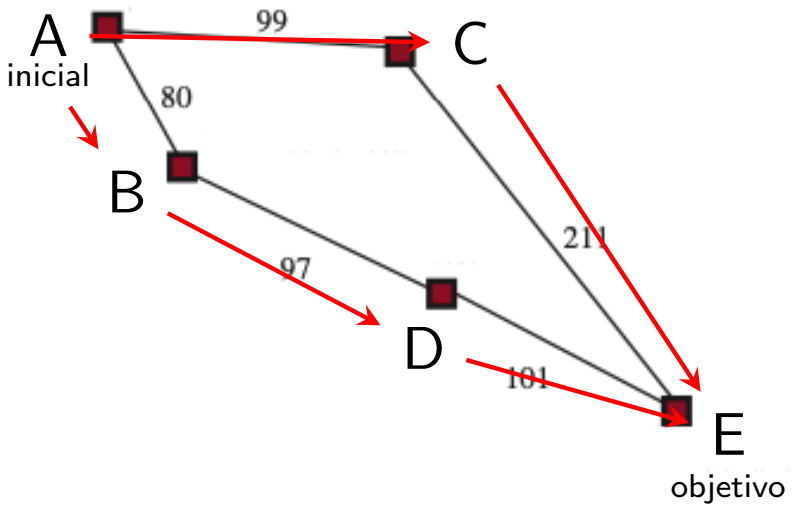
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



$$\text{PRIO} =$$

$$nodo \leftarrow \text{E (test} = \text{true)}$$

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Variaciones de Best First Search

Dependiendo de qué se use para medir el valor de un estado, se obtienen distintos algoritmos:

Variaciones de Best First Search

Dependiendo de qué se use para medir el valor de un estado, se obtienen distintos algoritmos:

- Si se usa el costo camino, se obtiene el algoritmo de Dijkstra.

Variaciones de Best First Search

Dependiendo de qué se use para medir el valor de un estado, se obtienen distintos algoritmos:

- Si se usa el costo camino, se obtiene el algoritmo de Dijkstra.
- Si se usa una heurística, se obtiene el algoritmo de búsqueda avara.

Variaciones de Best First Search

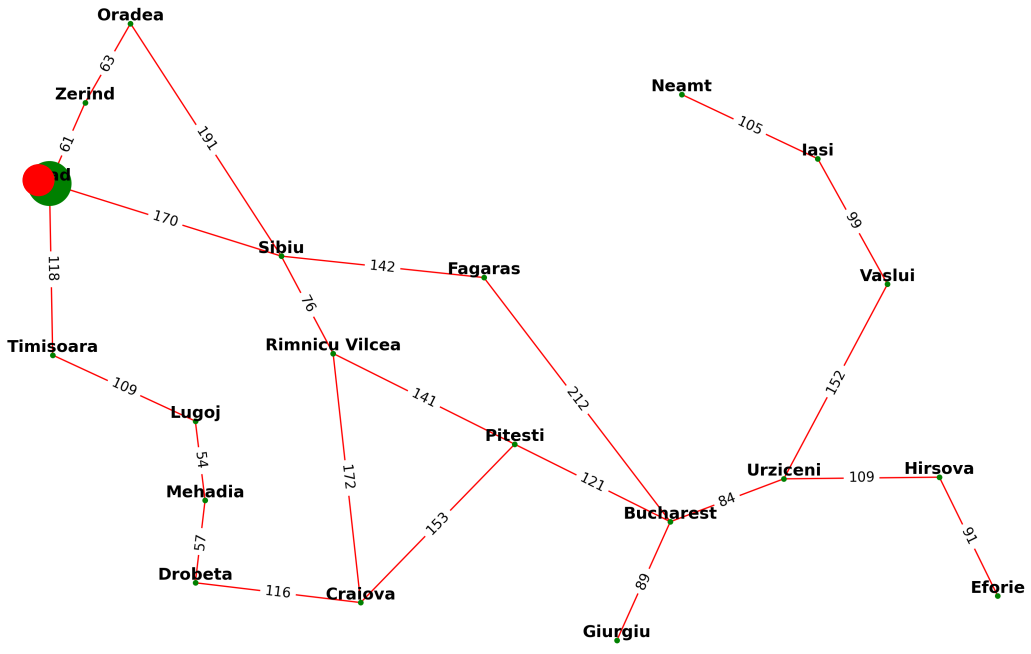
Dependiendo de qué se use para medir el valor de un estado, se obtienen distintos algoritmos:

- Si se usa el costo camino, se obtiene el algoritmo de Dijkstra.
- Si se usa una heurística, se obtiene el algoritmo de búsqueda avara.
- Si se combinan costo y heurística, se obtiene el algoritmo A^*.

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

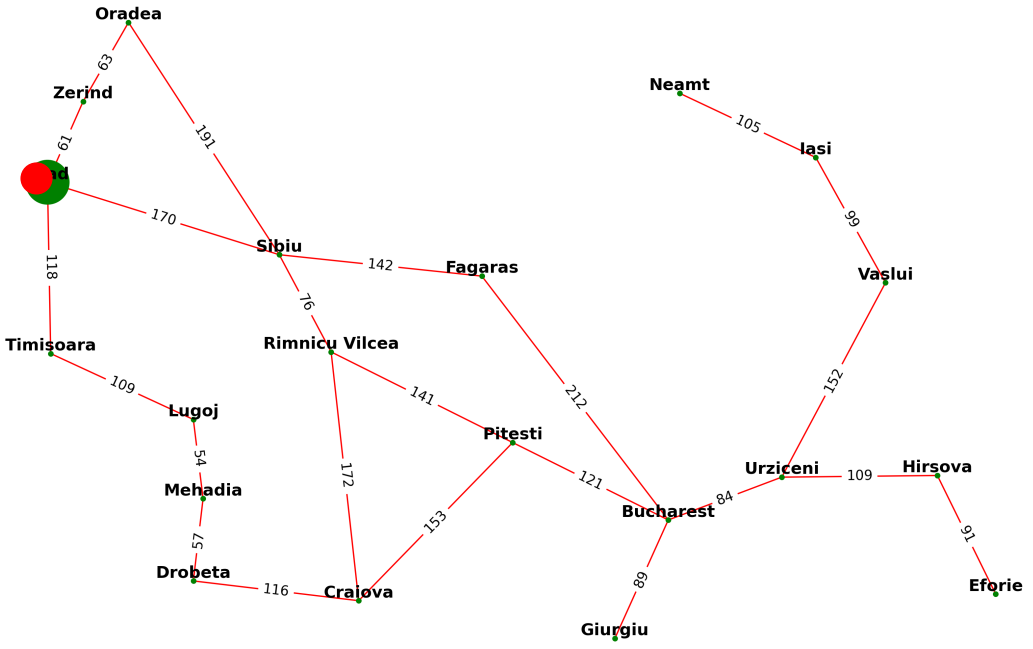
valor = costo_camino



Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

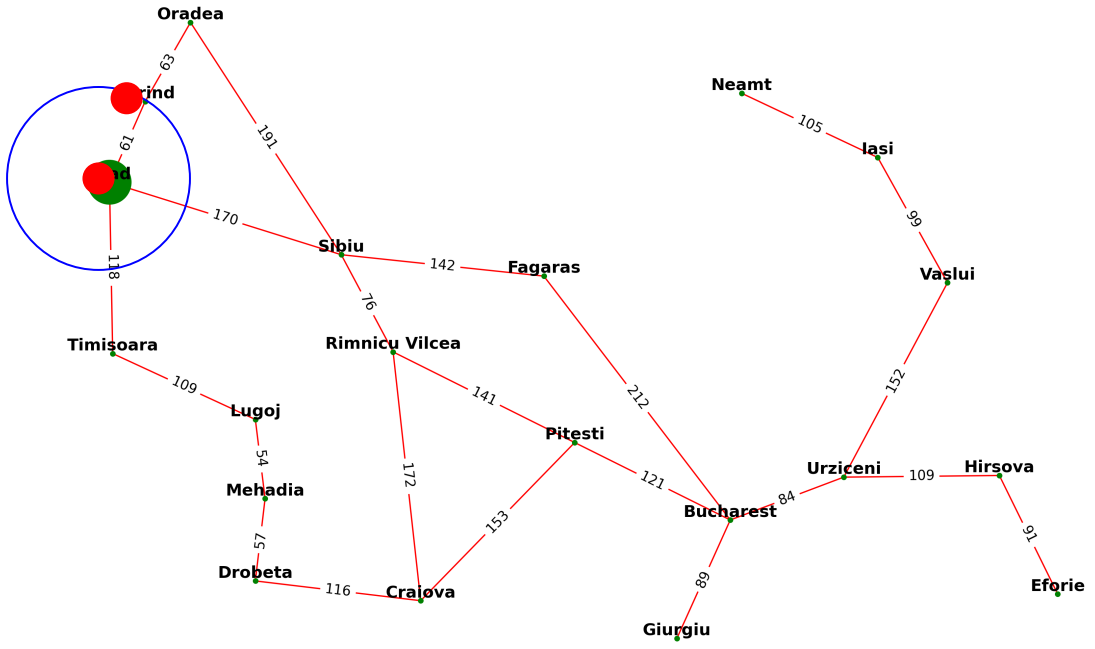


FRONTERA = $[(\text{Arad}_{\text{Zerind}}, 61), (\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Sibiu}}, 170)]$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

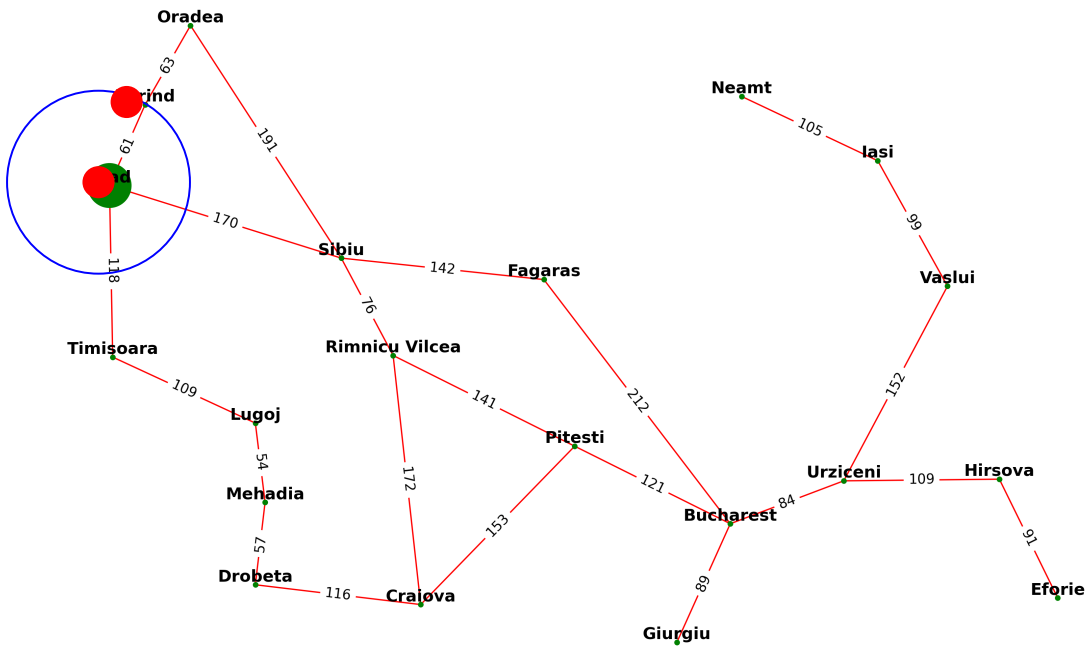


FRONTERA = $[(\text{Arad}_{\text{Zerind}}, 61), (\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Sibiu}}, 170)]$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

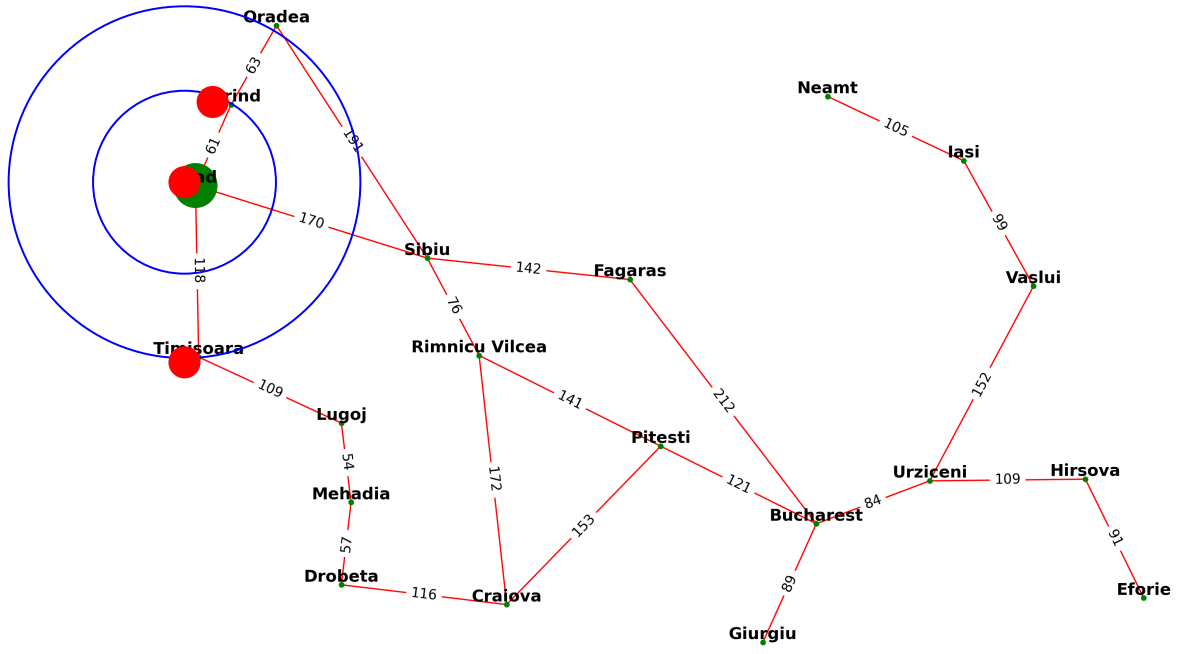


$$\text{FRONTERA} = [(\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Zerind}}^{\text{Oradea}}, 124), (\text{Arad}_{\text{Sibiu}}, 170)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

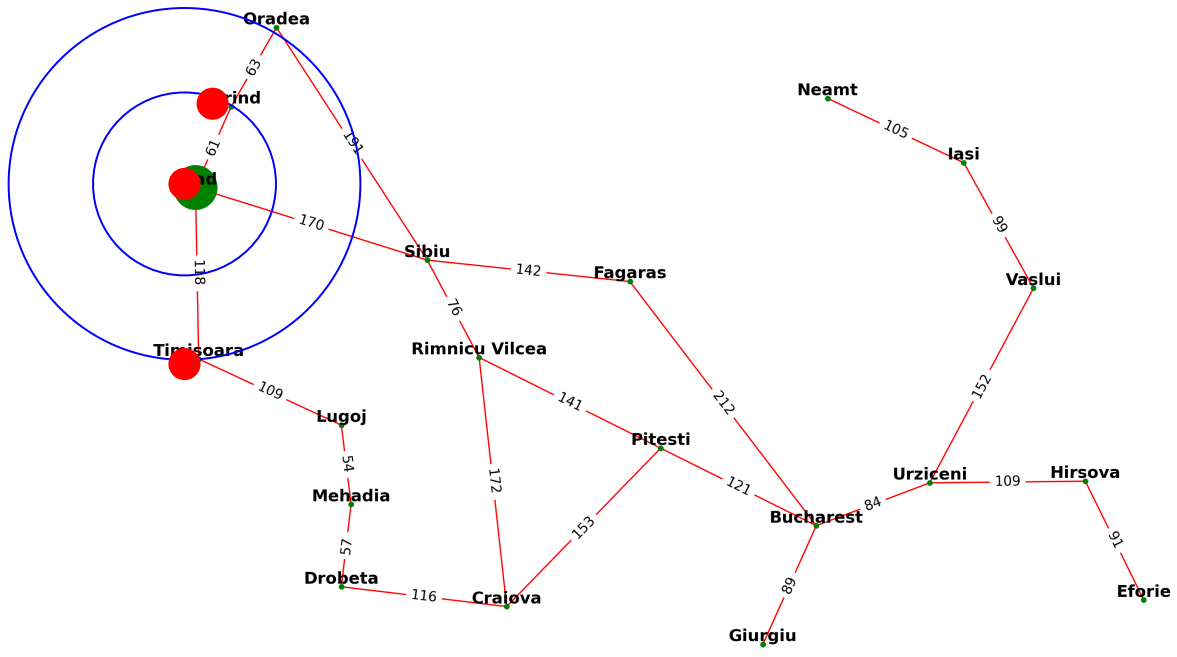


$$\text{FRONTERA} = [(\text{Arad Timisoara}, 118), (\text{Arad Zerind Oradea}, 124), (\text{Arad Sibiu}, 170)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

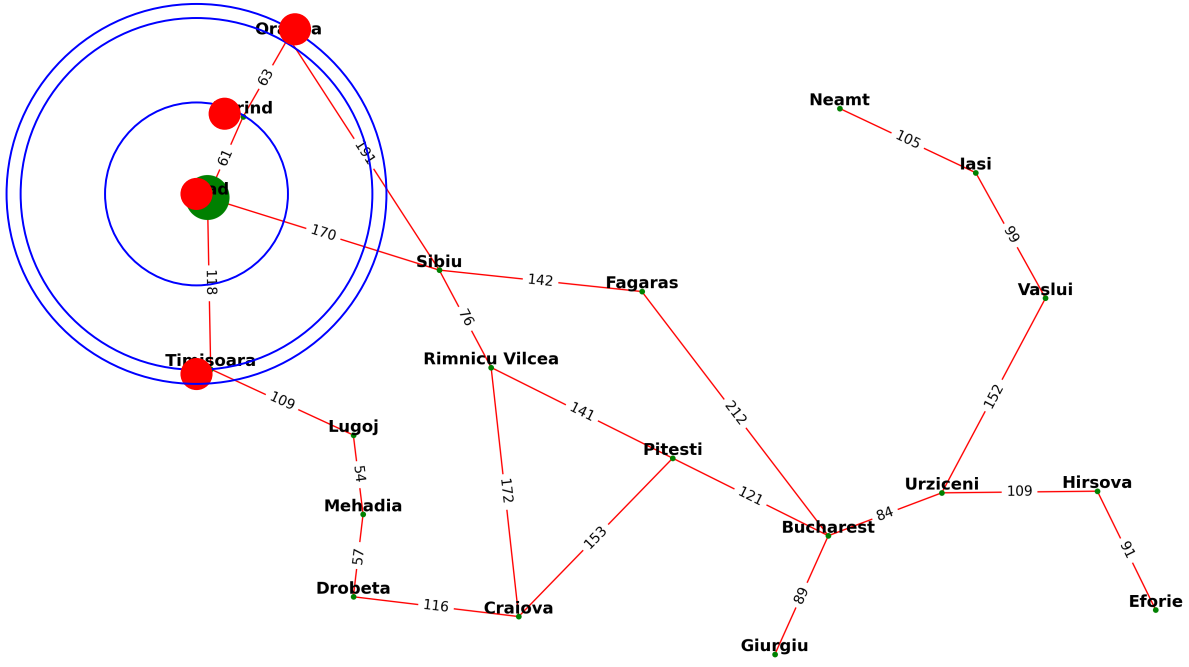


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\end{matrix}, 124), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

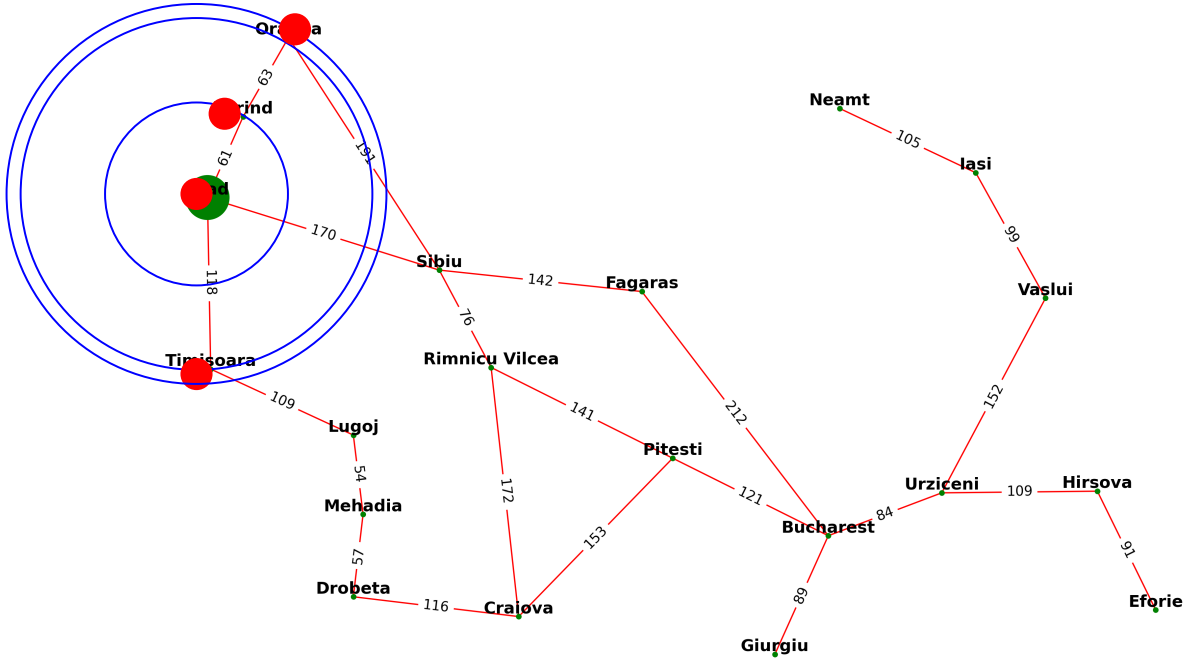


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\end{matrix}, 124), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

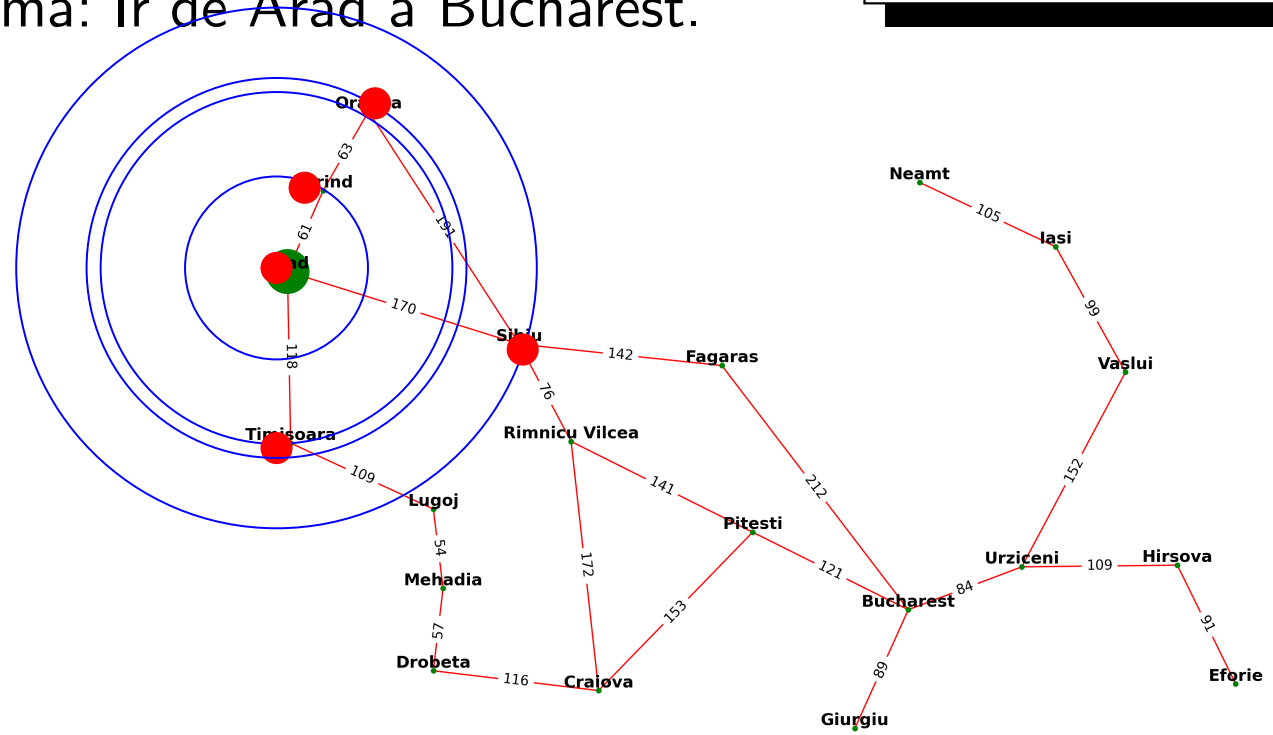


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227), (\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\\ \text{Sibiu}\end{matrix}, 315))]$$

Costo camino (Dijkstra)

valor = costo_camino

Problema: Ir de Arad a Bucharest.



$$\text{FRONTERA} = [(\text{Arad}\ \text{Sibiu},\ 170),\ (\text{Arad}\ \text{Timisoara}\ \text{Lugoj},\ 227),\ (\text{Arad}\ \text{Zerind}\ \text{Oradea}\ \text{Sibiu},\ 315))]$$

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Heurística para el viaje a Rumania

☞ Supongamos que vamos de Arad a Bucharest.

Heurística para el viaje a Rumania

☞ Supongamos que vamos de Arad a Bucharest.

☞ $h(\texttt{ciudad})$ = Distancia lineal desde `ciudad` hasta Bucharest.

Búsqueda avara

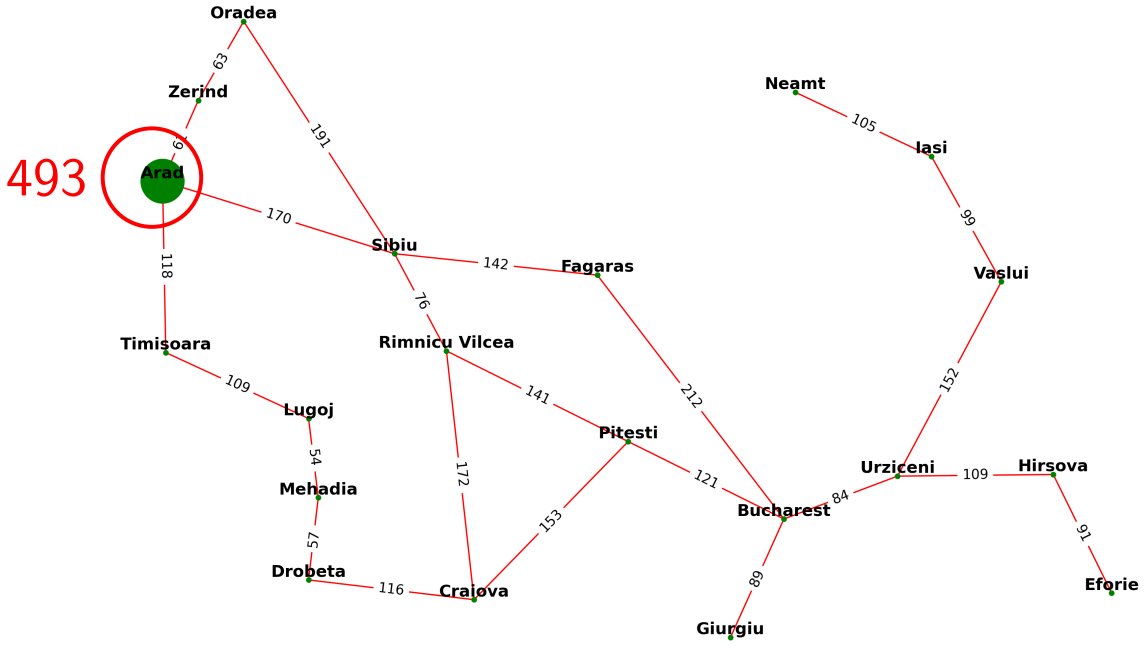
Problema: Ir de Arad a Bucharest.

valor = distancia a objetivo

Búsqueda avara

Problema: Ir de Arad a Bucharest.

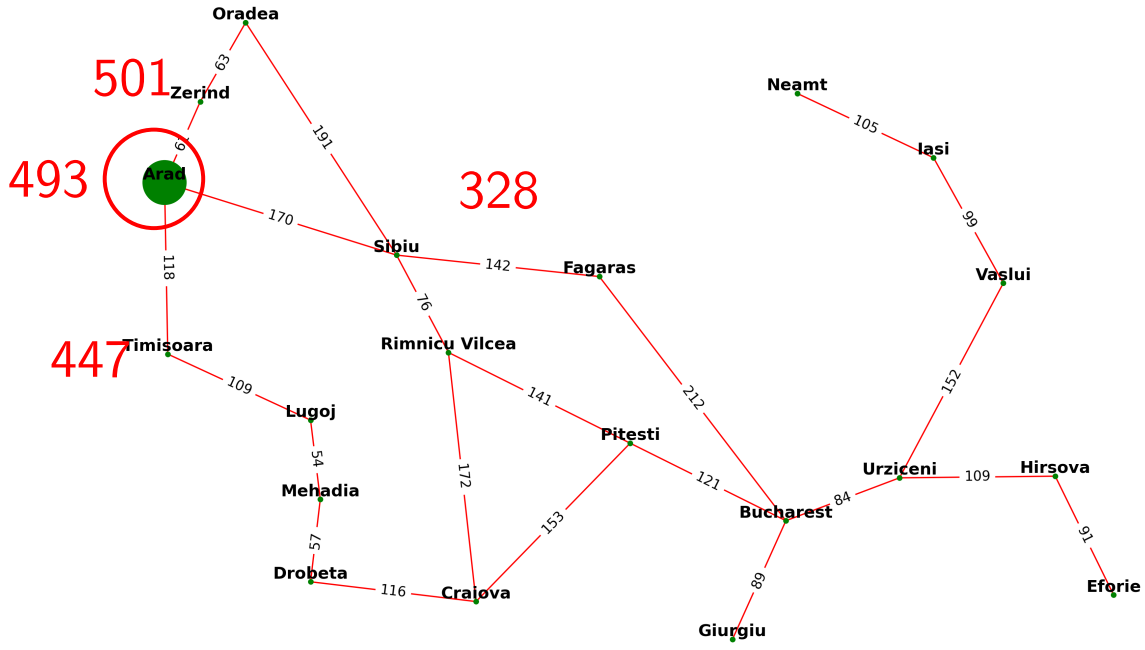
valor = distancia a objetivo



Búsqueda avara

Problema: Ir de Arad a Bucharest.

valor = distancia a objetivo

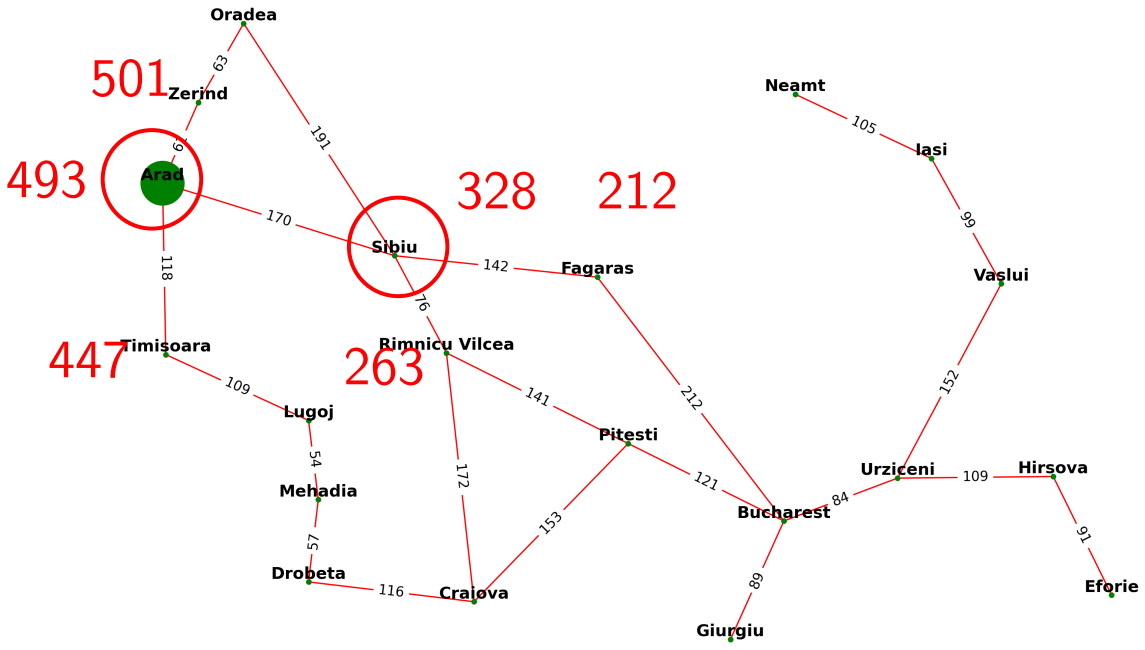


$$\text{FRONTERA} = [(\text{Arad}_{\text{Sibiu}}, 328), (\text{Arad}_{\text{Timisoara}}, 447), (\text{Arad}_{\text{Zerind}}, 501)]$$

Búsqueda avara

Problema: Ir de Arad a Bucharest.

valor = distancia a objetivo

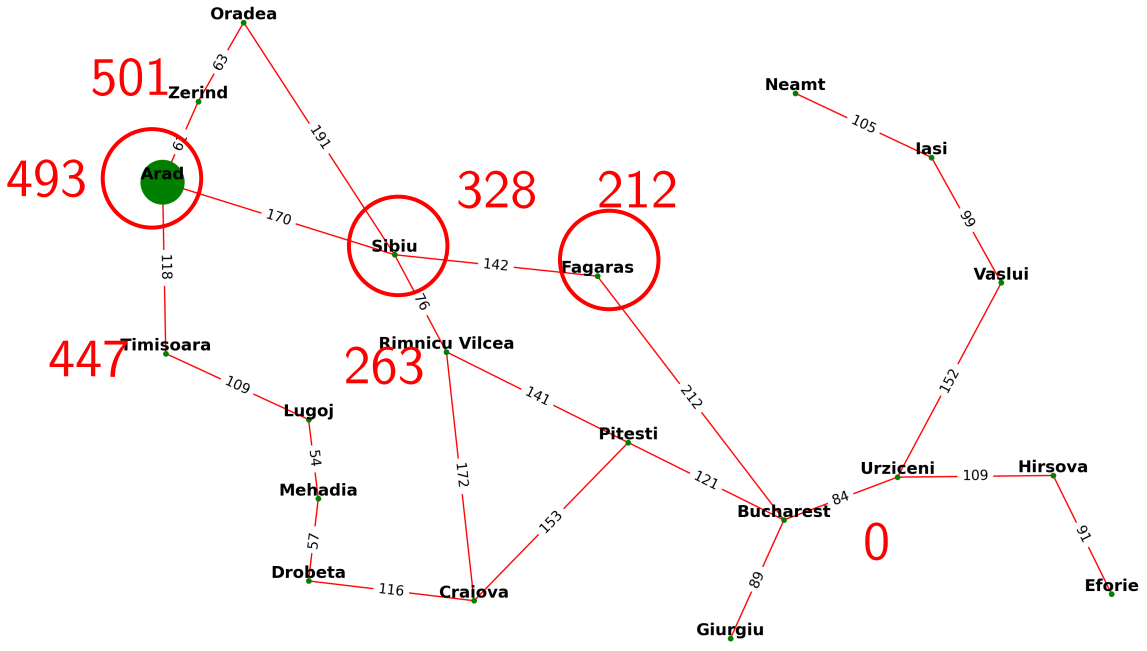


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Fagaras}\end{matrix}, 212), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Rimnicu Vilcea}\end{matrix}, 263), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\end{matrix}, 447), (\begin{matrix}\text{Arad}\\ \text{Zerind}\end{matrix}, 501)]$$

Búsqueda avara

Problema: Ir de Arad a Bucharest.

valor = distancia a objetivo

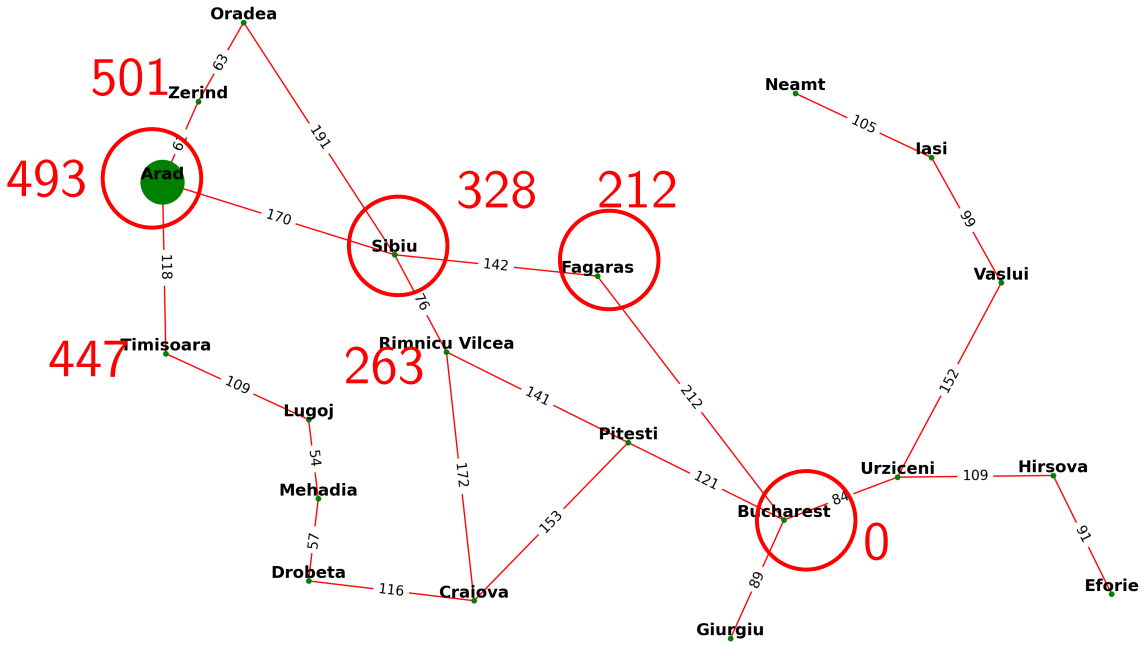


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Fagaras}\\ \text{Bucharest}\end{matrix}, 0), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Rimnicu Vilcea}\end{matrix}, 263), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\end{matrix}, 447), (\begin{matrix}\text{Arad}\\ \text{Zerind}\end{matrix}, 501)]$$

Búsqueda avara

Problema: Ir de Arad a Bucharest.

valor = distancia a objetivo



$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Fagaras}\\ \text{Bucharest}\end{matrix}, 0), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Rimnicu Vilcea}\end{matrix}, 263), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\end{matrix}, 447), (\begin{matrix}\text{Arad}\\ \text{Zerind}\end{matrix}, 501)]$$

Contenido

A*

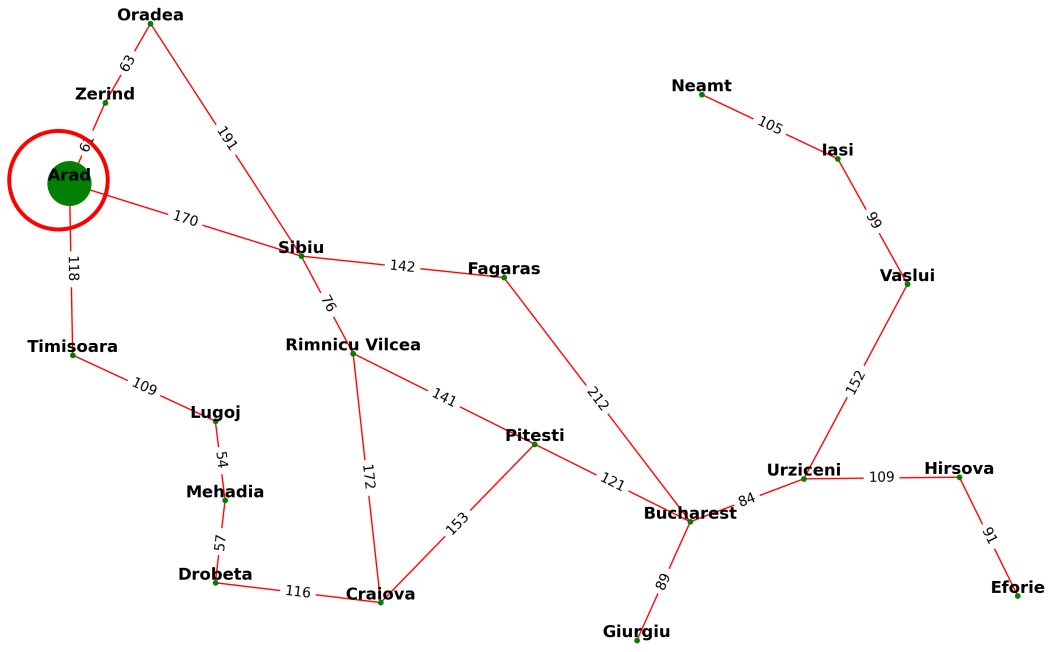
Usar tanto un costo camino como un estimado del costo del mejor camino hasta la solución (heurística).

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia

493 = 0 + 493

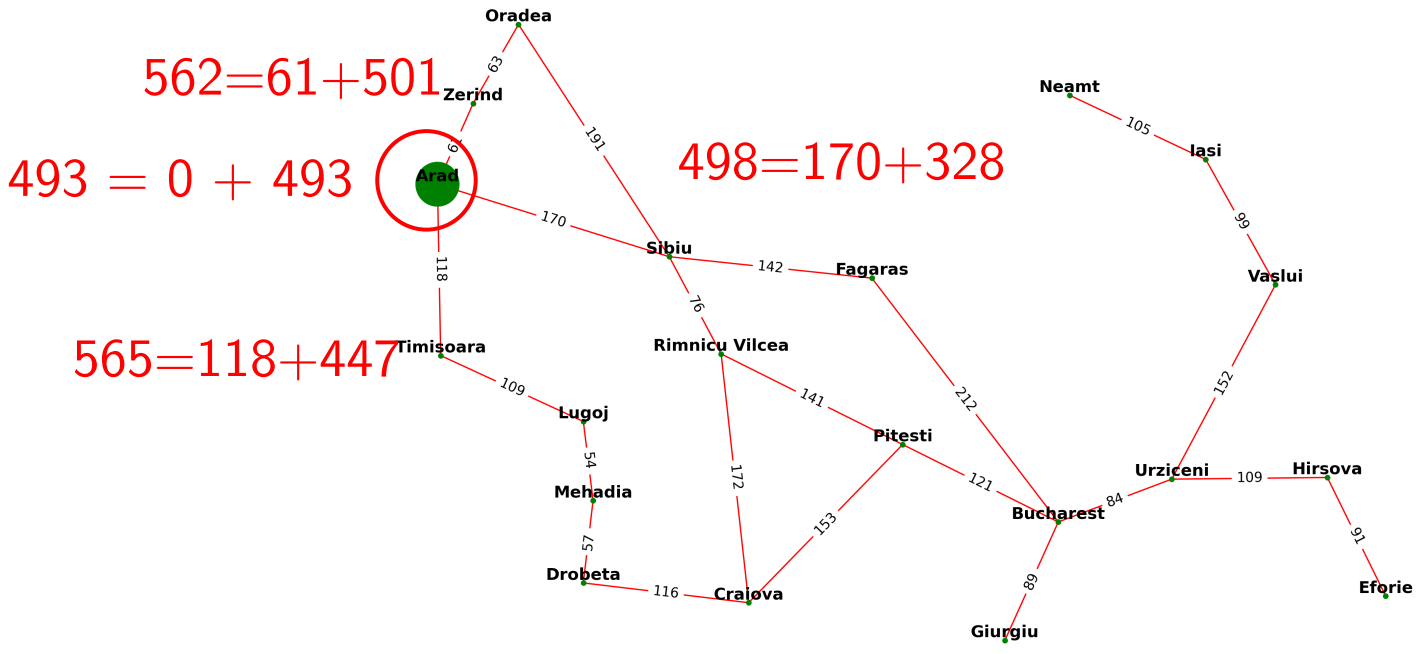


FRONTERA = [(Arad, 493)]

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia

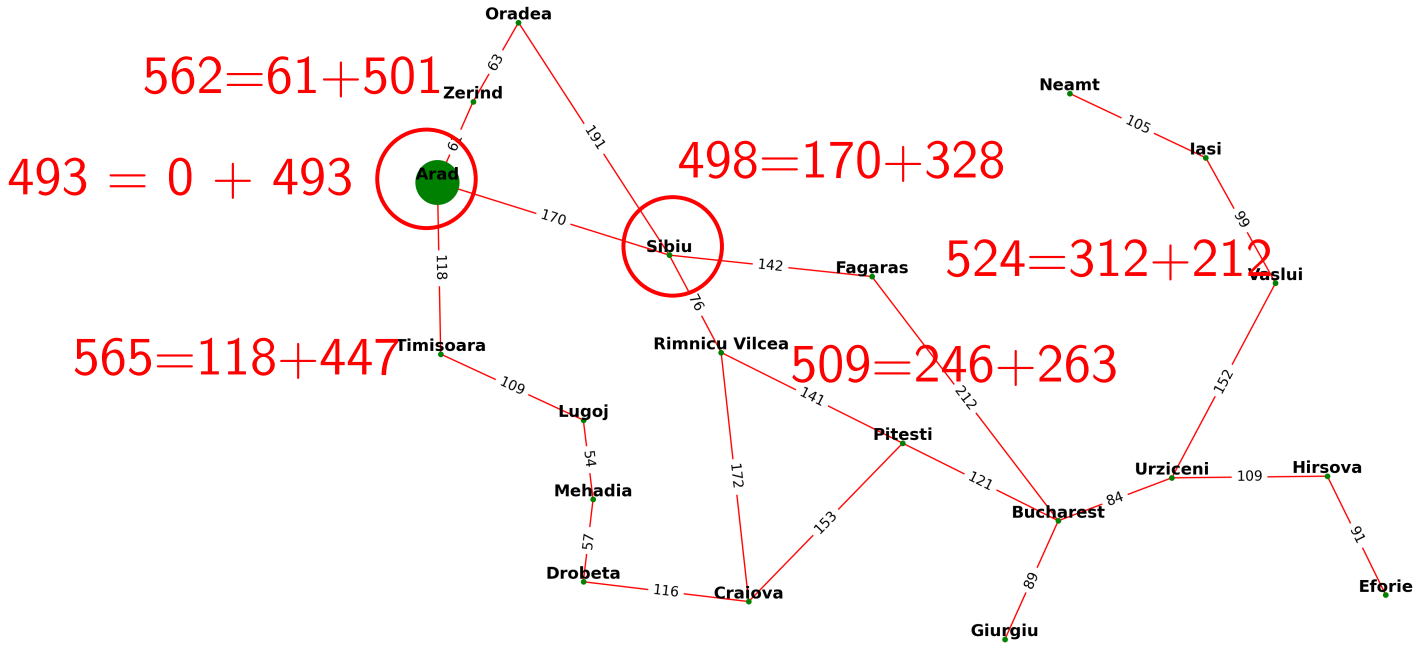


FRONTERA = $[(\underset{\text{Sibiu}}{\text{Arad}}, 498), (\underset{\text{Zerind}}{\text{Arad}}, 562), (\underset{\text{Timisoara}}{\text{Arad}}, 565)]$

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia

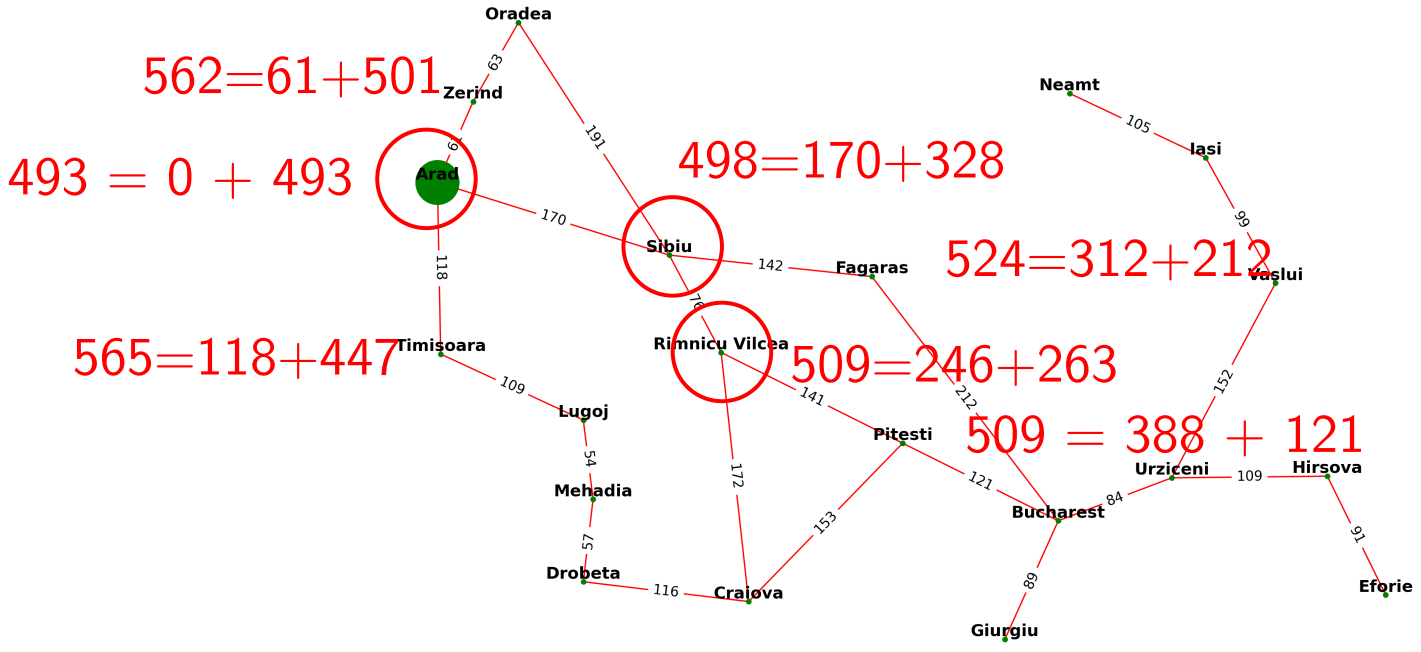


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\\text{Sibiu}\\\text{Rimnicu Vilcea}\end{matrix}, 509), (\begin{matrix}\text{Arad}\\\text{Sibiu}\\\text{Fargas}\end{matrix}, 524), (\begin{matrix}\text{Arad}\\\text{Zerind}\end{matrix}, 562), (\begin{matrix}\text{Arad}\\\text{Timisoara}\end{matrix}, 565)]$$

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia

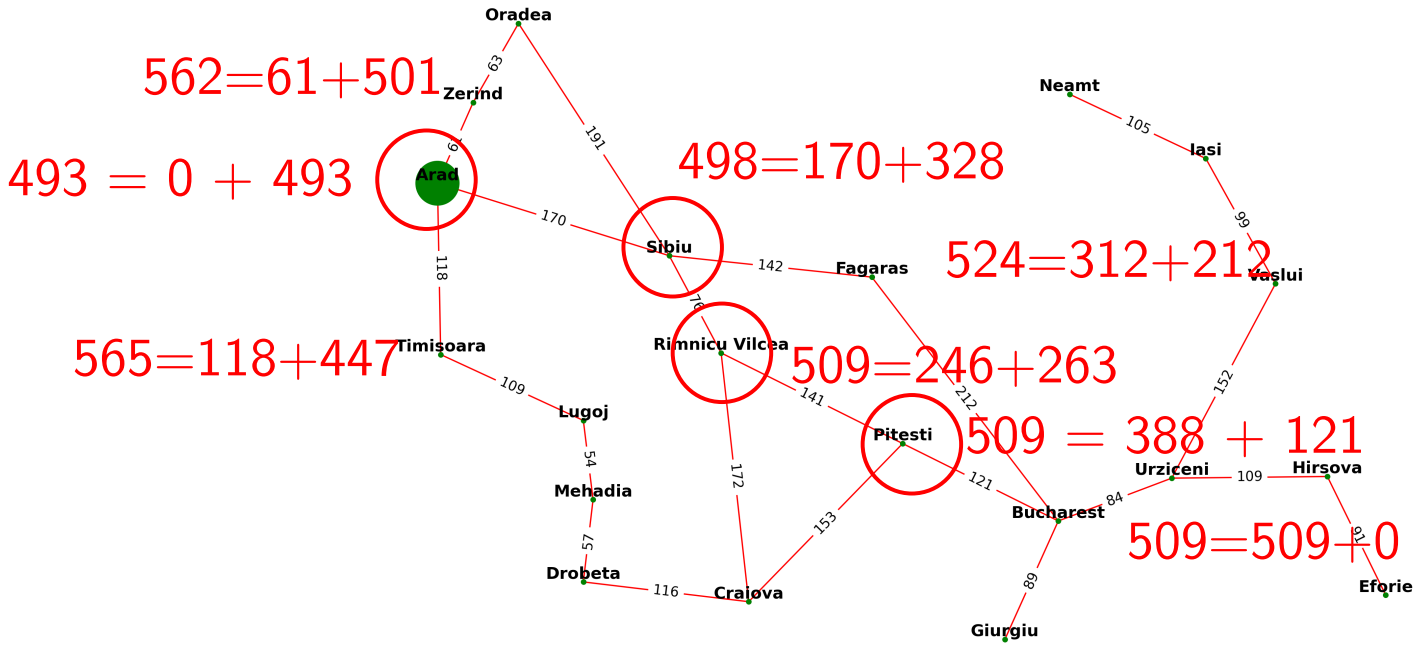


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Rimnicu Vilcea}\\ \text{Pitesti}\end{matrix}, 509), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Fargas}\end{matrix}, 524), (\begin{matrix}\text{Arad}\\ \text{Zerind}\end{matrix}, 562), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\end{matrix}, 565)]$$

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia

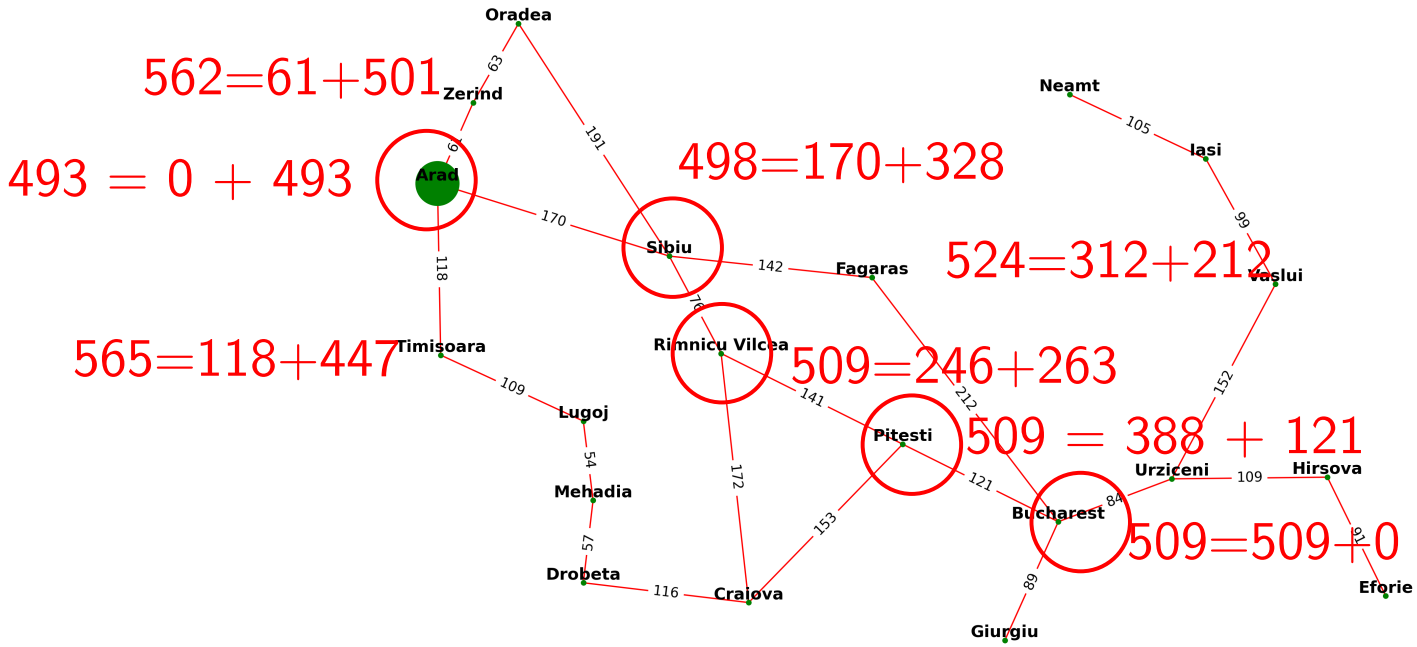


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Rimnicu Vilcea}\\ \text{Pitesti}\\ \text{Bucharest}\end{matrix}, 509), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\\ \text{Fargas}\end{matrix}, 524), (\begin{matrix}\text{Arad}\\ \text{Zerind}\end{matrix}, 562), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\end{matrix}, 565)]$$

A*

Problema: Ir de Arad a Bucharest.

valor = costo_camino + distancia



Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

El algoritmo más rápido es el Backtracking.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

Primero en Anchura y Búsqueda Avara dan la misma solución, pero es más rápido Primero en Anchura. ¿Por qué?

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

La lista prioritaria de la Búsqueda Avara es una gran carga computacional.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

Los algoritmos que dan la solución óptima son Dijkstra y A^*.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Tiempos en CPU para el viaje a Rumania

Caso ir de Arad a Bucharest.

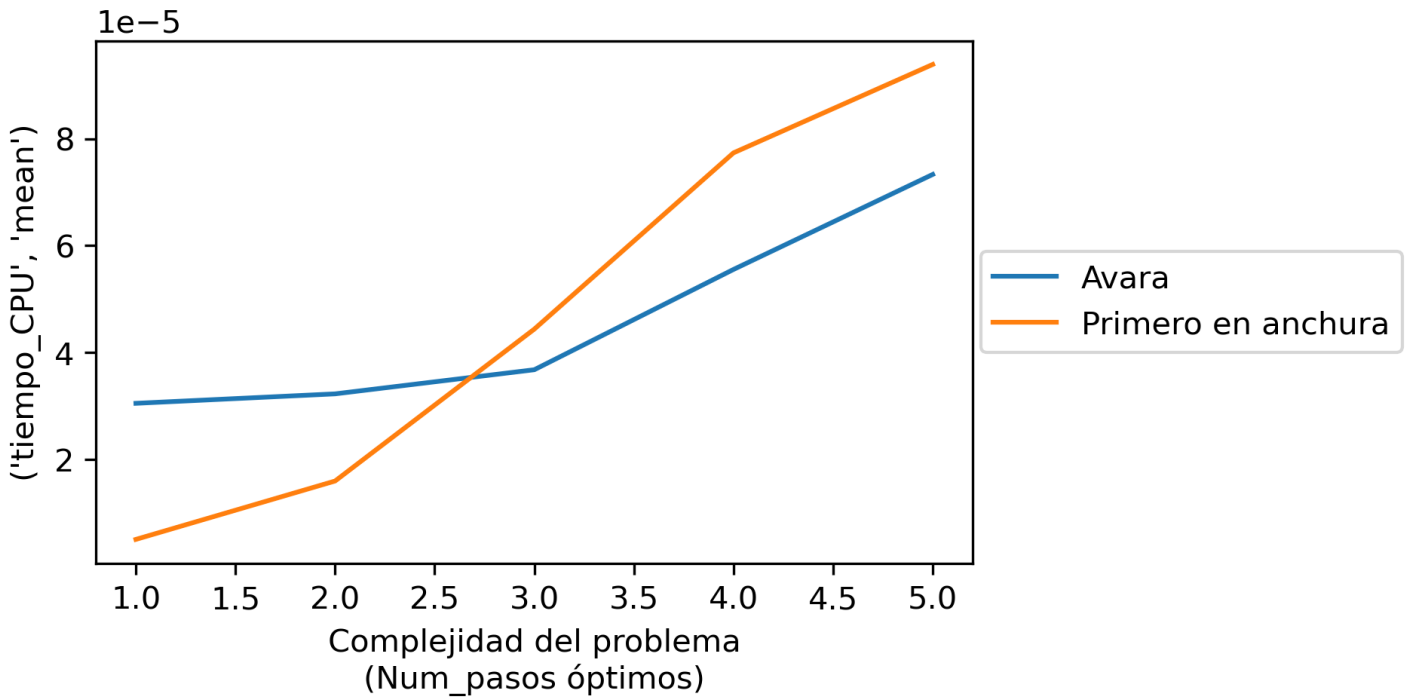
Función	Tiempo en CPU (μs)	Núm. de pasos	Kilómetros
Anchura	.026	**3**	524
Profundidad	.033	6	653
Backtracking	**.019**	7	728
Dijkstra	.045	4	**508**
Avara	.035	**3**	524
A^*	.048	4	**508**

De los algoritmos que dan una solución óptima, el más rápido es Dijkstra.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Avara vs. Anchura

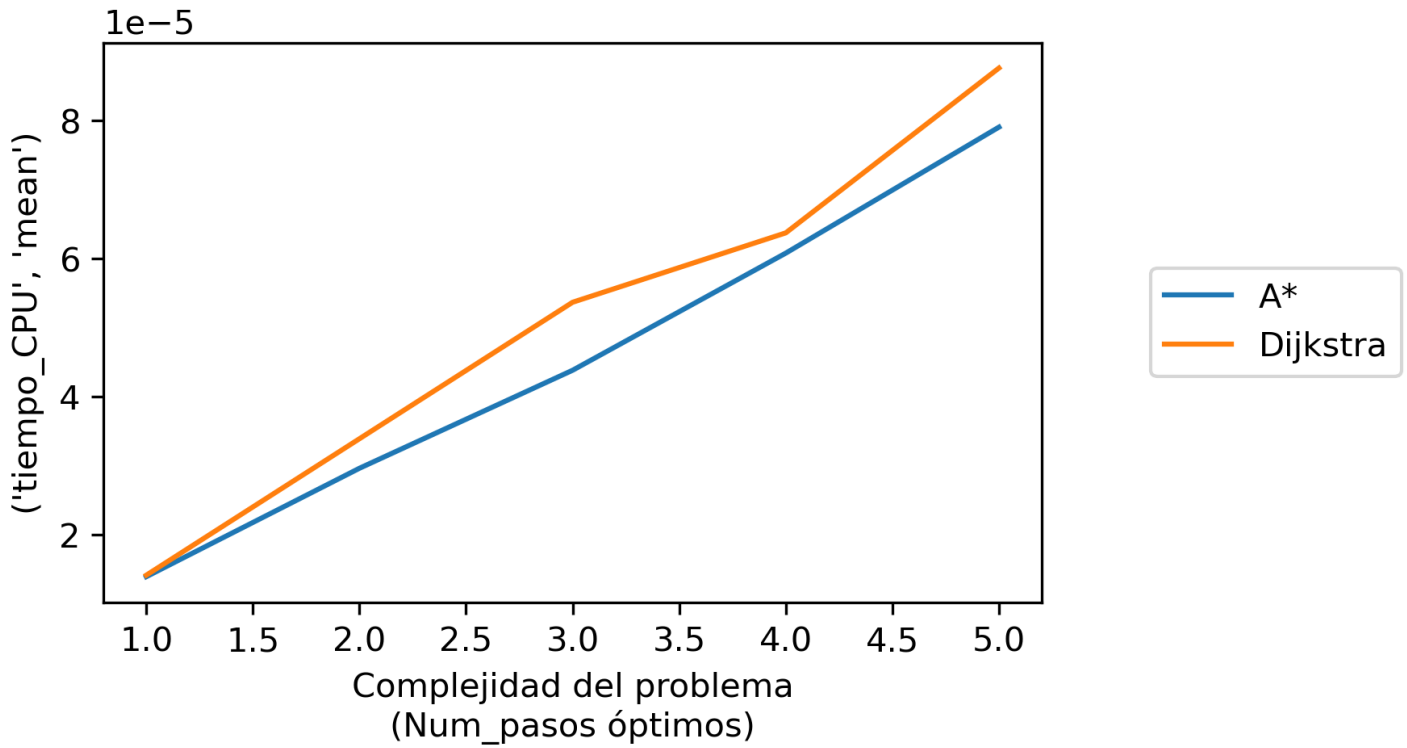
Caso: Test suite. Inicio: Pitesti. Complejidad: Una ciudad intermedia más.



Matemáticas Aplicadas y Ciencias de la Computación

Dijkstra vs. A^*

Caso: Test suite. Inicio: Pitesti. Complejidad: Una ciudad intermedia más.



Contenido

Definición

Heurística:

1. adj. Perteneciente o relativo a la heurística.

2. f. Técnica de indagación y descubrimiento.

Definición

Heurística:

1. adj. Perteneciente o relativo a la heurística.

2. f. Técnica de indagación y descubrimiento.

☞ Uso de información específica del problema para facilitar el proceso de búsqueda.

Definición

Heurística:

1. adj. Perteneciente o relativo a la heurística.

2. f. Técnica de indagación y descubrimiento.

☞ Uso de información específica del problema para facilitar el proceso de búsqueda.

☞ Una medida de costo estimado desde un estado hasta el objetivo.

Problema del rompecabezas de 8 piezas

	4	8
5	1	3
2	6	7

Se mueven las piezas . . .

Problema del rompecabezas de 8 piezas

5	4	8
	1	3
2	6	7

Se mueven las piezas . . .

Problema del rompecabezas de 8 piezas

5	4	8
1		3
2	6	7

Se mueven las piezas . . .

Problema del rompecabezas de 8 piezas

5	4	8
1		3
2	6	7

Hasta que finalmente se obtiene . . .

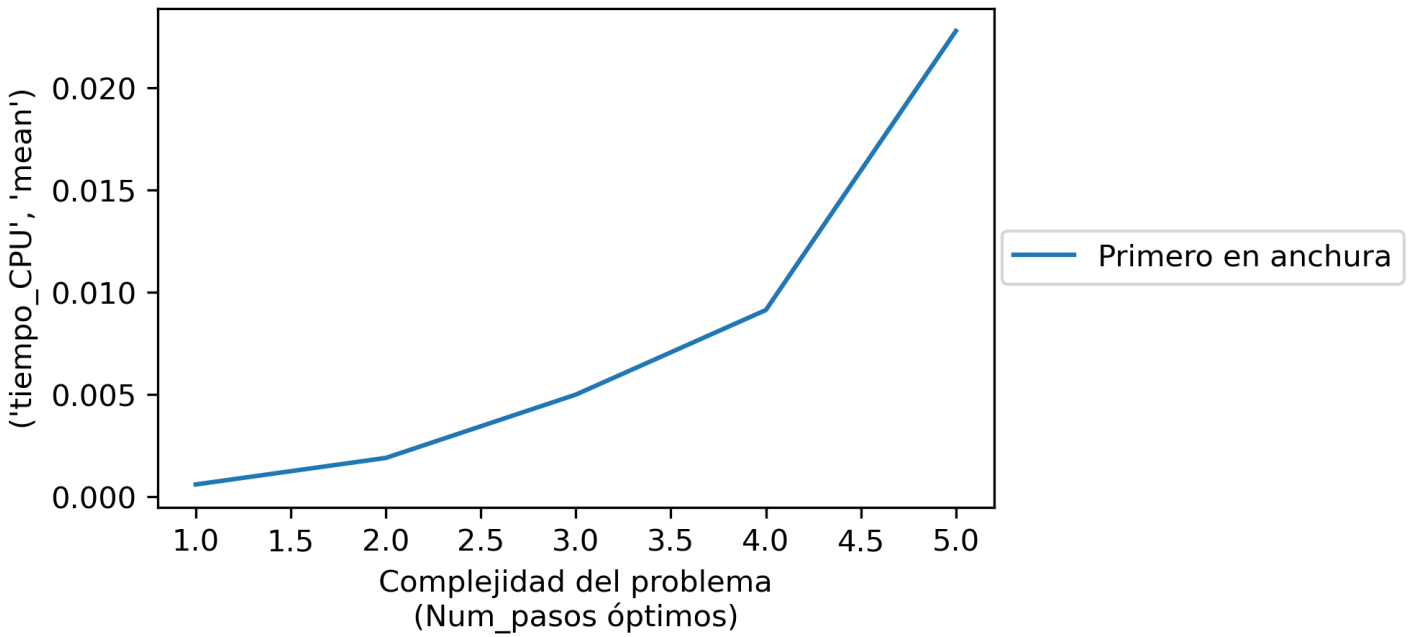
Problema del rompecabezas de 8 piezas

	1	2
3	4	5
6	7	8

El estado objetivo.

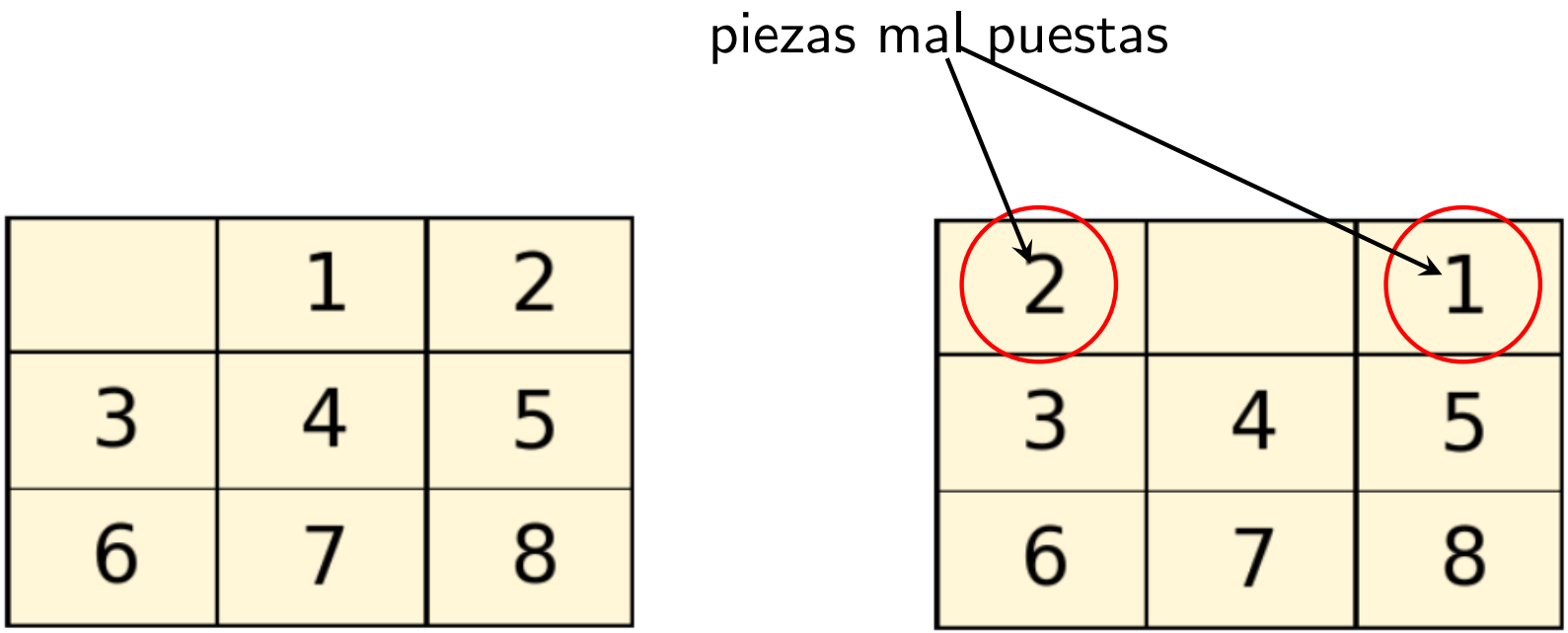
Necesidad de la heurística

Resultados obtenidos en el test suite de rompecabezas.



MACC
Matemáticas Aplicadas y Ciencias de la Computación

Distancia piezas mal puestas



Distancia piezas mal puestas $= 2$

Distancia Manhattan

	1	2
3	4	5
6	7	8

	3	2
4	1	5
6	7	8

El 1 dista uno de su lugar

Distancia Manhattan

	1	2
3	4	5
6	7	8

	3	2
4	1	5
6	7	8

El 1 dista uno de su lugar

El 3 dista dos de su lugar

Distancia Manhattan

	1	2
3	4	5
6	7	8

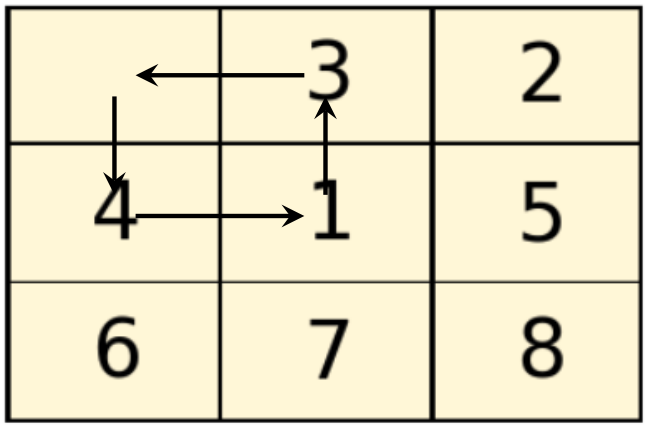
	3	2
4	1	5
6	7	8

El 1 dista uno de su lugar
El 3 dista dos de su lugar
El 4 dista uno de su lugar

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Distancia Manhattan

	1	2
3	4	5
6	7	8

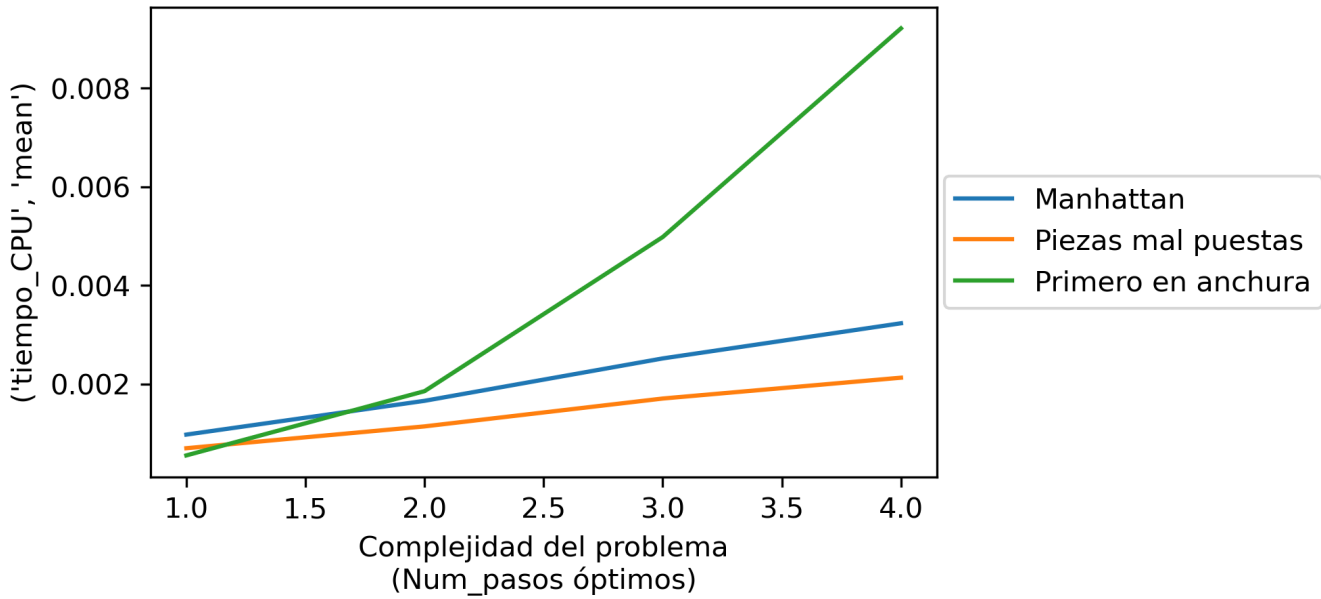


El 1 dista uno de su lugar
El 3 dista dos de su lugar
El 4 dista uno de su lugar

Distancia Manhattan $= 4$

Comparación empírica

Resultados obtenidos en el test suite de rompecabezas.

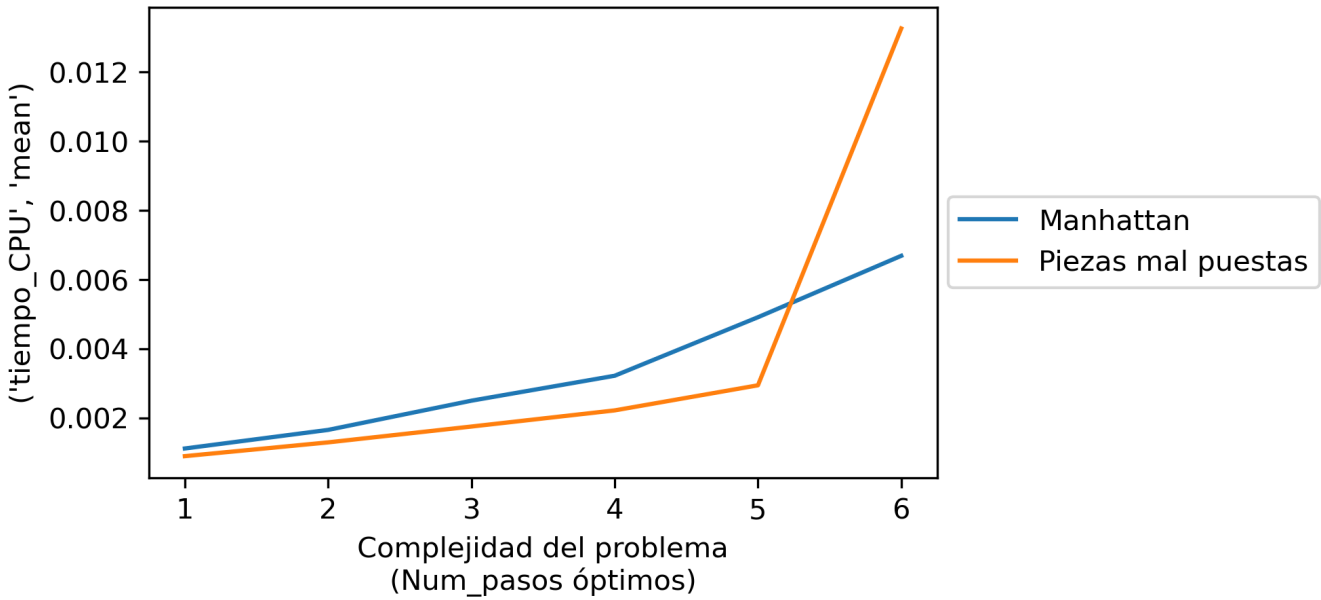


Las heurísticas son mucho más rápidas que la búsqueda primero en anchura.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Comparación empírica

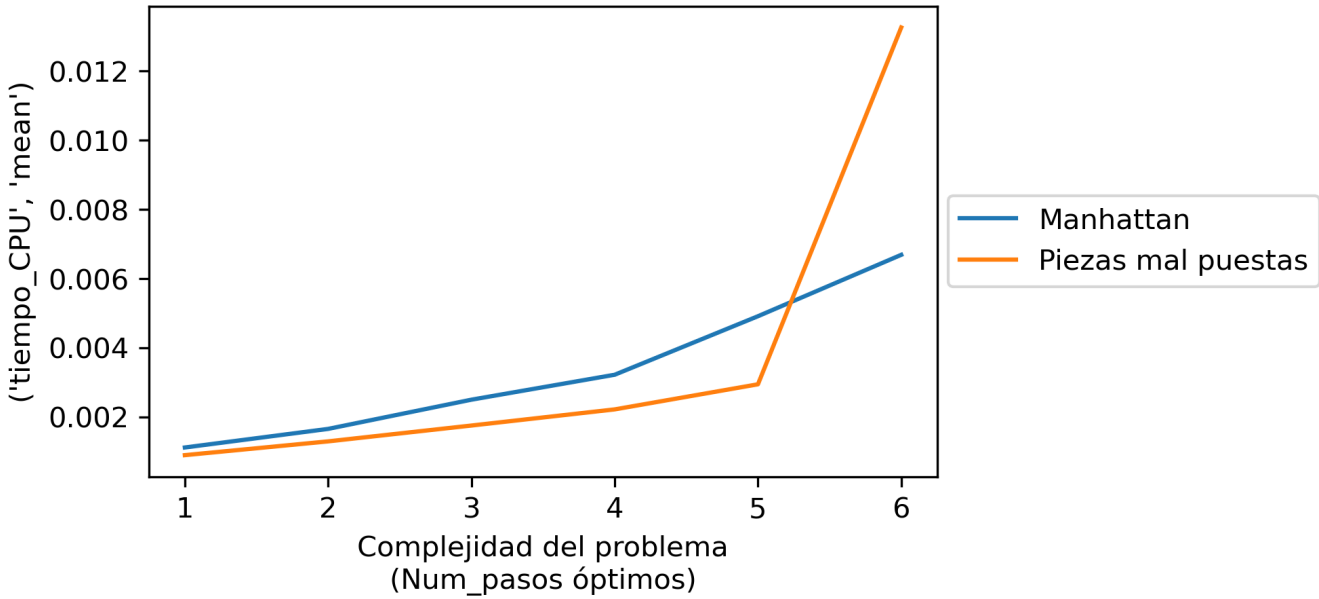
Resultados obtenidos en el test suite de rompecabezas.



MACC
Matemáticas Aplicadas y Ciencias de la Computación

Comparación empírica

Resultados obtenidos en el test suite de rompecabezas.

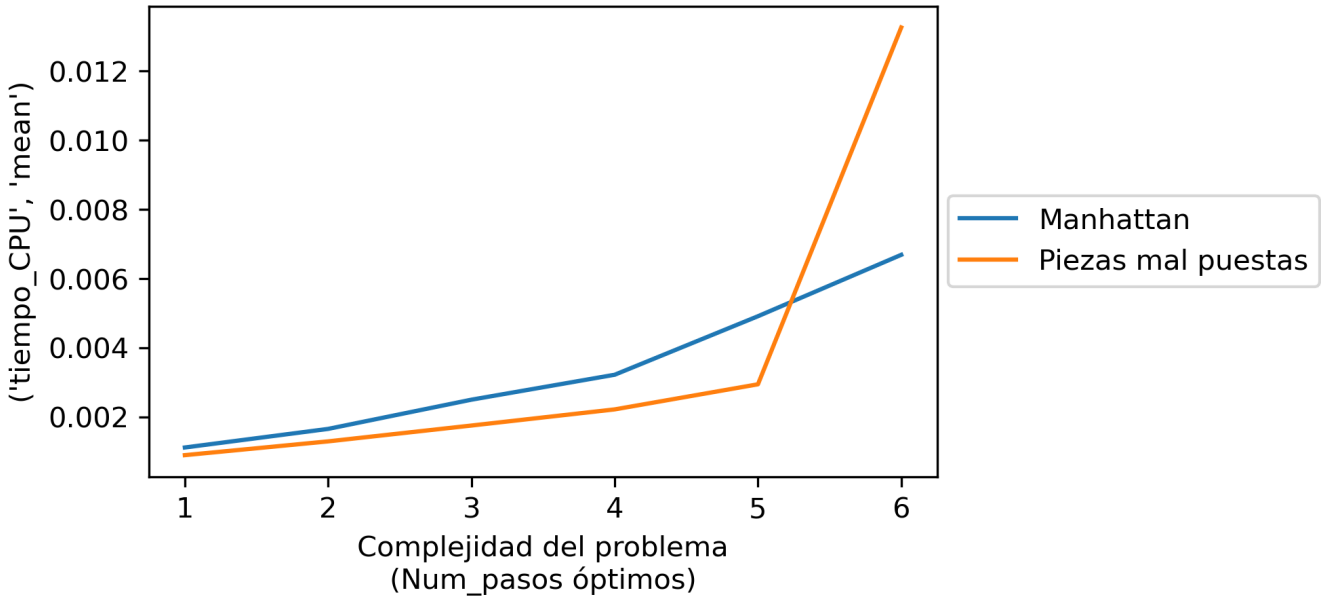


La heurística Manhattan da mejores resultados, puesto que estima mejor.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Comparación empírica

Resultados obtenidos en el test suite de rompecabezas.

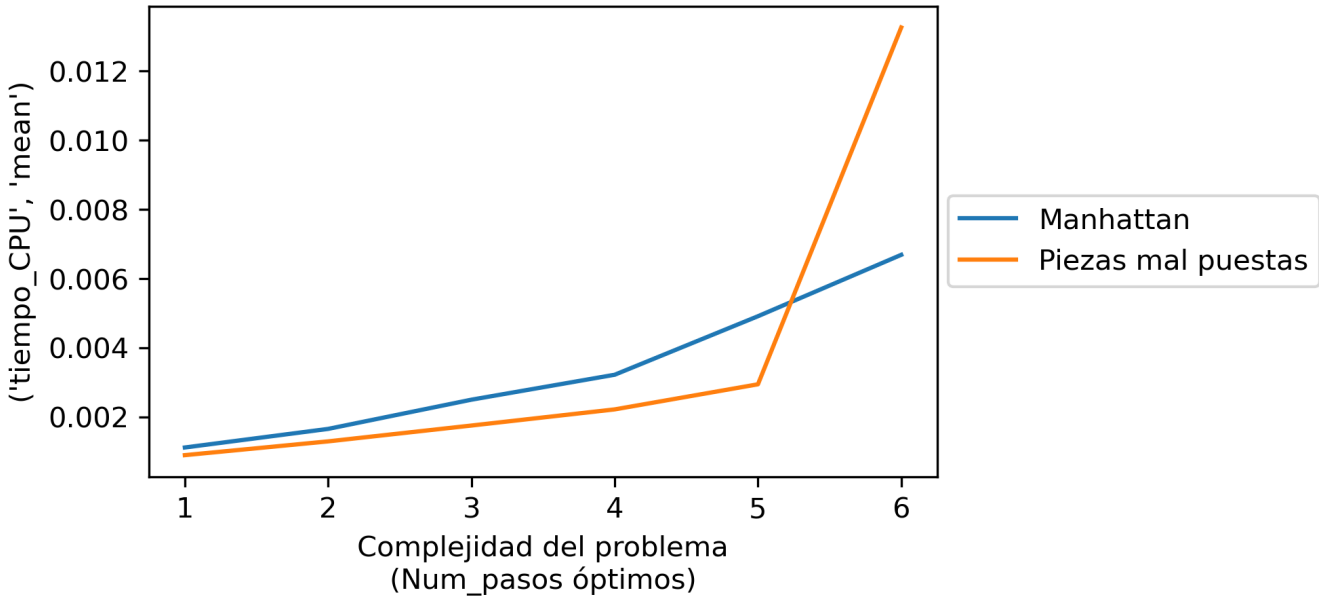


La exploración recorre menos nodos.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Comparación empírica

Resultados obtenidos en el test suite de rompecabezas.

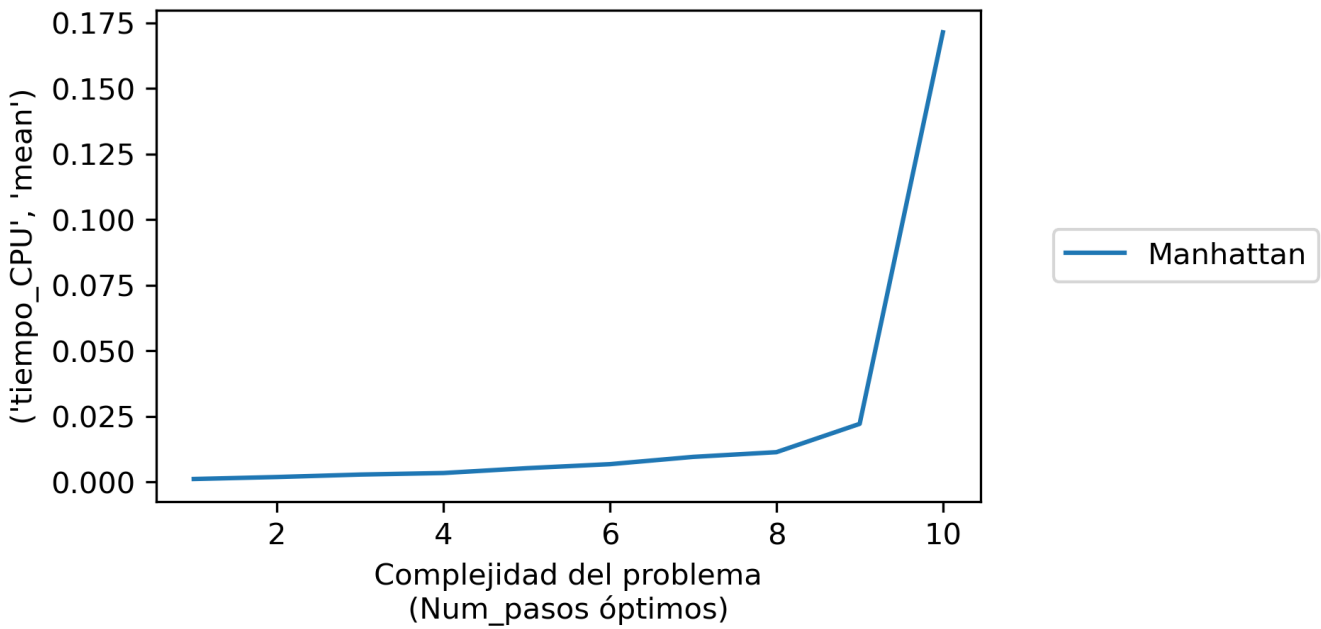


Por ejemplo, si el 1 está a 3 movimientos de distancia, Manhattan sumará 3, mientras que piezas mal puestas sólo suma 1.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

La maldición de la dimensionalidad

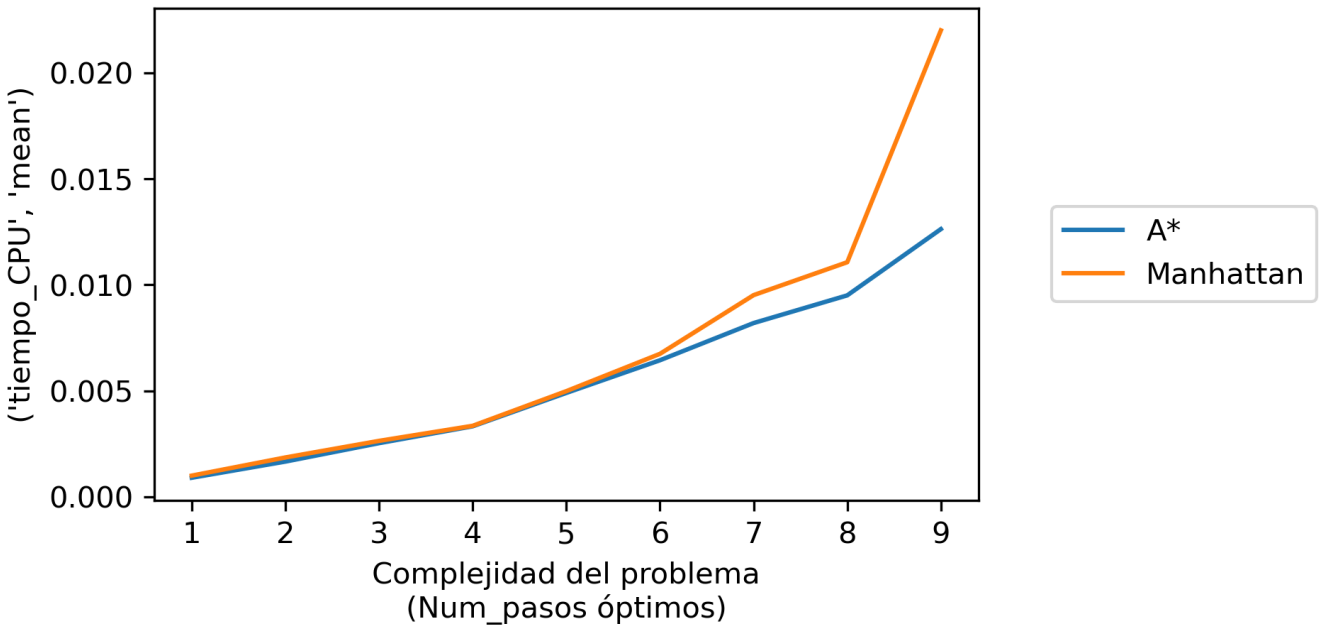
Resultados obtenidos en el test suite de rompecabezas.



Todos los algoritmos son exponenciales en el número de pasos.

A^*

Resultados obtenidos en el test suite de rompecabezas.



El algoritmo A^* es mejor que búsqueda avara. Ambos algoritmos corren con la heurística manhattan.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

A^* con pesos

Versión original:

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

donde g es el costo_camino y h la heurística.

A^* con pesos

Versión original:

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

donde g es el costo_camino y h la heurística.

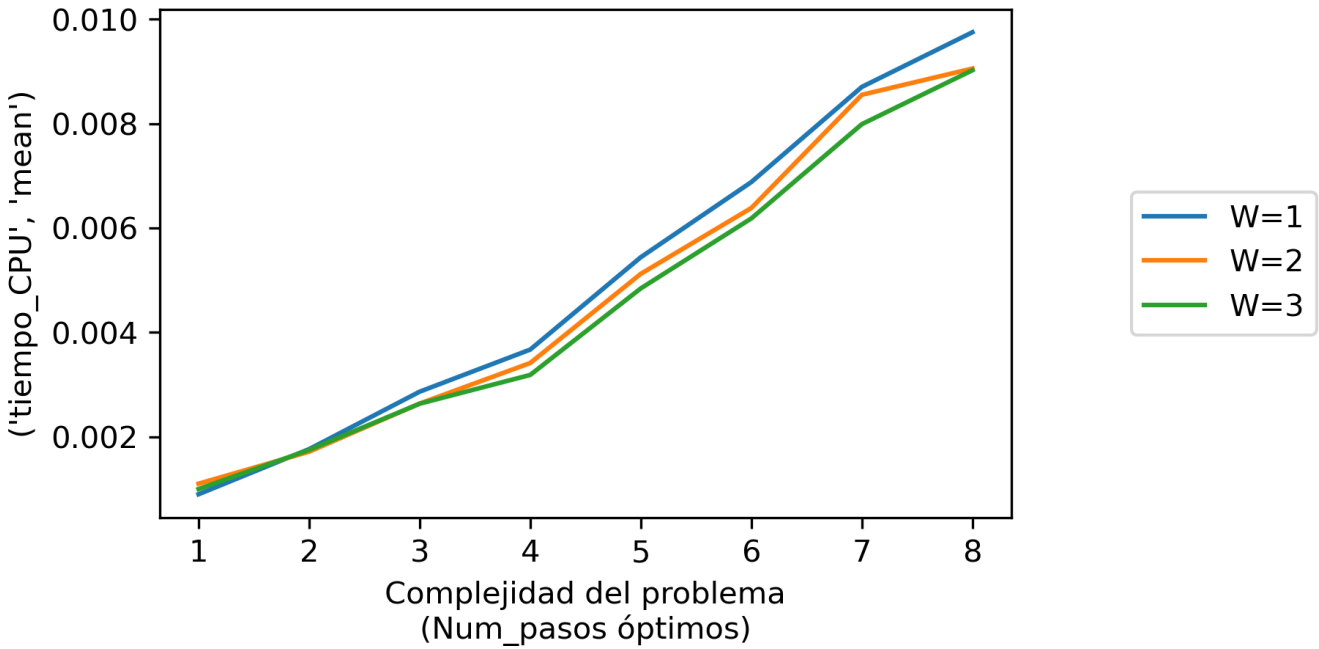
Versión con pesos:

$$f(n) = g(n) + W \times h(n)$$

donde $W \geq 1$.

A^* con pesos

Resultados obtenidos en el test suite de rompecabezas.



Al hacer $W = 3$, los tiempos de ejecución mejoran un poco.

Contenido

Backtracking con heurística

Esta variación del backtracking usa un poco más de memoria que el algoritmo tradicional, pero lo compensa dando prioridad en la exploración a aquellos nodos más prometedores de acuerdo a la heurística.

Backtracking con heurística

Esta variación del backtracking usa un poco más de memoria que el algoritmo tradicional, pero lo compensa dando prioridad en la exploración a aquellos nodos más prometedores de acuerdo a la heurística.

- Tiene tres argumentos: problema, nodo y heurística.

Backtracking con heurística

Esta variación del backtracking usa un poco más de memoria que el algoritmo tradicional, pero lo compensa dando prioridad en la exploración a aquellos nodos más prometedores de acuerdo a la heurística.

- Tiene tres argumentos: problema, nodo y heurística.
- Expande todos los hijos no cíclicos del nodo y los ordena por su valor de acuerdo a la heurística.

Backtracking con heurística

Esta variación del backtracking usa un poco más de memoria que el algoritmo tradicional, pero lo compensa dando prioridad en la exploración a aquellos nodos más prometedores de acuerdo a la heurística.

- Tiene tres argumentos: problema, nodo y heurística.
- Expande todos los hijos no cíclicos del nodo y los ordena por su valor de acuerdo a la heurística.
- Hace una llamada recursiva sobre los hijos en orden.

Beam search

Es una búsqueda avara en la cual la frontera tiene un tamaño limitado, k, llamado el ancho del haz. Este uso limitado de memoria puede hacer más rápida la búsqueda al restringirla sobre un subconjunto del entorno. Pero la solución puede ser subóptima o puede no encontrarla.

Beam search

Es una búsqueda avara en la cual la frontera tiene un tamaño limitado, k, llamado el ancho del haz. Este uso limitado de memoria puede hacer más rápida la búsqueda al restringirla sobre un subconjunto del entorno. Pero la solución puede ser subóptima o puede no encontrarla.

1. Tiene un argumento: problema.

Beam search

Es una búsqueda avara en la cual la frontera tiene un tamaño limitado, k, llamado el ancho del haz. Este uso limitado de memoria puede hacer más rápida la búsqueda al restringirla sobre un subconjunto del entorno. Pero la solución puede ser subóptima o puede no encontrarla.

1. Tiene un argumento: problema.
2. Crea una lista prioritaria llamada frontera que ordena los nodos por heurística y que tiene un tamaño limitado a k.

Beam search

Es una búsqueda avara en la cual la frontera tiene un tamaño limitado, k, llamado el ancho del haz. Este uso limitado de memoria puede hacer más rápida la búsqueda al restringirla sobre un subconjunto del entorno. Pero la solución puede ser subóptima o puede no encontrarla.

1. Tiene un argumento: problema.
2. Crea una lista prioritaria llamada frontera que ordena los nodos por heurística y que tiene un tamaño limitado a k.
3. Hace pop a frontera y chequea el test objetivo sobre el nodo obtenido.

Beam search

Es una búsqueda avara en la cual la frontera tiene un tamaño limitado, k, llamado el ancho del haz. Este uso limitado de memoria puede hacer más rápida la búsqueda al restringirla sobre un subconjunto del entorno. Pero la solución puede ser subóptima o puede no encontrarla.

1. Tiene un argumento: problema.
2. Crea una lista prioritaria llamada frontera que ordena los nodos por heurística y que tiene un tamaño limitado a k.
3. Hace pop a frontera y chequea el test objetivo sobre el nodo obtenido.
4. Por cada hijo del nodo, considera si no es cíclico y su valor heurístico. Lo incluye en la frontera, cuidando de nunca superar el tamaño k. Vuelve al paso 3.

Take away

En esta sesión usted aprendió:

- El uso de costos para mejorar la solución encontrada mediante el algoritmo "primero el mejor".
- El uso de heurísticas para mejorar el proceso de búsqueda.
- El algoritmo A^* para combinar ambas ventajas.
- Algunos algoritmos adicionales que usan heurísticas, pero en la literatura pueden encontrarse muchísimas variaciones y alternativas.