Sistemas de Inteligencia Artificial

TP1 - Métodos de Búsqueda

Integrantes del grupo 6

Boullosa Gutierrez, Juan Cruz
63414

Deyheralde, Ben
63559

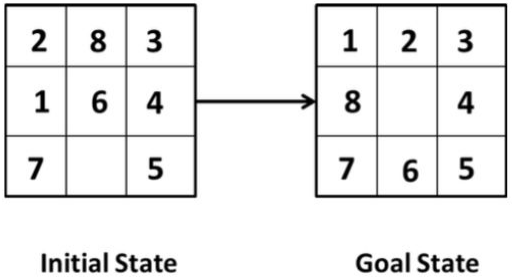
Freire, Tomás
62027

Stanfield, Theo
63403

Ejercicio 1: 8-Puzzle

Características del 8-Puzzle

- **Estados:** Cada estado es una configuración del tablero representado por una **matriz 3x3**.
- **Acciones posibles:** Las celdas pueden moverse hacia arriba, abajo, derecha o izquierda
- **Estado target:** Las celdas tienen que quedar en forma ordenada
- **Costo del movimiento:** Cada movimiento de celda tiene un costo = 1



Heurísticas admisibles no-triviales

Distancia de Manhattan:

- Calcula las distancias horizontales y verticales que cada ficha debe recorrer desde su posición actual hasta el objetivo.

$$h_1(n) = \sum_{i=1}^{8}(|x_i - x_i^*| + |y_i - y_i^*|)$$

Conflicto Lineal + Manhattan:

- Mejora la precisión de la Manhattan
- Penaliza casos donde dos fichas se encuentran en la misma fila o columna que su posición objetivo pero en orden invertido.

$$h_2(n) = h_{\text{Manhattan}}(n) + 2 \times (\text{número de conflictos lineales})$$

Método de búsqueda y heurística.

Algoritmo óptimo: **A*** en combinación con la heurística de **conflicto lineal + distancia Manhattan**

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

Combina el costo real acumulado **(g(n))** y una estimación del costo restante **(h(n))**. Dado que la heurística es admisible se priorizan nodos con menor costo estimado total. Se garantiza la expansión de menor cantidad de nodos que usando un algoritmo desinformado, por ende mayor eficiencia.

Ejercicio 2: Sokoban

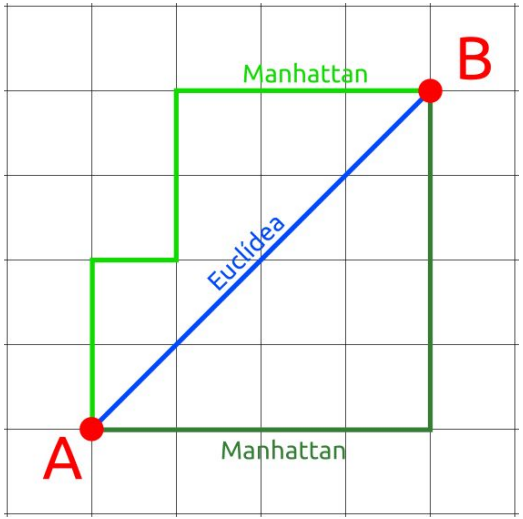
Estructura de datos

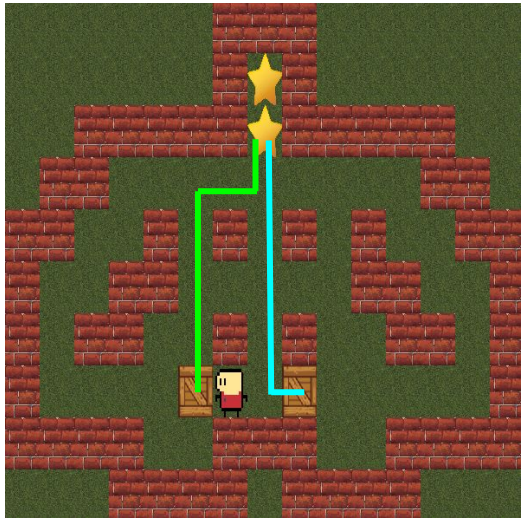
- Estructura de estados:
  - posición del jugador
  - posición de las cajas
- Mapa del nivel con las posiciones de las paredes y objetivos

```
class State:
    def __init__(self, player, boxes, move=None, parent=None):
        self.player = player
        self.boxes = boxes
        self.move = move
        self.parent = parent
```

Heurísticas

Distancia Manhattan



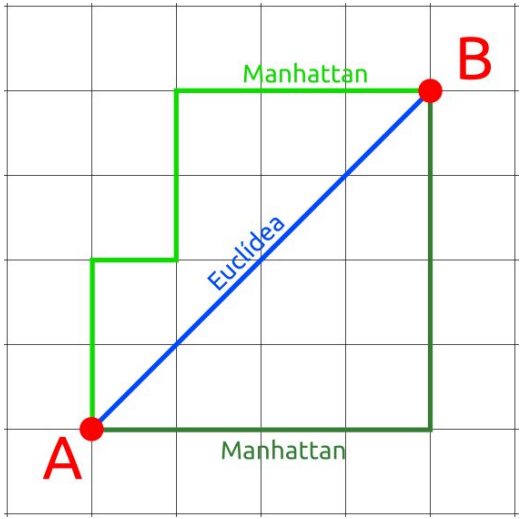


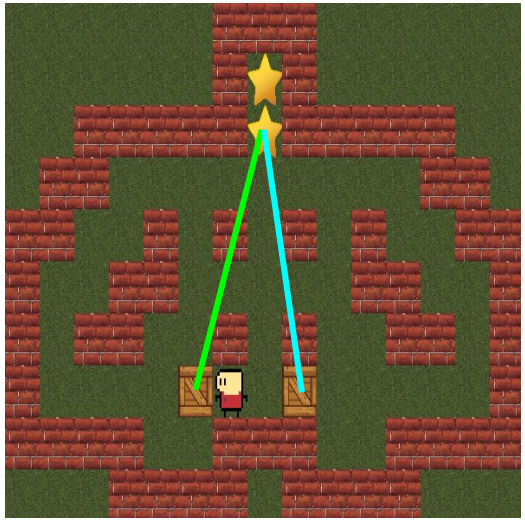
Caja izquierda: 7

Caja derecha: 6

h(n): 13

Distancia Euclidiana





Caja izquierda:

$$\sqrt{5^2 + 2^2} = \sqrt{29} = 5.38$$

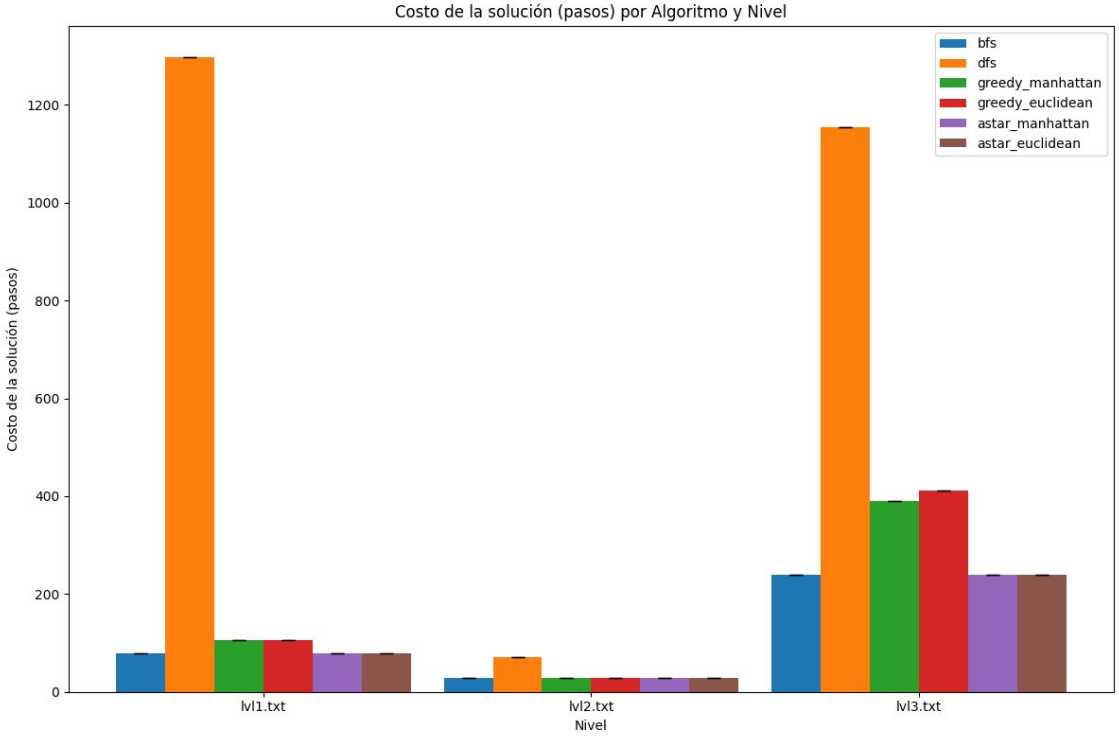
Caja derecha:

$$\sqrt{5^2 + 1^2} = \sqrt{28} = 5.1$$

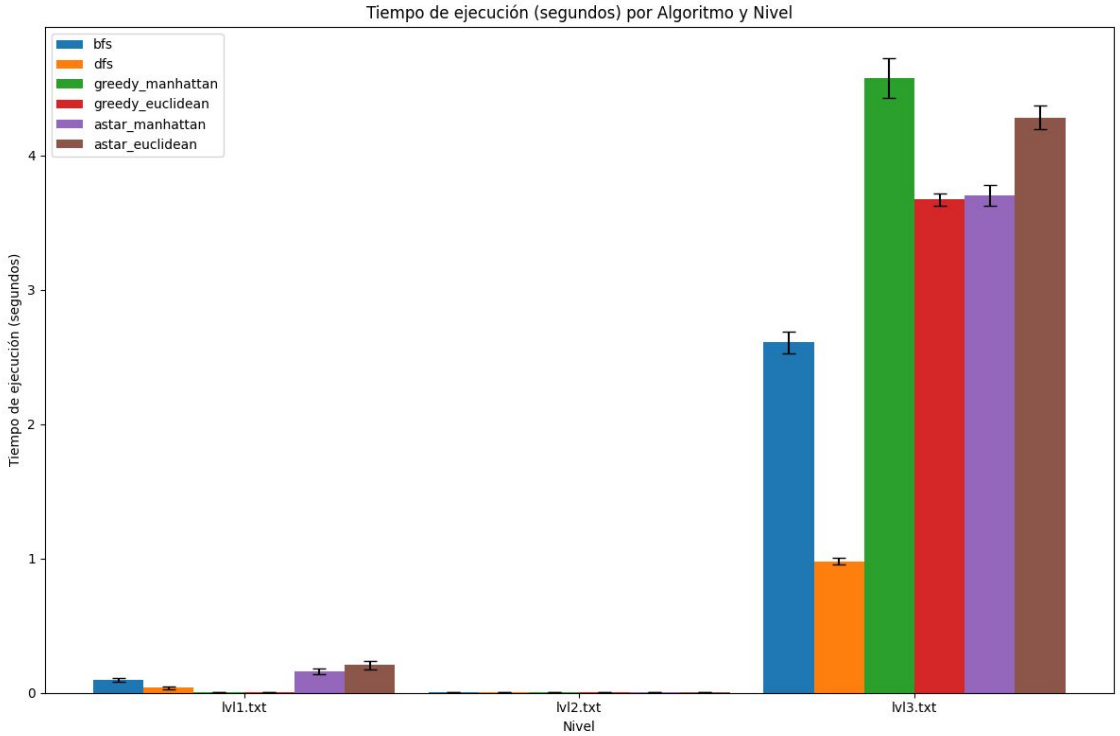
h(n): 10.48

Resultados

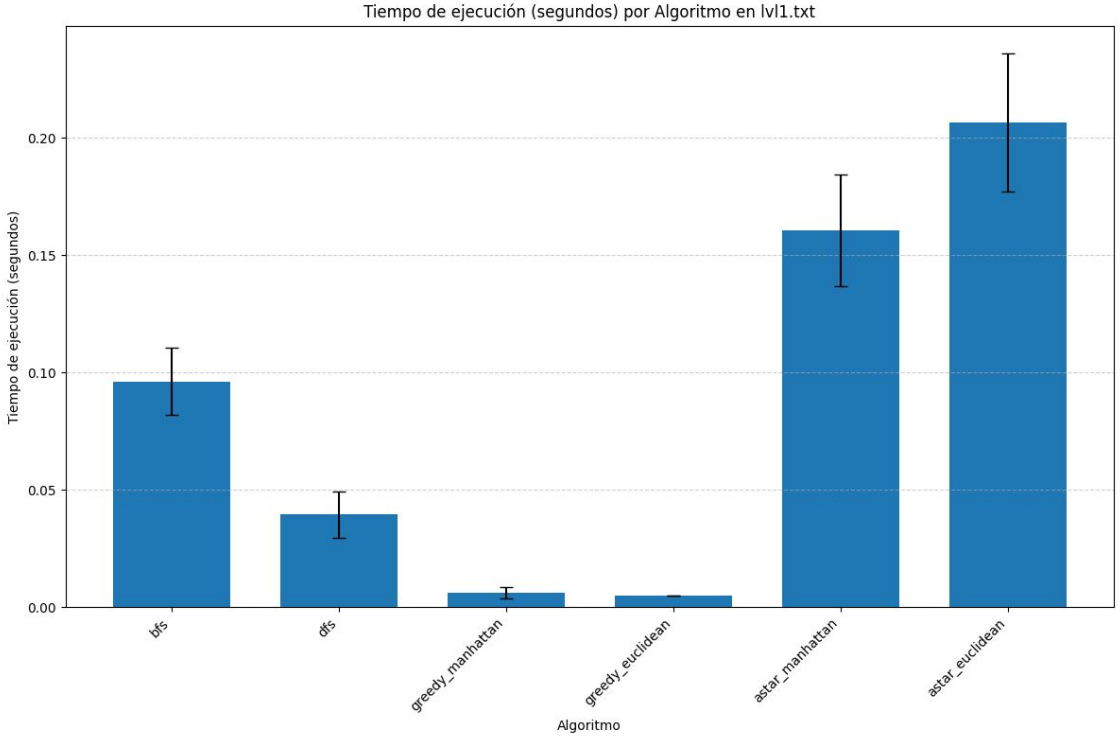
Longitud de la solución



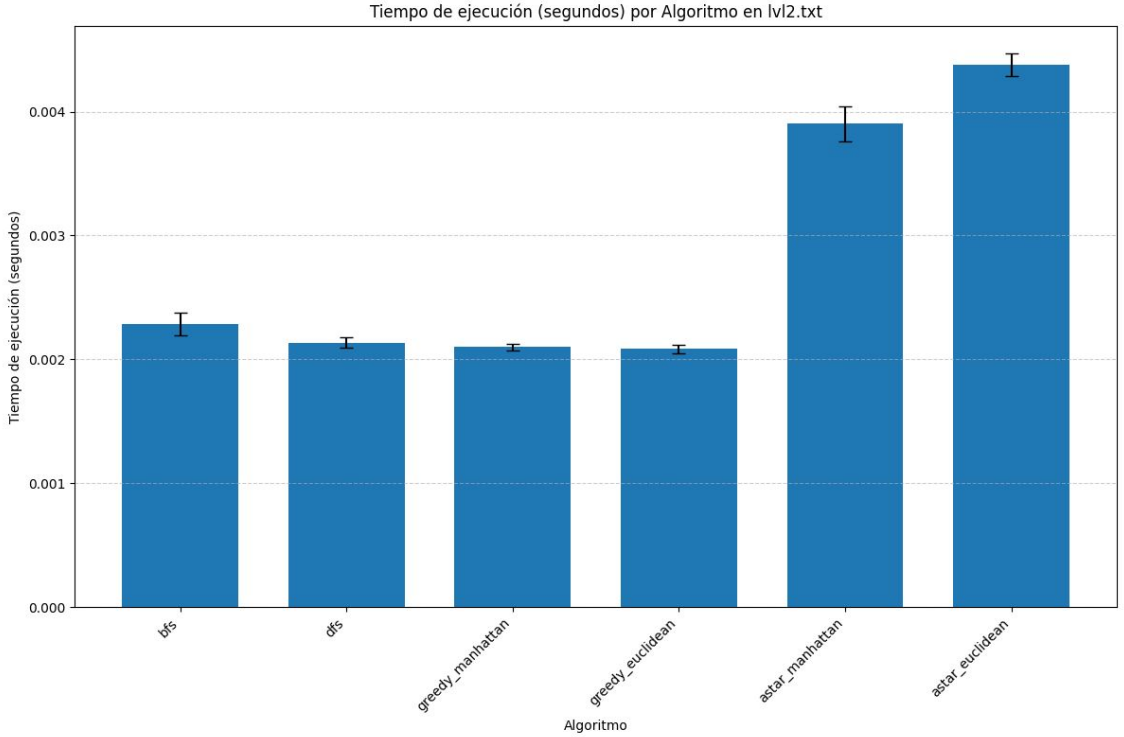
Tiempo de ejecución



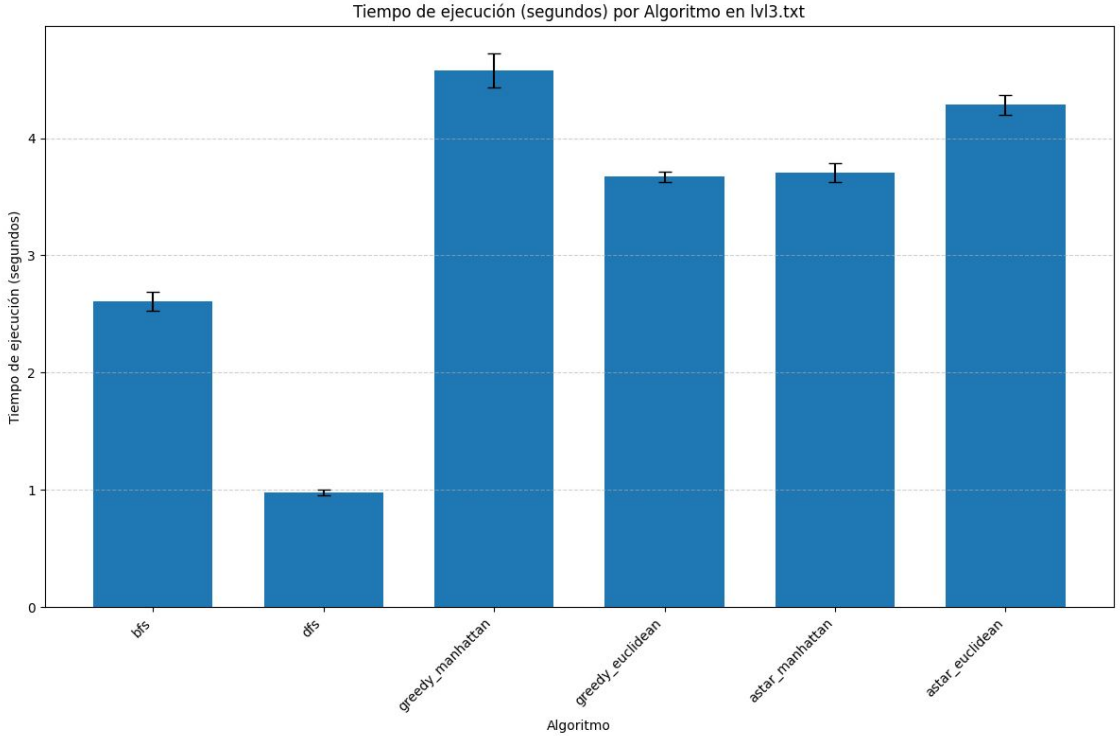
Tiempo de ejecución - Nivel 1



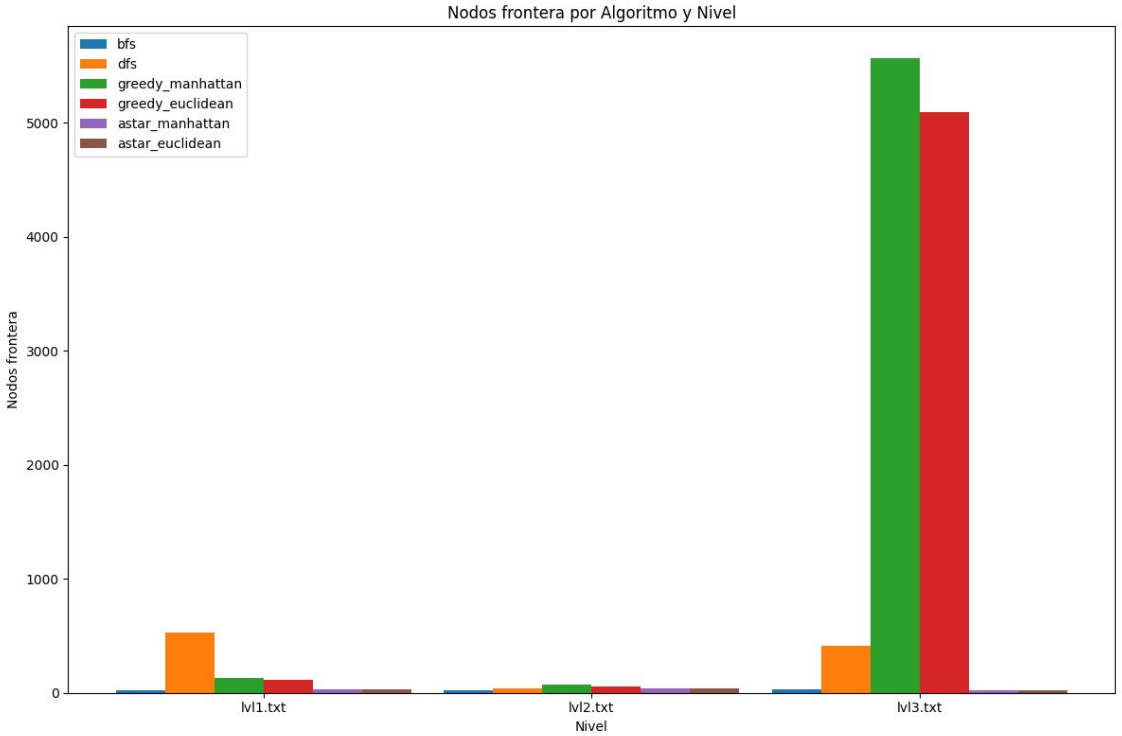
Tiempo de ejecución - Nivel 2



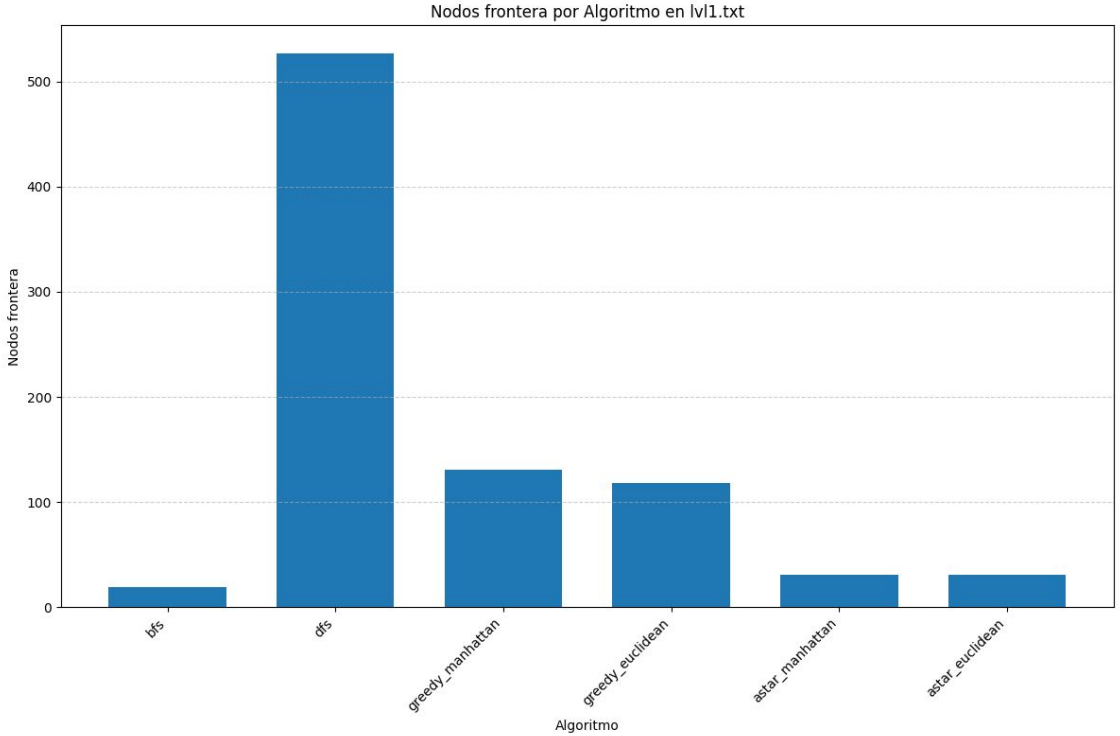
Tiempo de ejecución - Nivel 3



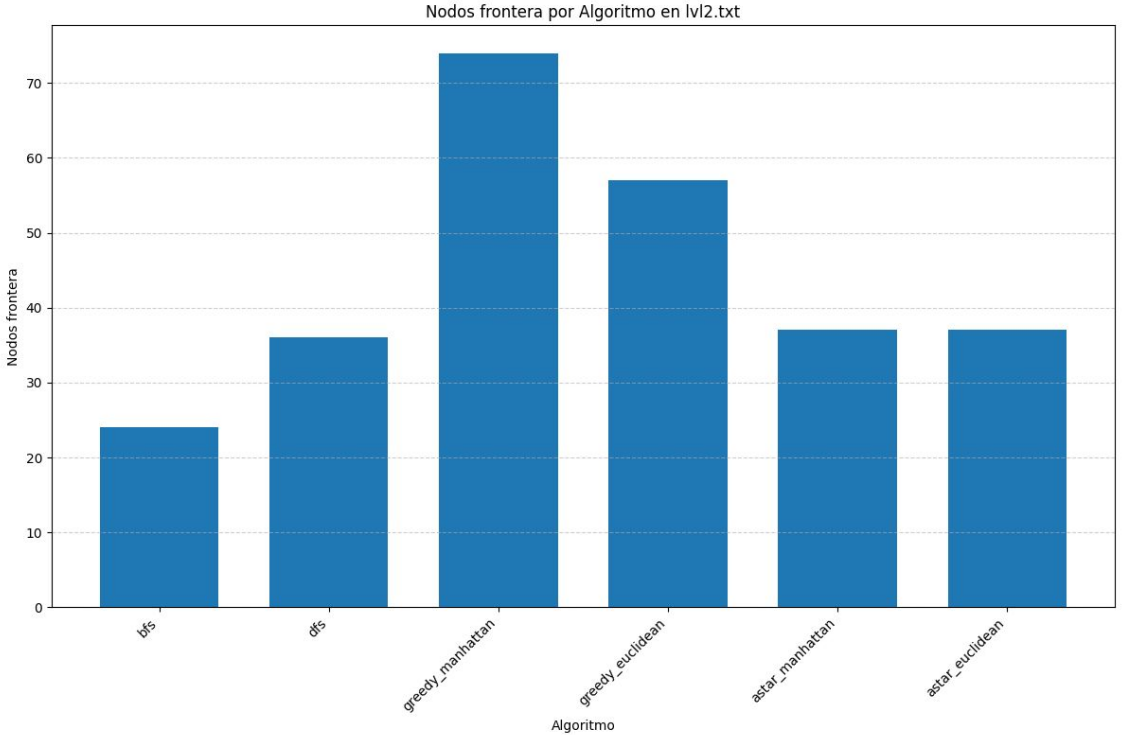
Cantidad de nodos frontera



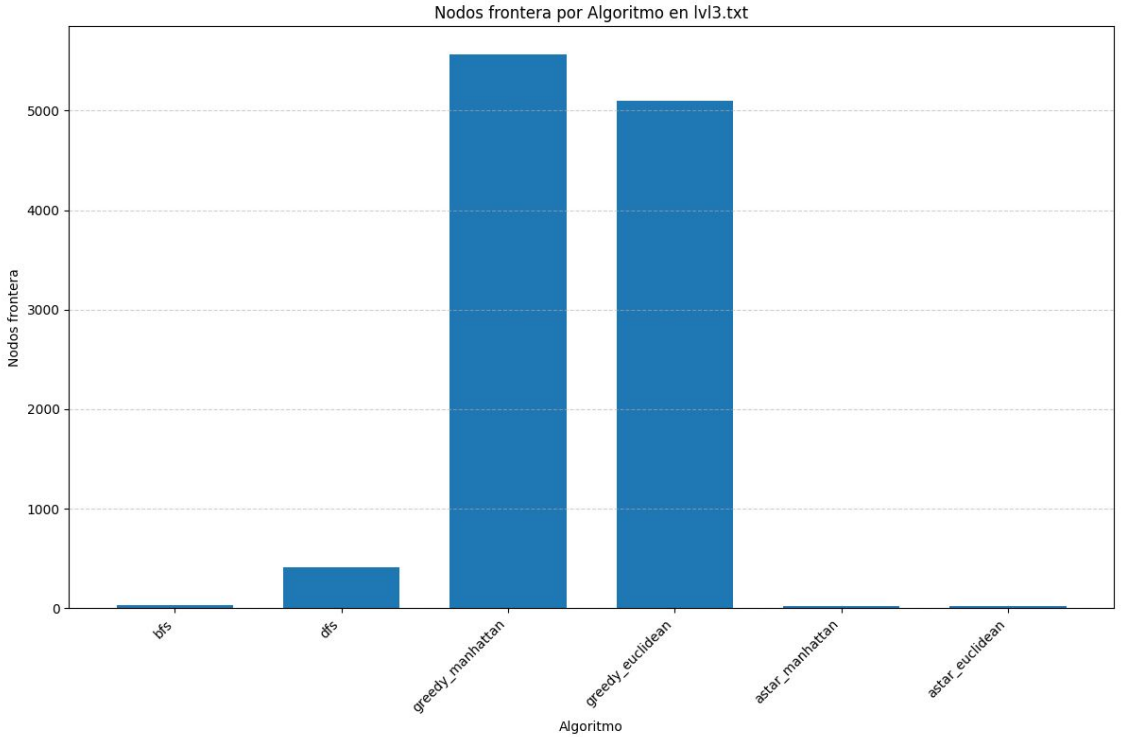
Cantidad de nodos frontera - Nivel 1



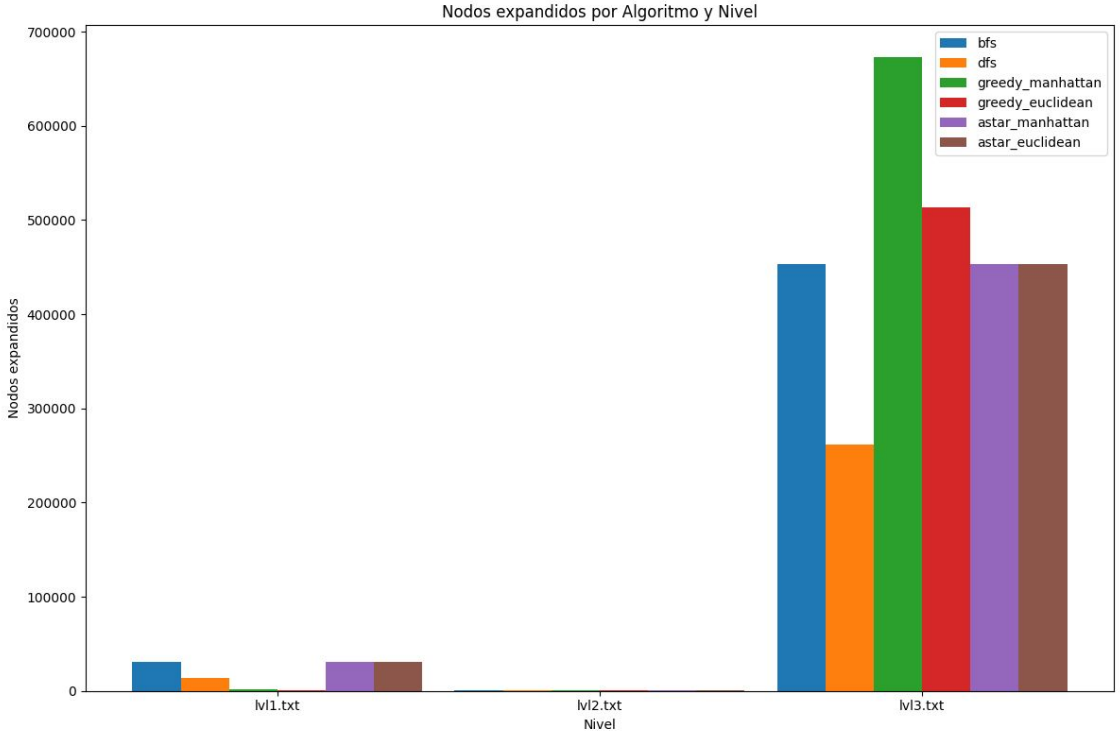
Cantidad de nodos frontera - Nivel 2



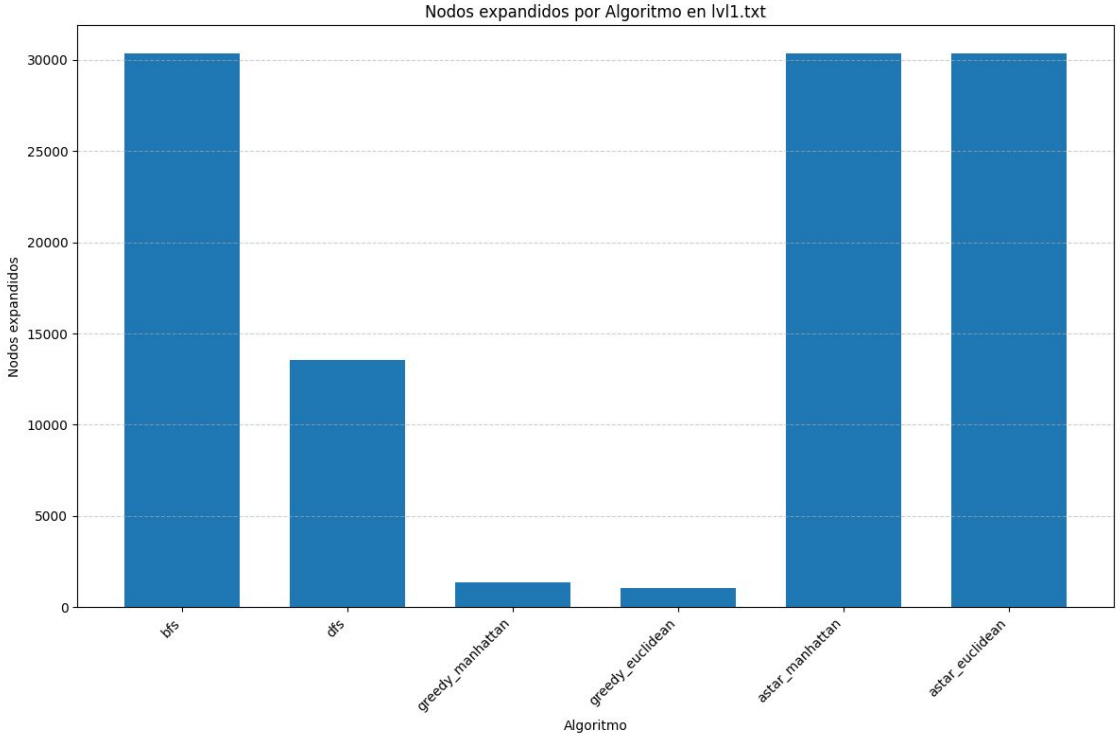
Cantidad de nodos frontera - Nivel 3



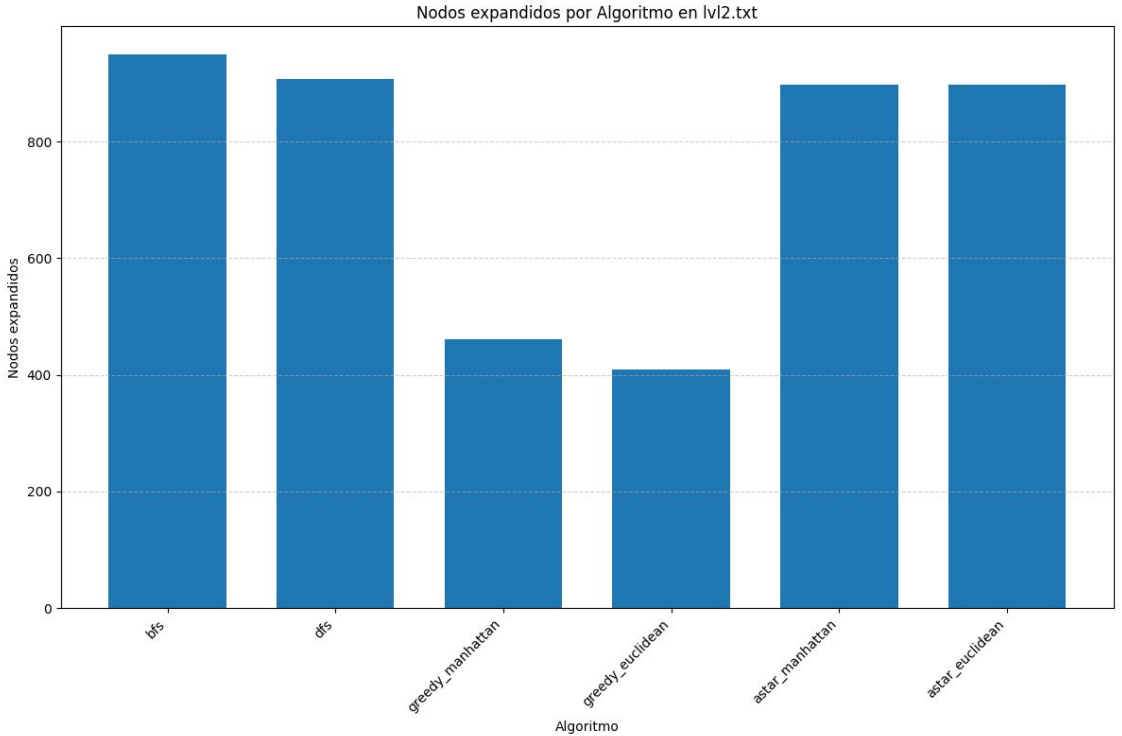
Cantidad de nodos expandidos



Cantidad de nodos expandidos - Nivel 1



Cantidad de nodos expandidos - Nivel 2



Cantidad de nodos expandidos - Nivel 3

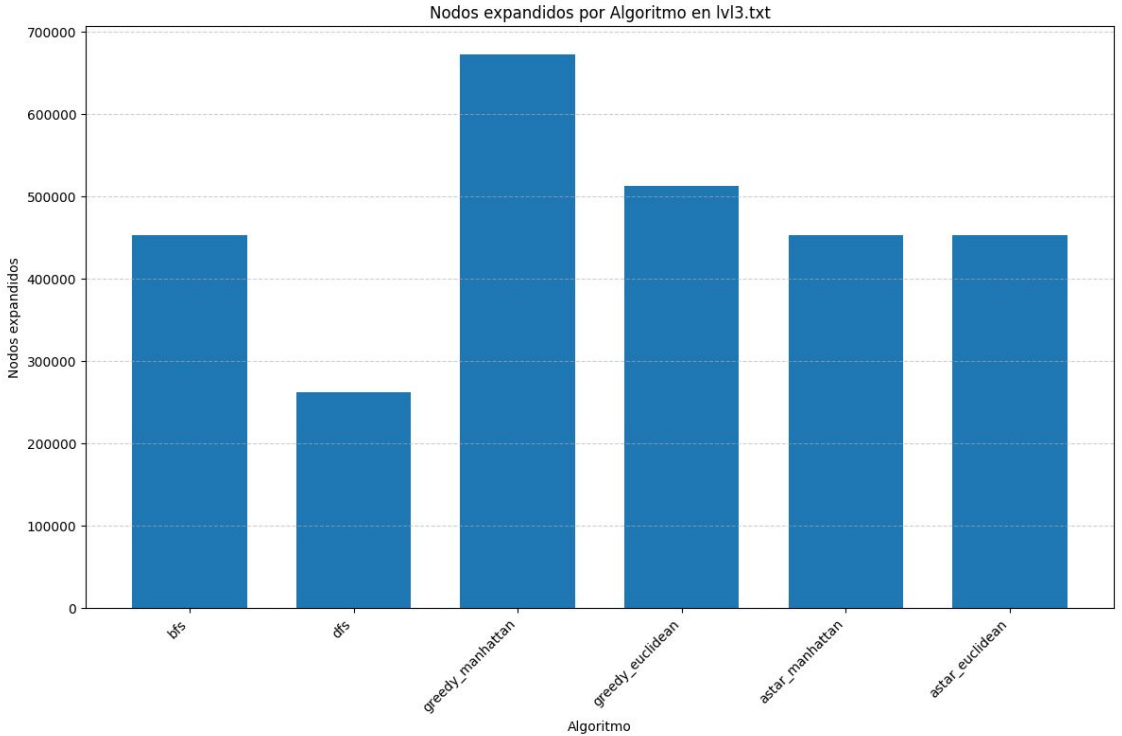


Tabla Comparativa

Comparación de performance por algoritmo

Nivel	Algoritmo	Tiempo promedio (s)	Desvío estándar	Costo promedio	Desvío estándar	Nodos exp. prom.	Nodos front. prom.
lvl1.txt	bfs	0.0961	0.0143	79.00	0.00	30377.00	19.00
lvl1.txt	dfs	0.0395	0.0098	1297.00	0.00	13536.00	527.00
lvl1.txt	greedy_manhattan	0.0060	0.0024	105.00	0.00	1380.00	131.00
lvl1.txt	greedy_euclidean	0.0049	0.0001	105.00	0.00	1028.00	118.00
lvl1.txt	astar_manhattan	0.1606	0.0236	79.00	0.00	30337.00	31.00
lvl1.txt	astar_euclidean	0.2066	0.0293	79.00	0.00	30339.00	31.00
lvl2.txt	bfs	0.0023	0.0001	29.00	0.00	950.00	24.00
lvl2.txt	dfs	0.0021	0.0000	72.00	0.00	907.00	36.00
lvl2.txt	greedy_manhattan	0.0021	0.0000	29.00	0.00	461.00	74.00
lvl2.txt	greedy_euclidean	0.0021	0.0000	29.00	0.00	410.00	57.00
lvl2.txt	astar_manhattan	0.0039	0.0001	29.00	0.00	898.00	37.00
lvl2.txt	astar_euclidean	0.0044	0.0001	29.00	0.00	898.00	37.00
lvl3.txt	bfs	2.6084	0.0830	239.00	0.00	453070.00	32.00
lvl3.txt	dfs	0.9794	0.0231	1155.00	0.00	261895.00	410.00
lvl3.txt	greedy_manhattan	4.5751	0.1463	391.00	0.00	673117.00	5569.00
lvl3.txt	greedy_euclidean	3.6719	0.0454	411.00	0.00	513042.00	5095.00
lvl3.txt	astar_manhattan	3.7047	0.0784	239.00	0.00	452977.00	25.00
lvl3.txt	astar_euclidean	4.2829	0.0879	239.00	0.00	452978.00	26.00

Conclusiones

- **Greedy Search** muy ineficiente. Hace muchos movimientos erráticos porque solo considera el paso que mejore la heurística.

- **BFS** es mejor que **A*** en mapas chicos por reducir el costo computacional. Sin embargo, en mapas grandes **A*** dio resultados hasta 20% más rápidos que **BFS**. Estimamos entonces que para mapas más grandes **A*** da mejores resultados.

- **DFS** y **Greedy** suelen hallar una solución mucho más rápido, pero realizando muchos más movimientos.

- Por lo tanto, podemos concluir que se debe elegir el algoritmo a usar en función de si se quiere hallar una solución con mayor velocidad o si es preferible hallar la solución con menor cantidad de movimientos.