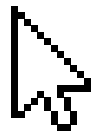
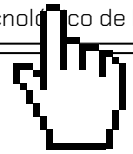
(72.27) Sistemas de Inteligencia Artificial

Trabajo Práctico Nº1

Métodos de Búsqueda



Alejo Flores Lucey 62622
Andrés Carro Wetzel 61655
Ian Franco Tognetti 61215
Matías Daniel Della Torre 61016



1 Introducción

Presentación del tema

2 Ejercicio 1

8 - Puzzle

3 Ejercicio 2

Sokoban

3.1 Implementación

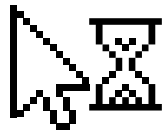
Parser - Métodos de Búsqueda - Heurísticas

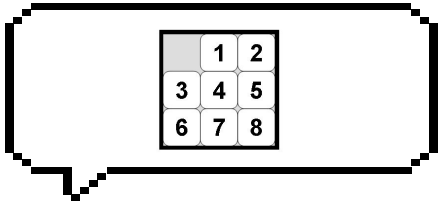
3.2 Análisis de Resultados

Costo y Tiempo de Procesamiento

Introducción

El objetivo de este Trabajo Práctico es resolver distintos juegos mediante Algoritmos de Búsqueda Desinformados e Informados, para luego realizar la interpretación de los resultados.





01

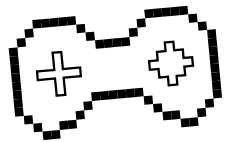
8 - Puzzle

Juego que consiste en un marco de fichas numeradas en orden aleatorio a las que les falta una ficha. El objetivo es desplazar las fichas para ordenarlas ascendentemente.

Estructura de estados

Se presenta en forma de matriz con el estado inicial:

5	7	3
8	2	
1	6	4



Se podría usar una estructura de estado en forma de lista que contempla repetidos: 573820164

- Ejemplo de estado repetido → 18562740**3**

OBS: El 0 representa la casilla vacía

Estructura de estados

Para verificar dichos estados repetidos, guardamos la información de los casilleros adyacentes (vecinos) a las 4 esquinas, entonces chequeamos en los siguientes estados si ya pasó por ese estado…

Tomemos el siguiente estado de ejemplo:

8		3
7	1	2
6	5	4

Su representación en lista es 8**0**3712654

- Nos guardamos los vecinos de las 4 esquinas

8 → [0, 7] | 3 → [0, 2] | 6 → [5, 7] | 4 → [2, 5]

OBS: llamamos esquina a los valores en los índices 0, 2, 6 y 8 en la lista. Veamos que es válido…

6	7	8
5	1	
4	2	3

Este es un estado rotado!
8**0**3712654
678510423

4	7	8
5	1	
6	2	3

Este **NO** es un estado rotado!
8**0**3712654
47851**0****6**23

Estructura de estados

573820164 - 573802164 - 573082164 - 073582164

703582164 - 783502164 - 783052164 - 783152064

783152604 - 783102654 - 783012654 - 083712654

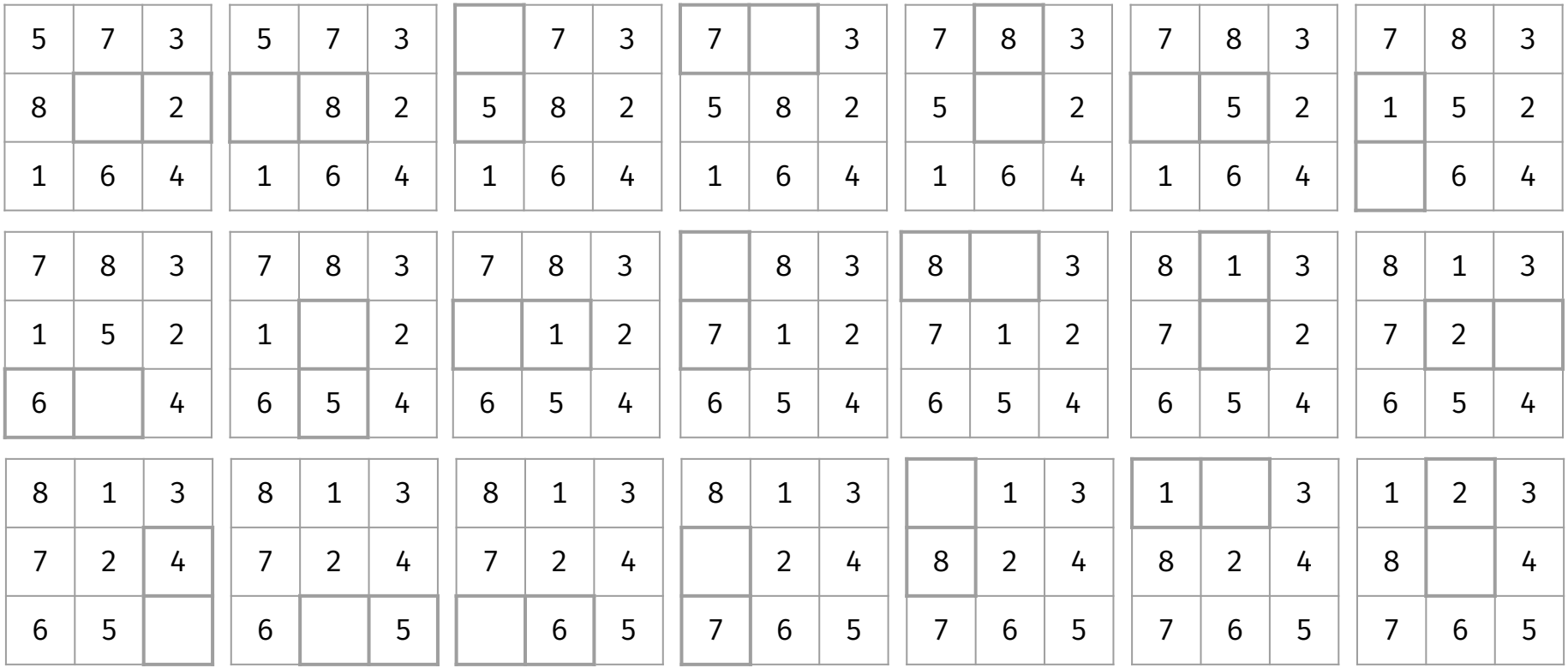
803712654 - 813702654 - 813720654 - 813724650

813724605 - 813724065 - 813024765 - 013824765

103824765 - 123804765

5	7	3
8	2	
1	6	4

Pasos para la Resolución



Resolución: heurísticas

Distancia de Manhattan

Suma de las distancias de cada ficha a su posición correcta, sólo moviéndose verticalmente y horizontalmente.

Cantidad de fichas en posiciones incorrectas

Se cuentan la cantidad de fichas que están en una posición incorrecta.

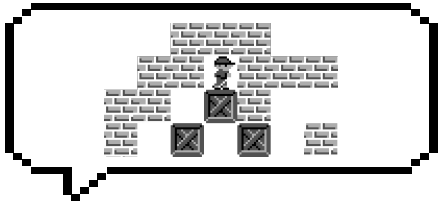
Cantidad de fichas en filas o columnas incorrectas

Se cuentan la cantidad de fichas que están en una fila incorrecta y se suma a la cantidad de fichas que están en una columna incorrecta.

¿Cuál es el mejor algoritmo para resolverlo?

A* pues al ser informado es eficiente para encontrar el camino óptimo (en este caso, la secuencia de movimientos) desde un estado inicial hasta un estado objetivo.

La heurística de Manhattan sería la mejor en este caso.



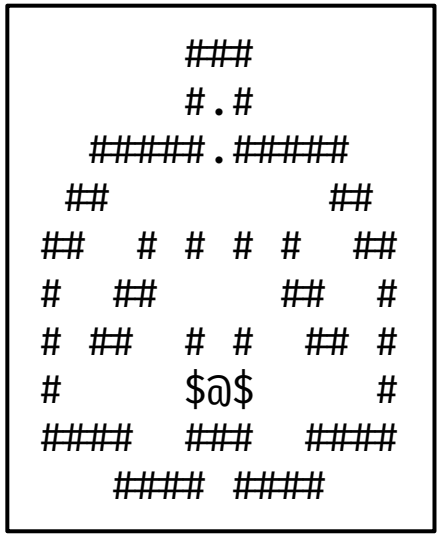
02

Sokoban

Videojuego de puzzle en el que un jugador empuja cajas en un almacén, intentando llevarlas a los lugares de almacenamiento.

Estructura de estados

INPUT



IMMUTABLE STATE

Posición de los objetivos
Posición de las paredes

MUTABLE STATE

Posición de las cajas
Posición del jugador

Detalles de implementación

Se utilizan **sets** (`set()`) para almacenar estados y elementos como cajas para que la operación de inclusión sea **O(1).**

Se utiliza **Pygame** para generar un GIF de la solución encontrada.

Se hace uso del patrón **Singleton** para mantener el estado global del sistema.

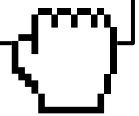
DFS y BFS se implementan usando **deque()**. Greedy y A* se implementan usando **PriorityQueue()**

Para el estudio de resultados, **cada algoritmo se ejecuta 10 veces**, para evitar sacar conclusiones de procesos estocásticos.

¿Cómo finaliza el juego?

Winning State

Cada una de las cajas está posicionada sobre una posición objetivo.



Losing State

También llamados deadlocks. Son estados en los que ya se sabe que no se podrá encontrar una solución.

Redundant State

Estados por los que el jugador ya ha pasado. Se deben repetir las posiciones de las cajas y la del jugador.

Algoritmos de búsqueda implementados

Algoritmos Desinformados

- Breadth First Search (BFS)
- Depth First Search (DFS)

Algoritmos Informados

- Global Greedy Search
- A*

Heurísticas implementadas

Distancia Manhattan

Sumatoria de distancia de caja a objetivo más cercano

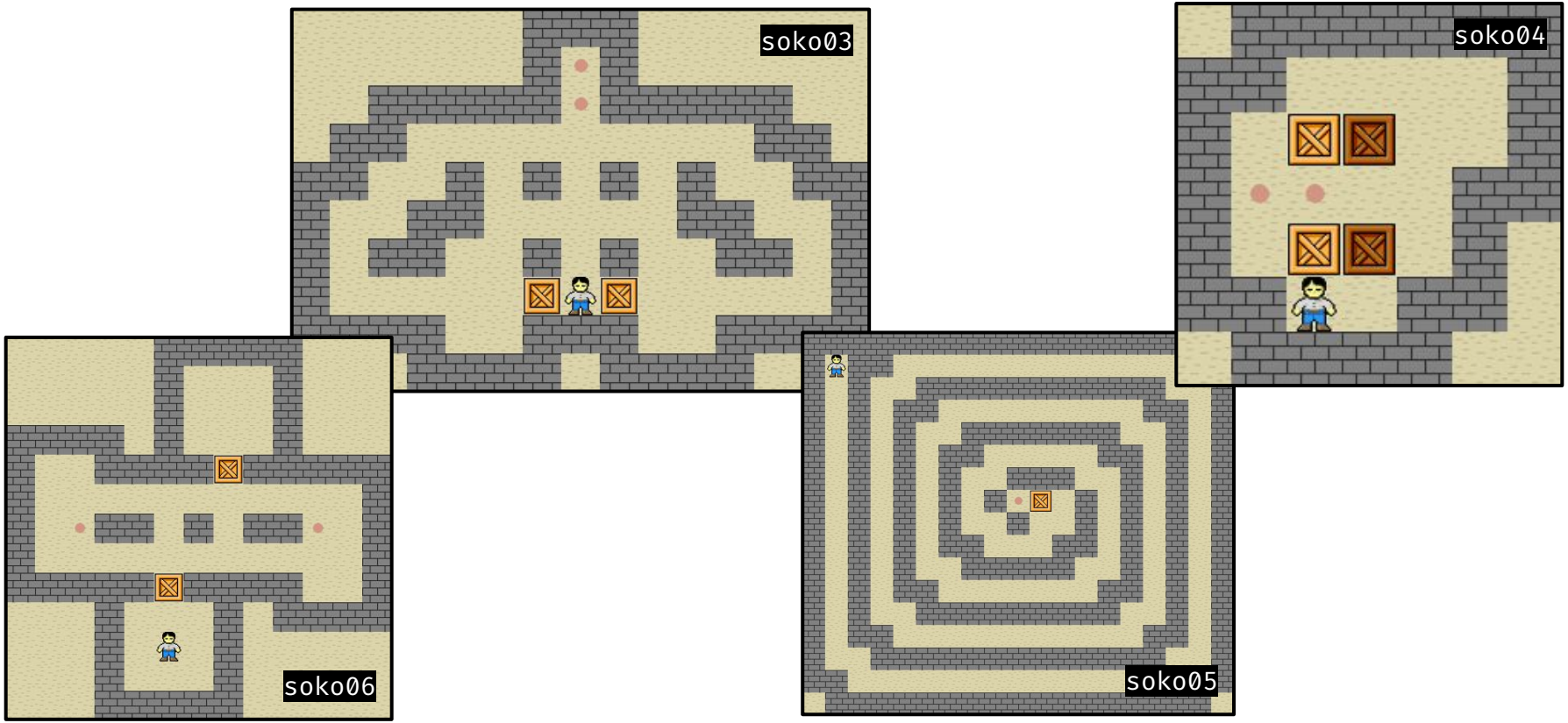
Distancia Euclidea

Sumatoria de distancia de caja a objetivo más cercano

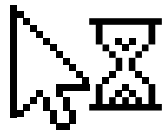
Bounding Box

Utilizando Manhattan, selecciona la distancia máxima

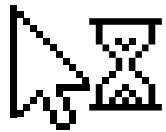
Niveles a resolver



Análisis de los resultados obtenidos



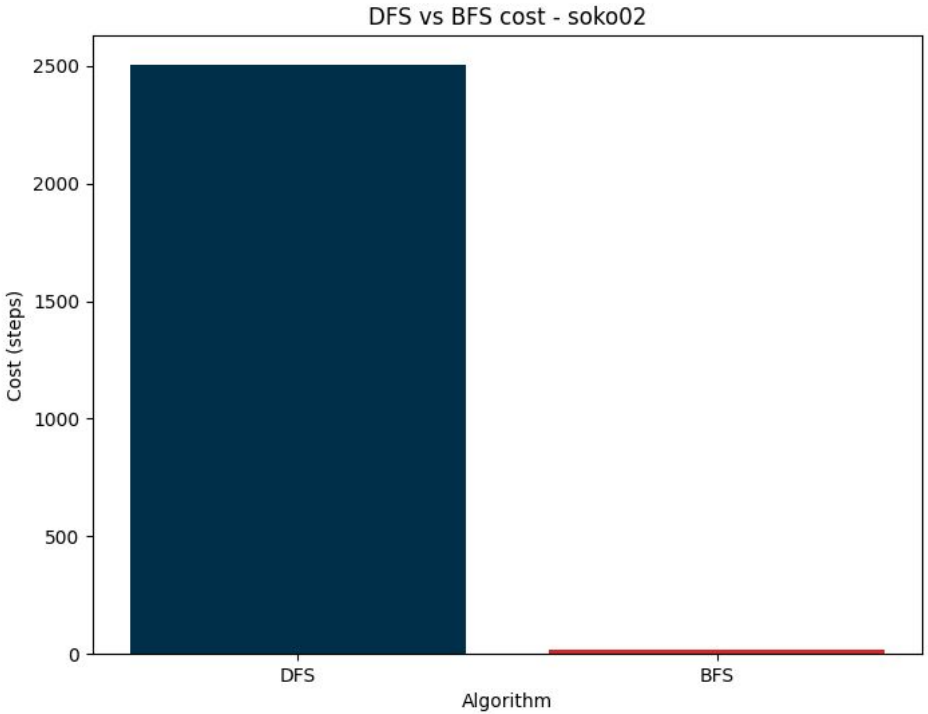
Métodos de Búsqueda Desinformados

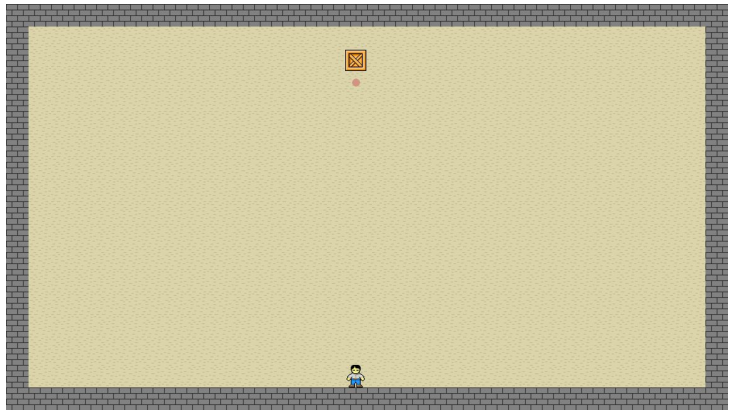


Resolución de soko02 mediante BFS

Resolución de soko02 mediante DFS

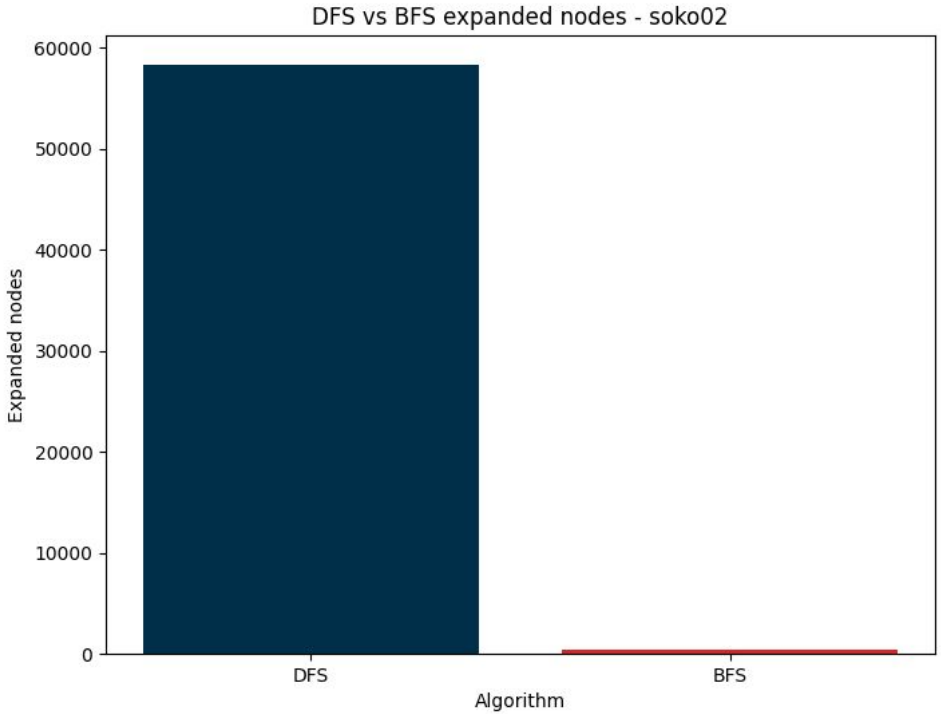
BFS vs DFS - costo

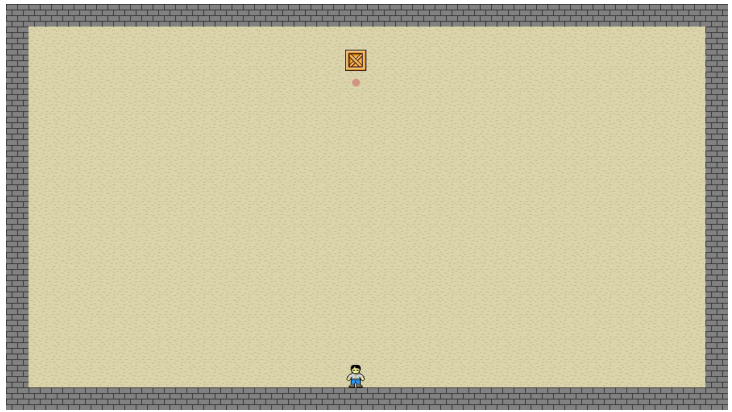




Repeticiones: 10
Tablero: soko02
Algoritmo: BFS & DFS
Heurística: n/a

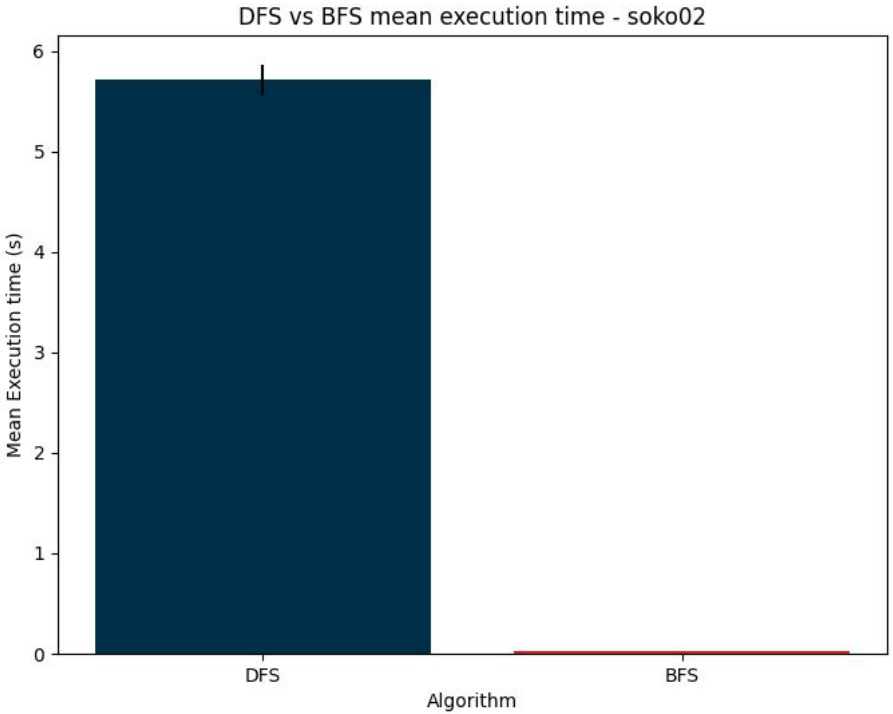
BFS vs DFS - nodos expandidos

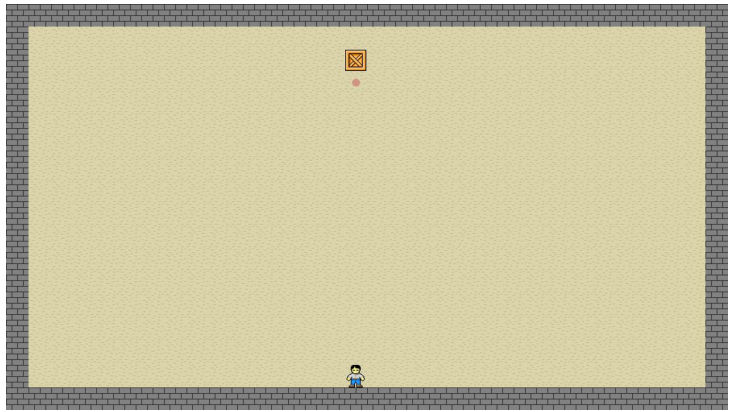




Repeticiones: 10
Tablero: soko02
Algoritmo: BFS & DFS
Heurística: n/a

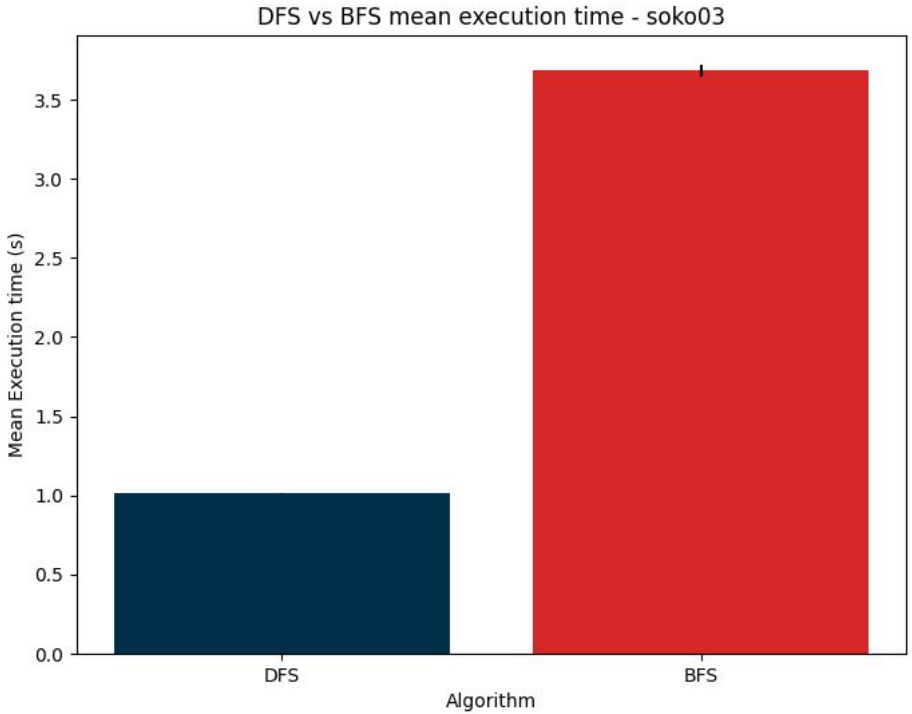
BFS vs DFS - tiempo de ejecución

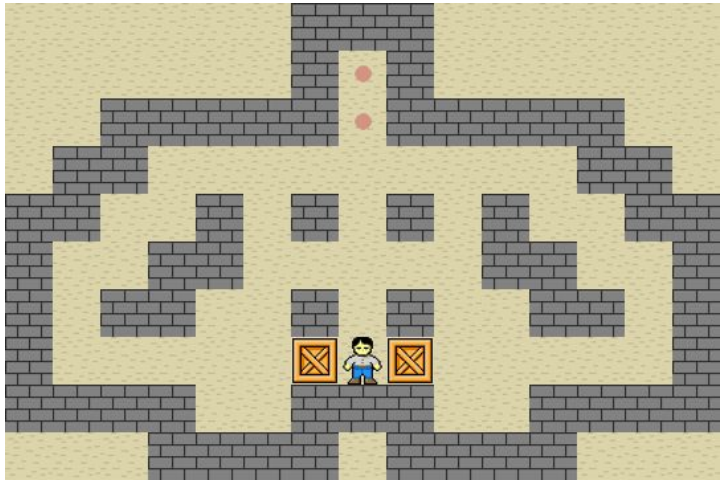




Repeticiones: 10
Tablero: soko02
Algoritmo: BFS & DFS
Heurística: n/a

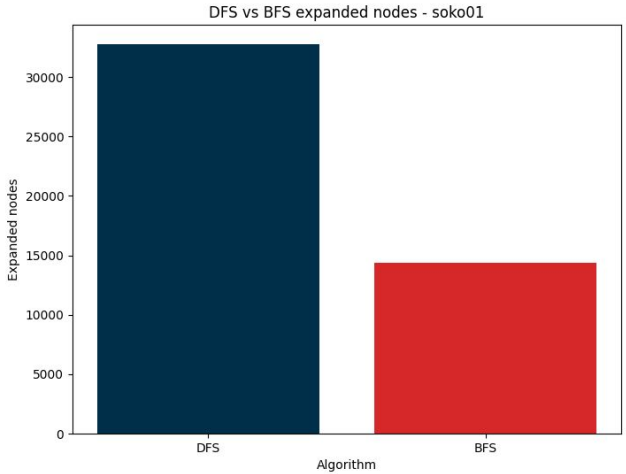
BFS vs DFS - tiempo de ejecución

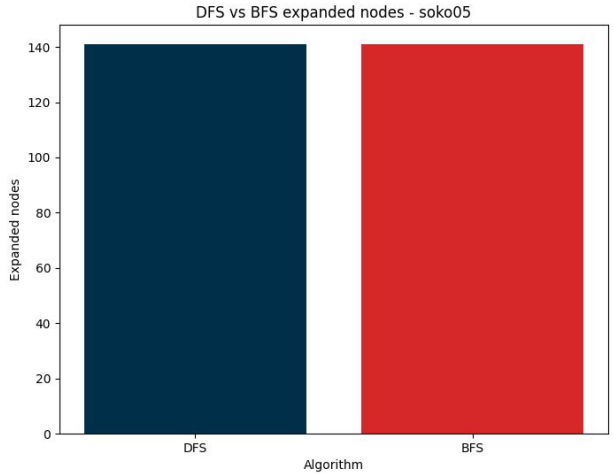


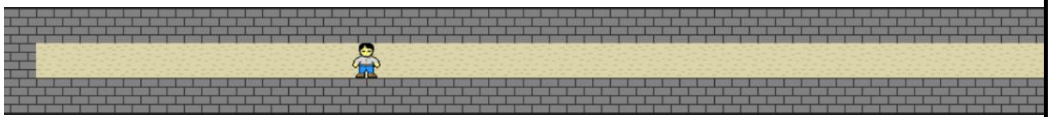


Repeticiones: 10
Tablero: soko03
Algoritmo: BFS & DFS
Heurística: n/a

BFS vs DFS - soko01 vs soko05

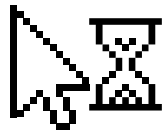


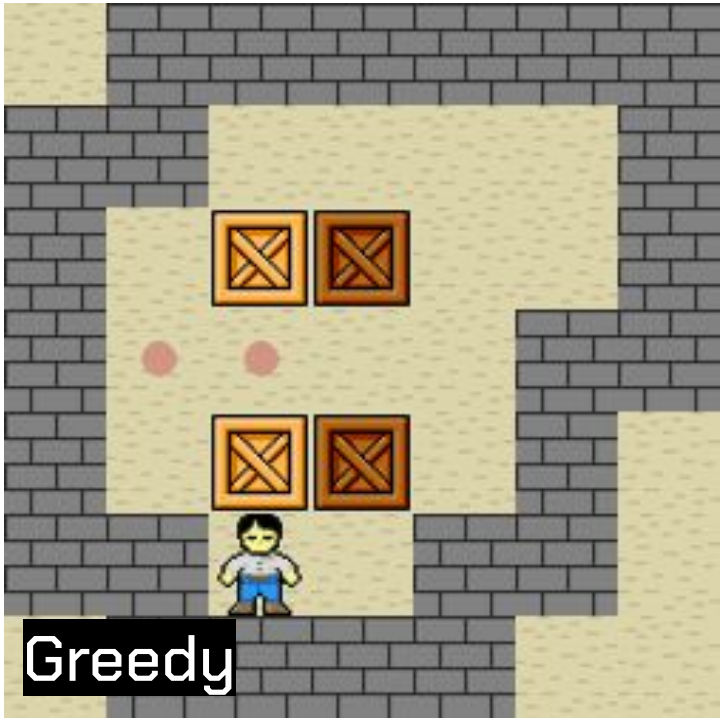


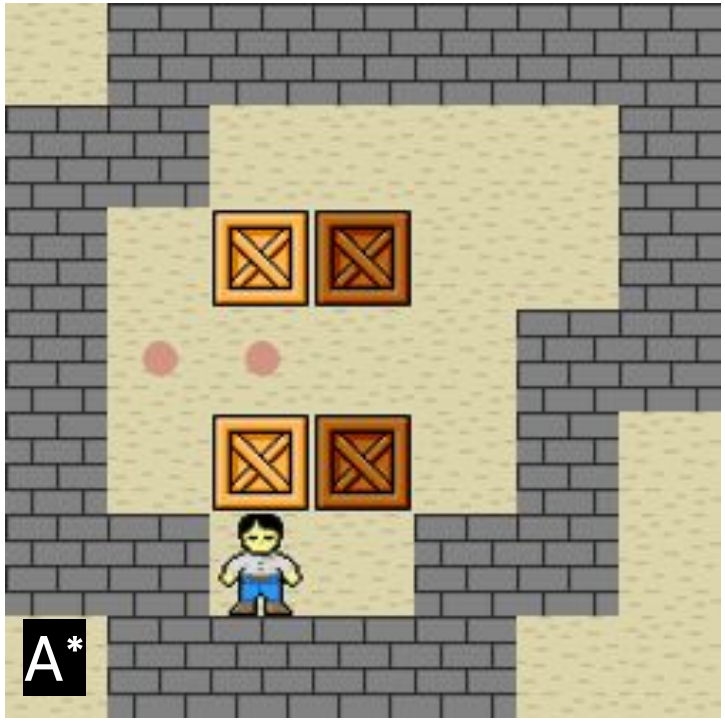


Repeticiones: 10
Tablero: soko01 & soko05
Algoritmo: BFS & DFS
Heurística: n/a

Métodos de Búsqueda Informados

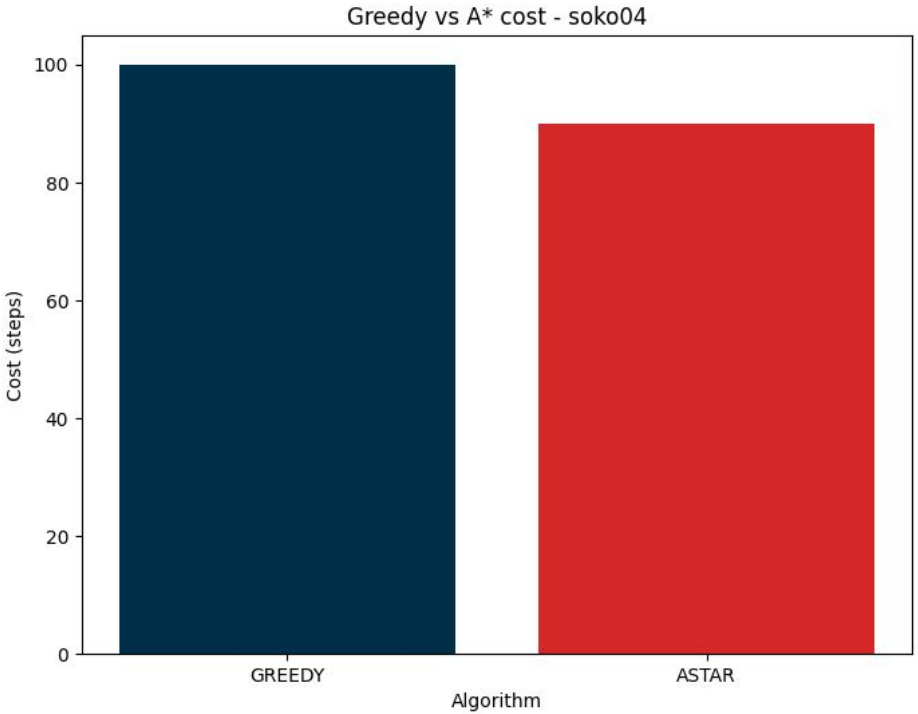


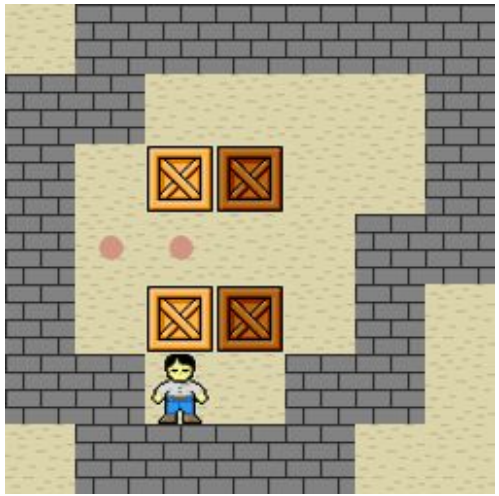




Resolución de soko04 con heurística MANHATTAN

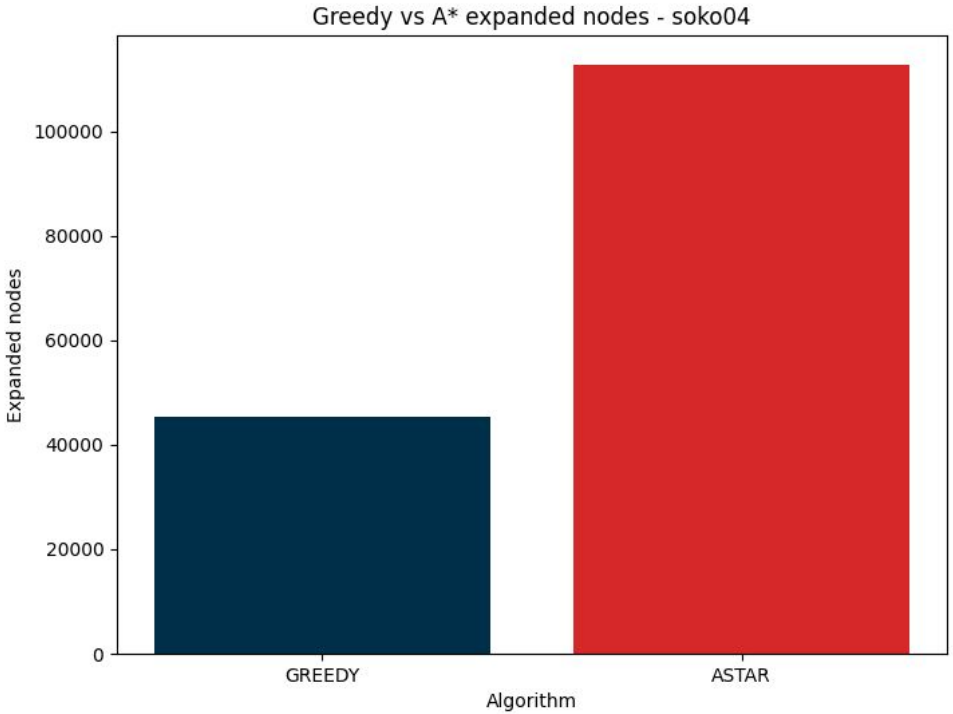
A* vs Greedy - costo

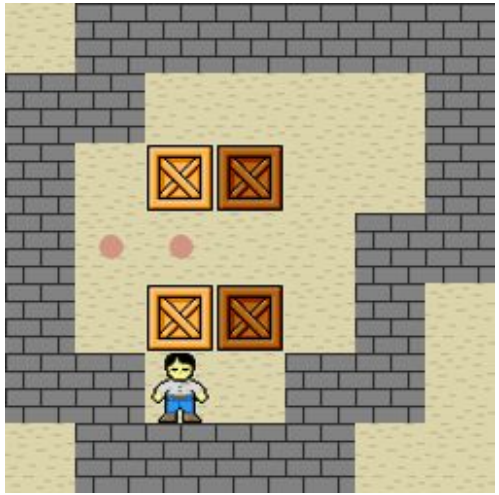




Repeticiones: 10
Tablero: soko04
Algoritmo: Greedy & A*
Heurística: Manhattan Distance

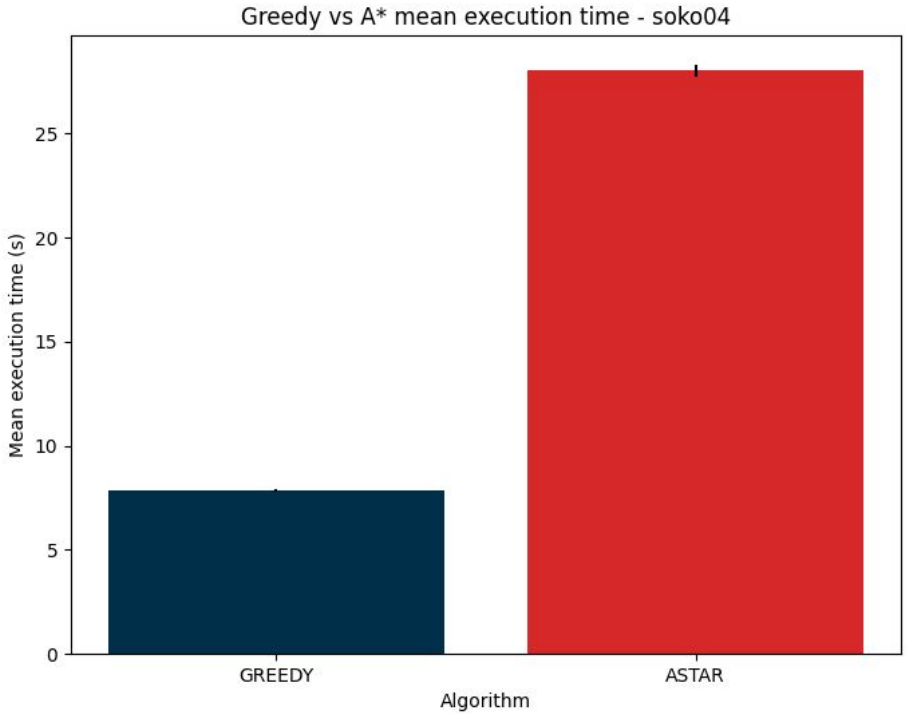
A* vs Greedy - nodos expandidos

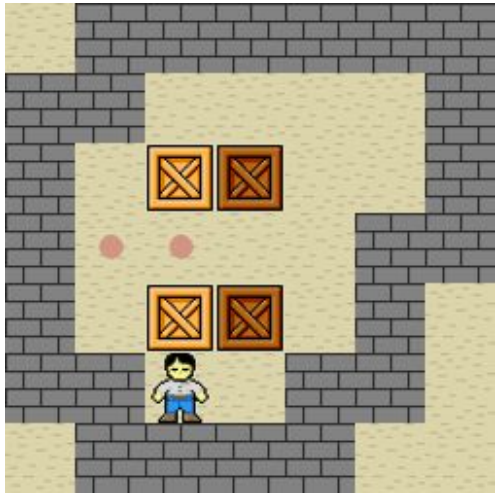




Repeticiones: 10
Tablero: soko04
Algoritmo: Greedy & A*
Heurística: Manhattan Distance

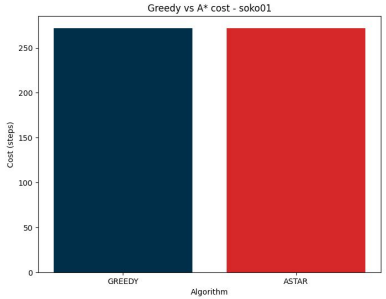
A* vs Greedy - tiempo de ejecución

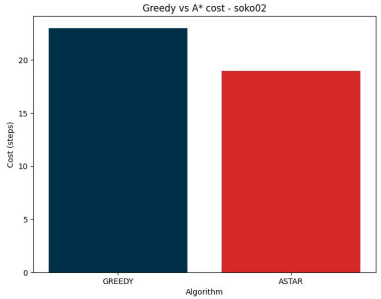


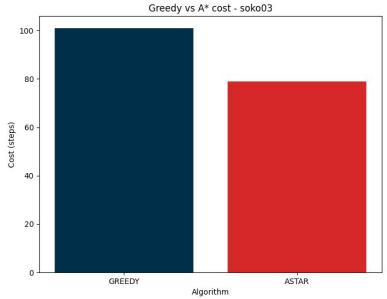


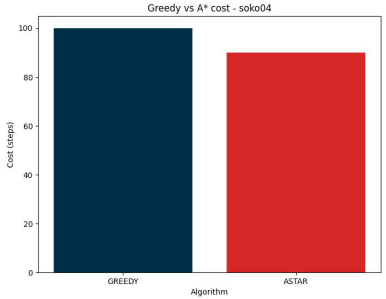
Repeticiones: 10
Tablero: soko04
Algoritmo: Greedy & A*
Heurística: Manhattan Distance

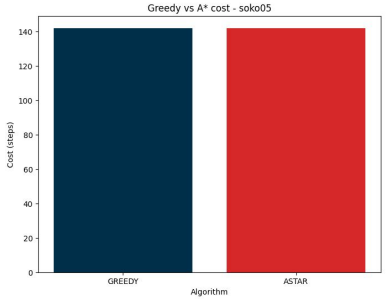
A* vs Greedy - costos

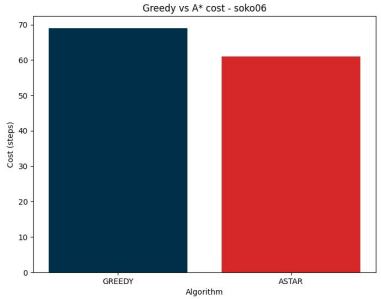






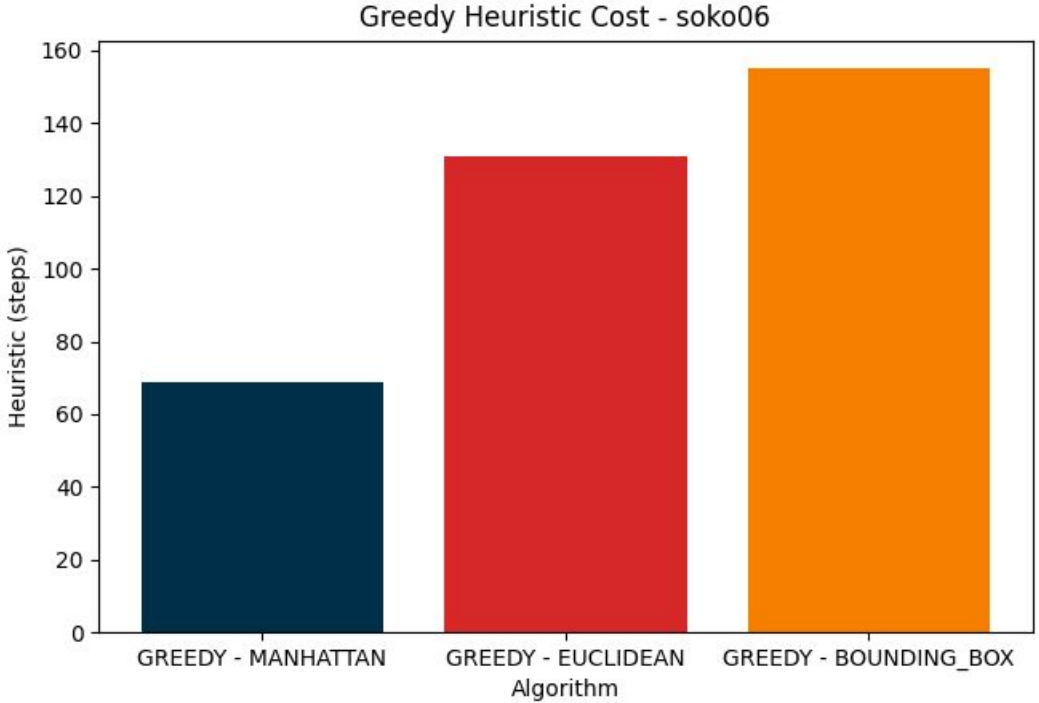


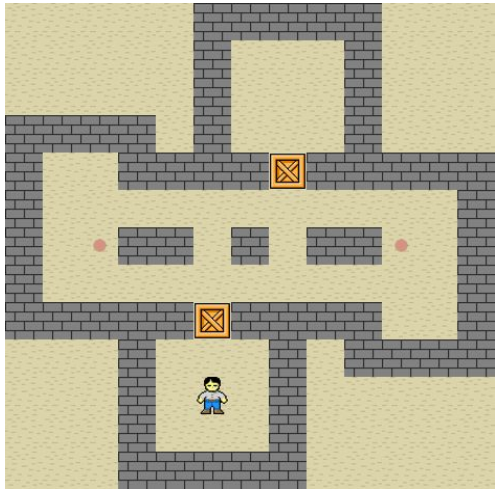




Repeticiones: 10
Tableros: soko01 - soko06
Algoritmo: Greedy & A*
Heurística: Manhattan Distance

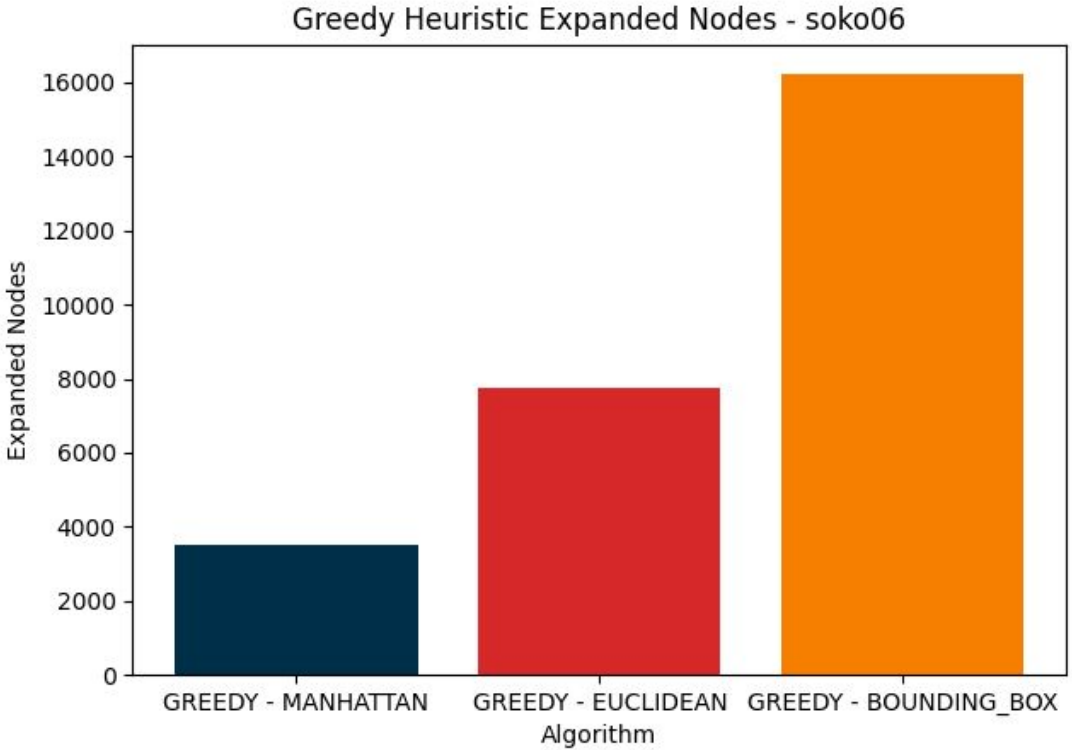
Greedy: Comparación de Heurísticas

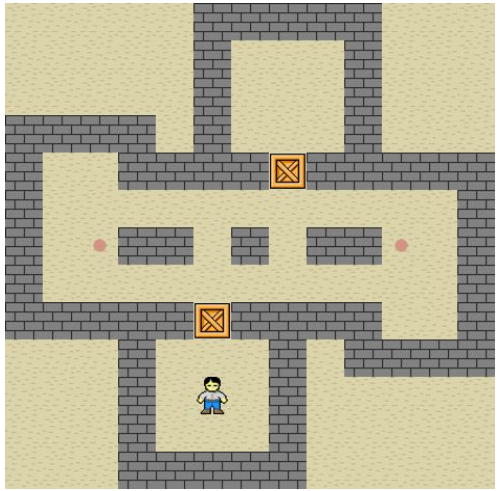




Repeticiones: 10
Tablero: soko06
Algoritmo: Greedy
Heurística: *

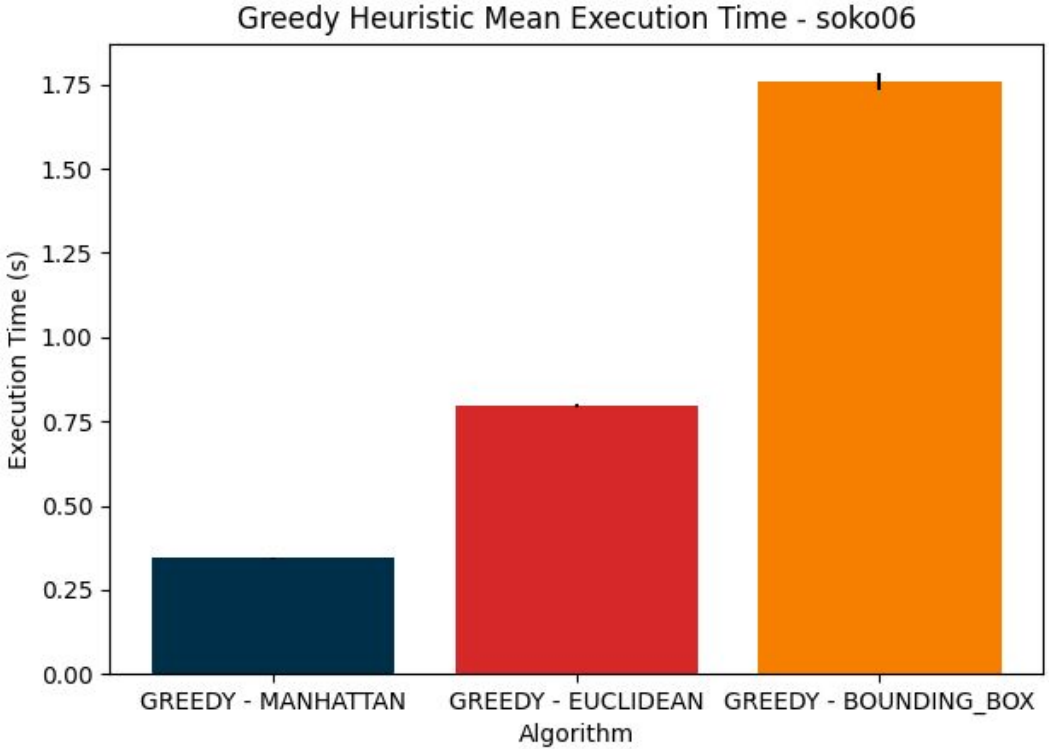
Greedy: Comparación de Heurísticas

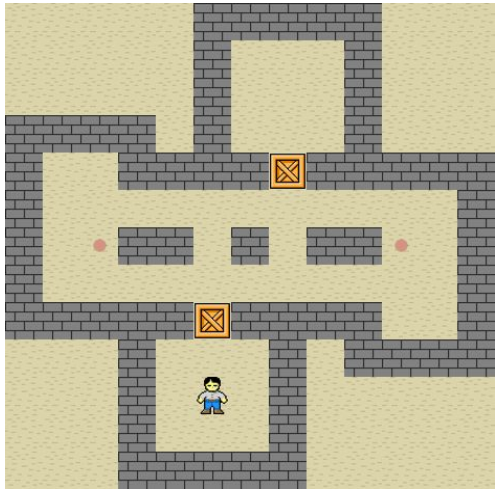




Repeticiones: 10
Tablero: soko06
Algoritmo: Greedy
Heurística: *

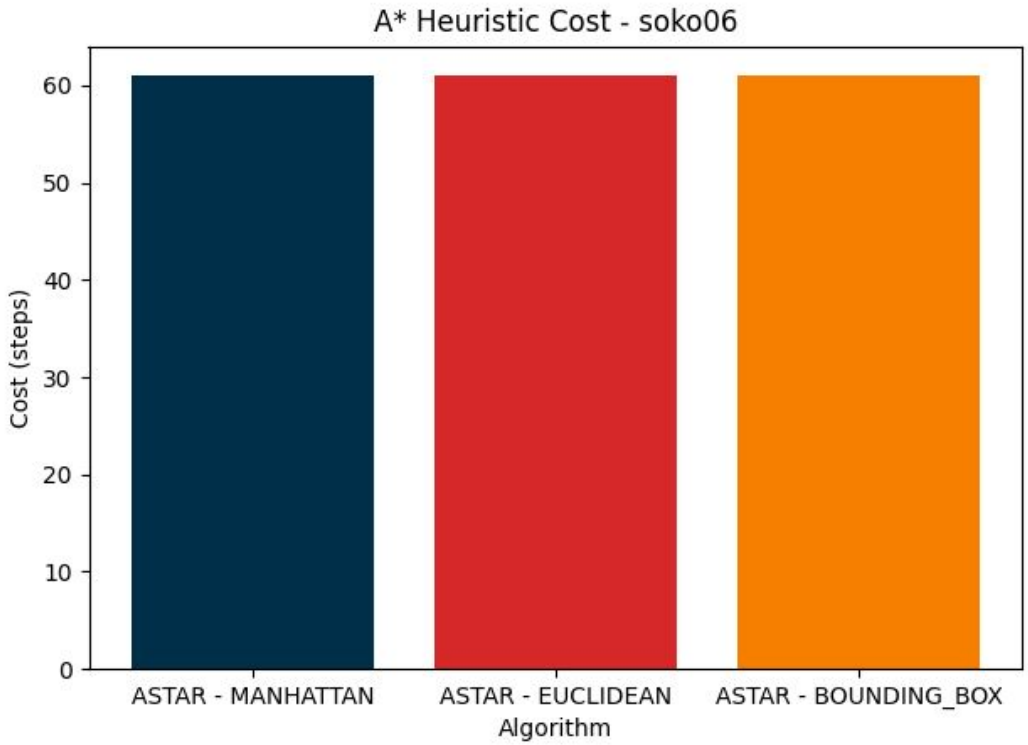
Greedy: Comparación de Heurísticas

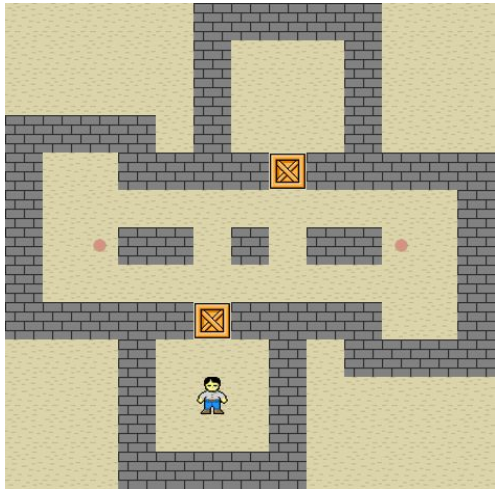




Repeticiones: 10
Tablero: soko06
Algoritmo: Greedy
Heurística: *

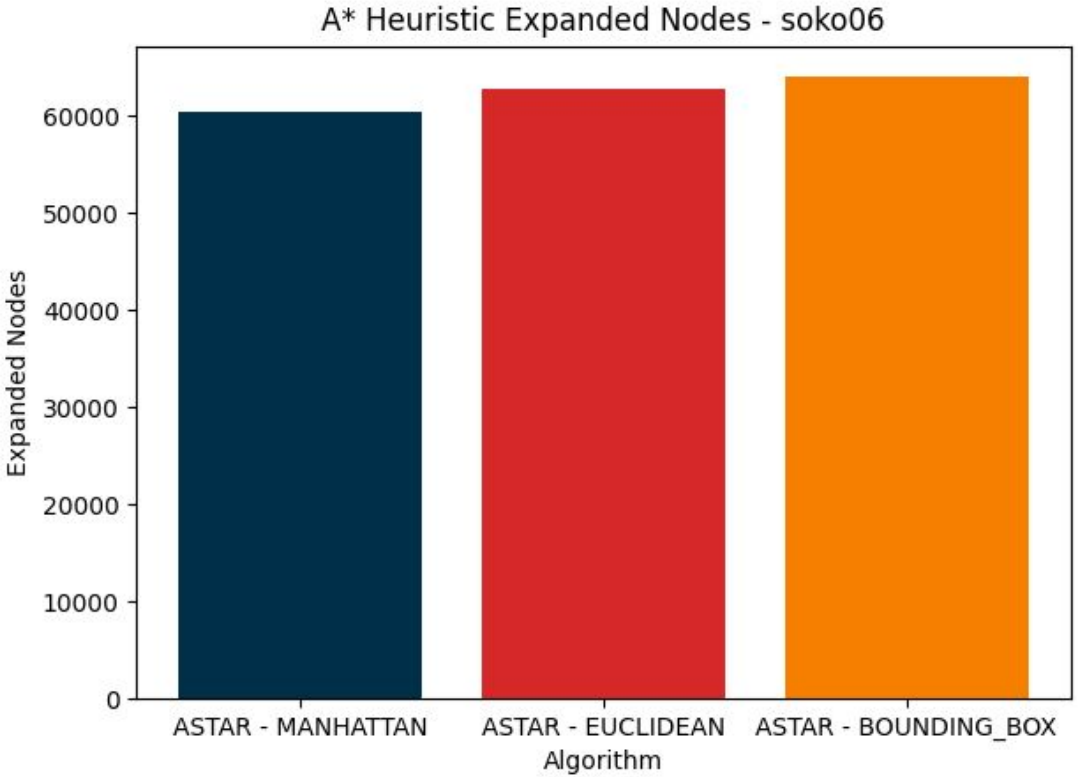
A*: Comparación de Heurísticas

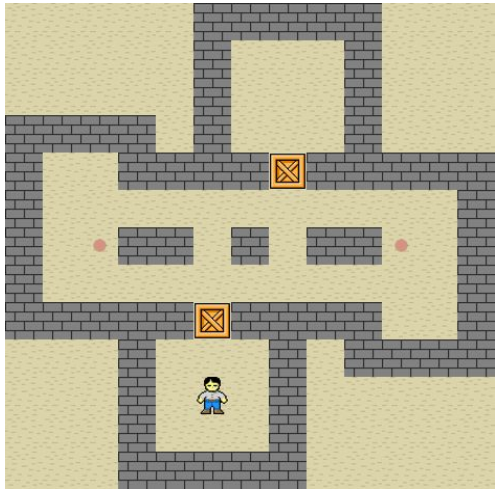




Repeticiones: 10
Tablero: soko06
Algoritmo: A*
Heurística: *

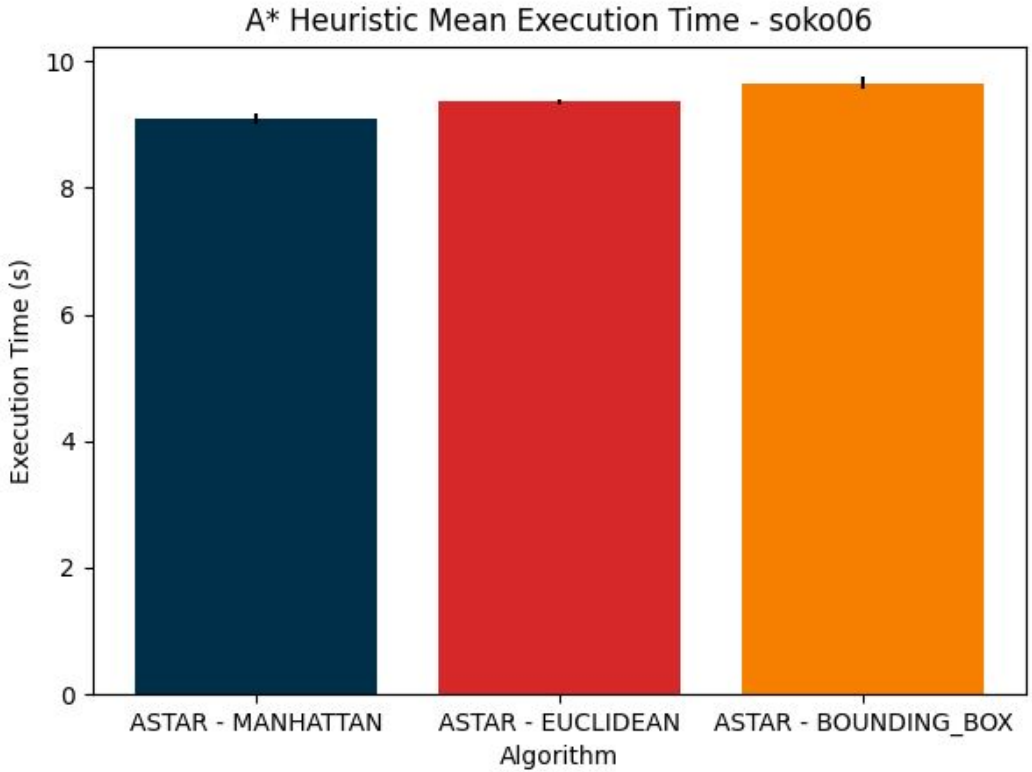
A*: Comparación de Heurísticas

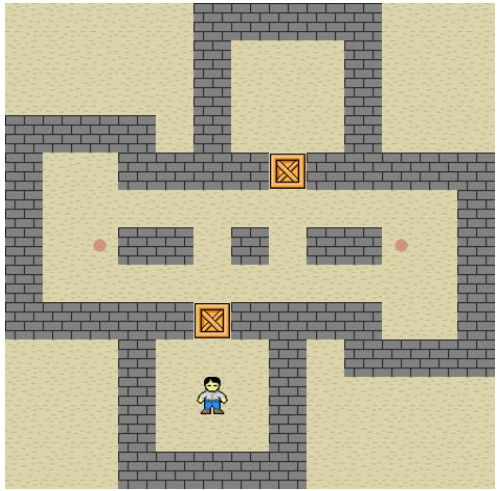




Repeticiones: 10
Tablero: soko06
Algoritmo: A*
Heurística: *

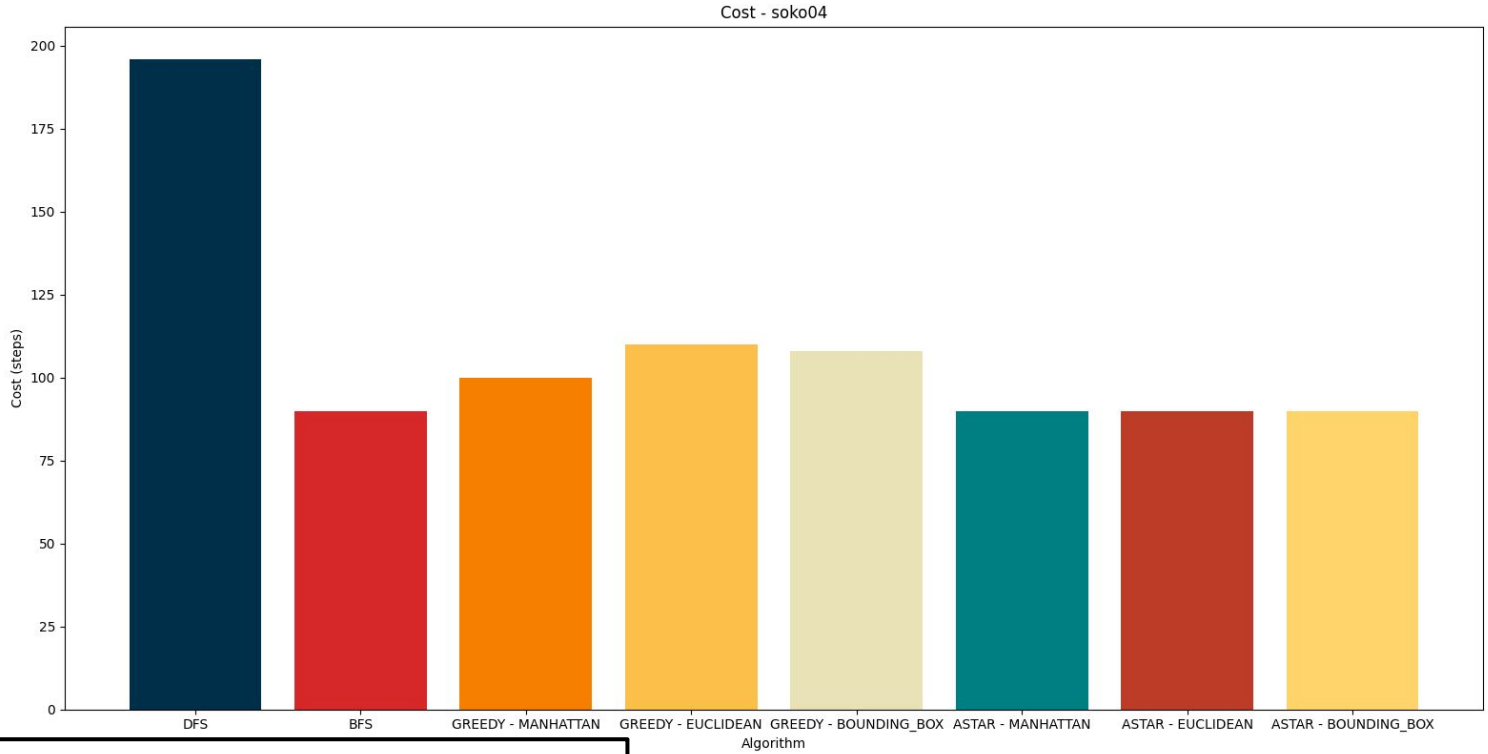
A*: Comparación de Heurísticas





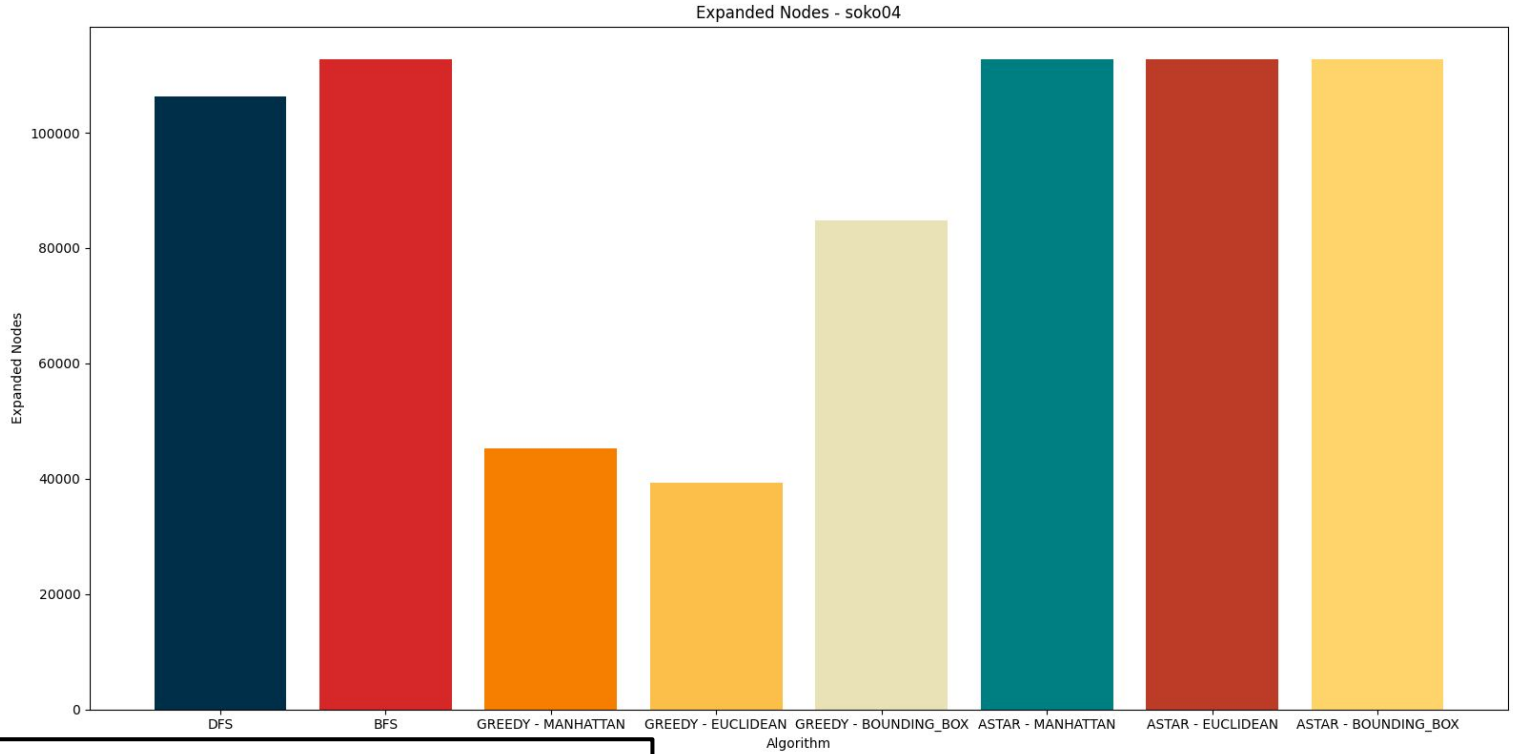
Repeticiones: 10
Tablero: soko06
Algoritmo: A*
Heurística: *

Comparación de Algoritmos: Costo



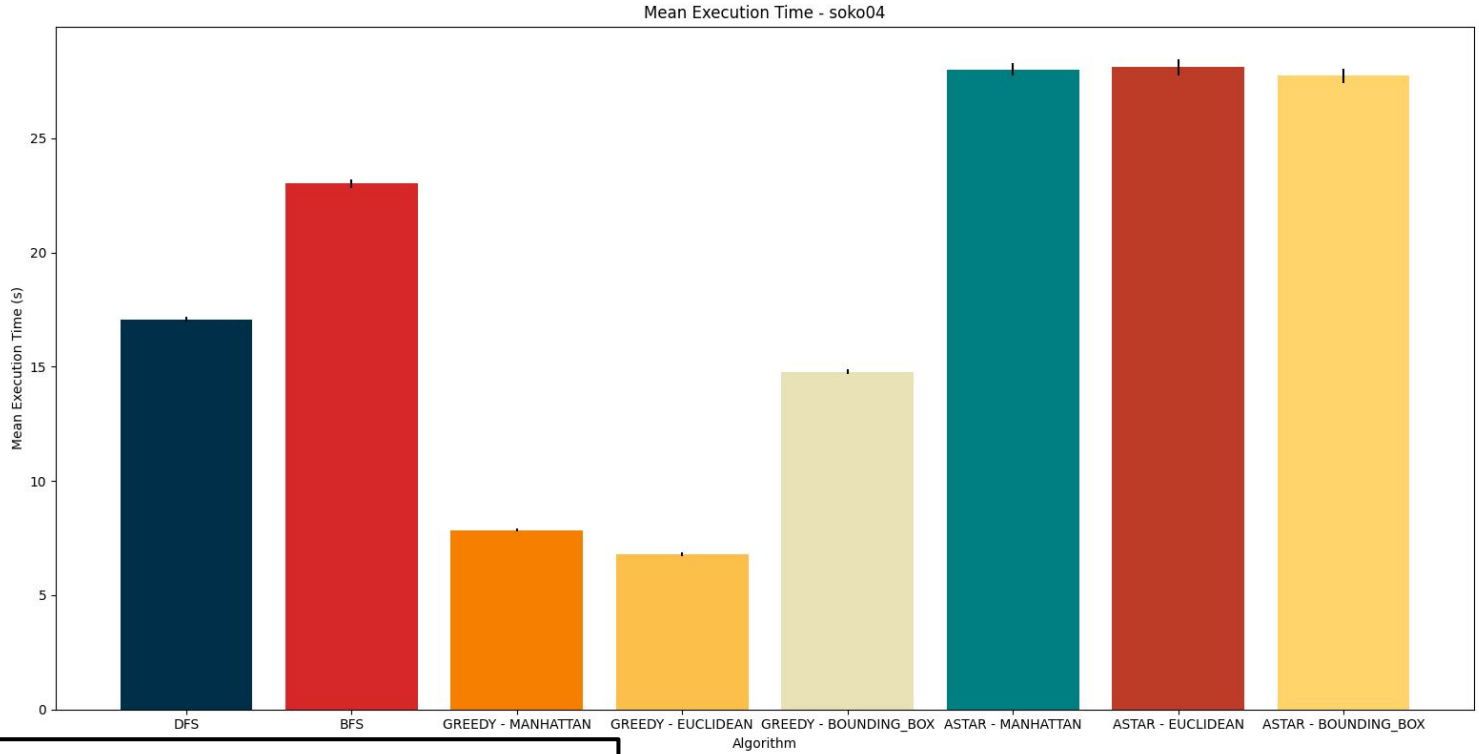
Repeticiones: 10 | Tablero: soko04

Comparación de Algoritmos: Nodos



Repeticiones: 10 | Tablero: soko04

Comparación de Algoritmos: Tiempo



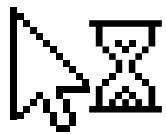
Repeticiones: 10 | Tablero: soko04

Conclusiones

Los métodos de búsqueda informados superan a los métodos de búsqueda desinformados.

Notamos que las heurísticas admisibles como Manhattan y Euclidean son superiores a las heurísticas no admisibles.

Por último, podemos visualizar que hay una relación proporcional entre el tiempo y los nodos expandidos.



¿Cuál es el mejor método de búsqueda?

DEPENDE.

- A* encuentra la solución más óptima pero requiere de más recursos.
- Greedy hace un buen balance de costo-beneficio.
- La heurística Manhattan resulta superadora a la Euclídea.
- DFS es ideal para tableros con árboles con alta ariedad.

¡Muchas gracias por su atención!

Enlaces Útiles

- https://github.com/alejofl/sia
  Repositorio del proyecto

- http://www.game-sokoban.com/index.php?mode=catalog
  Niveles del Sokoban