Métodos de Búsqueda

ITBA - SIA - 2021

Introducción

- El trabajo práctico fue implementado utilizando Python. Para el trabajo de renderizado del nivel se tomo la implementación de Sokoban de Gemkodor.

- Se consideraron tres niveles sobre los cuales se realizaron los análisis del tiempo y el espacio ocupado.

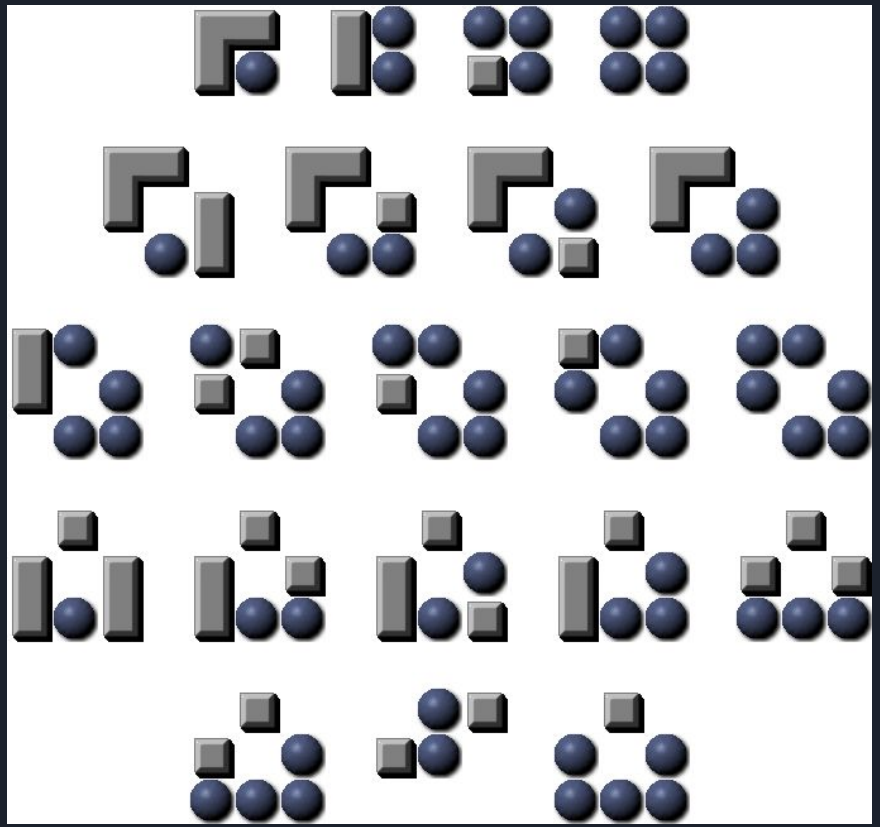
- El sistema puede reconocer cualquier nivel siempre que utilice los caracteres correspondientes. También reconoce cuando un sistema es imposible de superar

Consideraciones

- Para todos los métodos implementados se filtran los estados “visitados”. Estos se analizan en base a la posición del jugador y la posición de todas las cajas. De este modo, un nodo que posea un estado que ya tuvo otro que fue explotado previamente, no es explotado.

- Para todos los métodos se filtraron los estados en los que alguna de las cajas haya llegado a una esquina, ya que se lo considera un estado de exito imposible. Próximas implementaciones podrían tener en cuenta aun mas de estos estados.

Ejemplos de Deadlocks



Métodos de búsqueda

Métodos desinformados considerados:

- DFS
- BFS
- IDDFS

Métodos informados considerados:

- Greedy
- A*
- IDA

Heurísticas

- **Open Goal**: Cantidad de objetivos vacíos restantes
- **Player-Box Distance**: Distancia mínima del jugador a alguna caja.
- **Target-Box Distance**: Suma de las distancias mínimas de los objetivos a la caja más cercana.
- **Unique Target-Box**: Igual que **Target-Box Distance**, pero emparejando targets y cajas sin repetir

Todas estas heurísticas resultan admisibles. La idea fue centrarse únicamente en las propiedades que toda solución debe tener.

Por ejemplo, ninguna tiene en cuenta las paredes o deadlocks. En el caso de **Target-Box Distance** se asume que nunca habrá que sacar una caja de un objetivo para llegar a la solución.

Explicación de las heurísticas

Tomamos **Open Goal** como la heurística base para medir la performance del resto, no puede ser más simple.

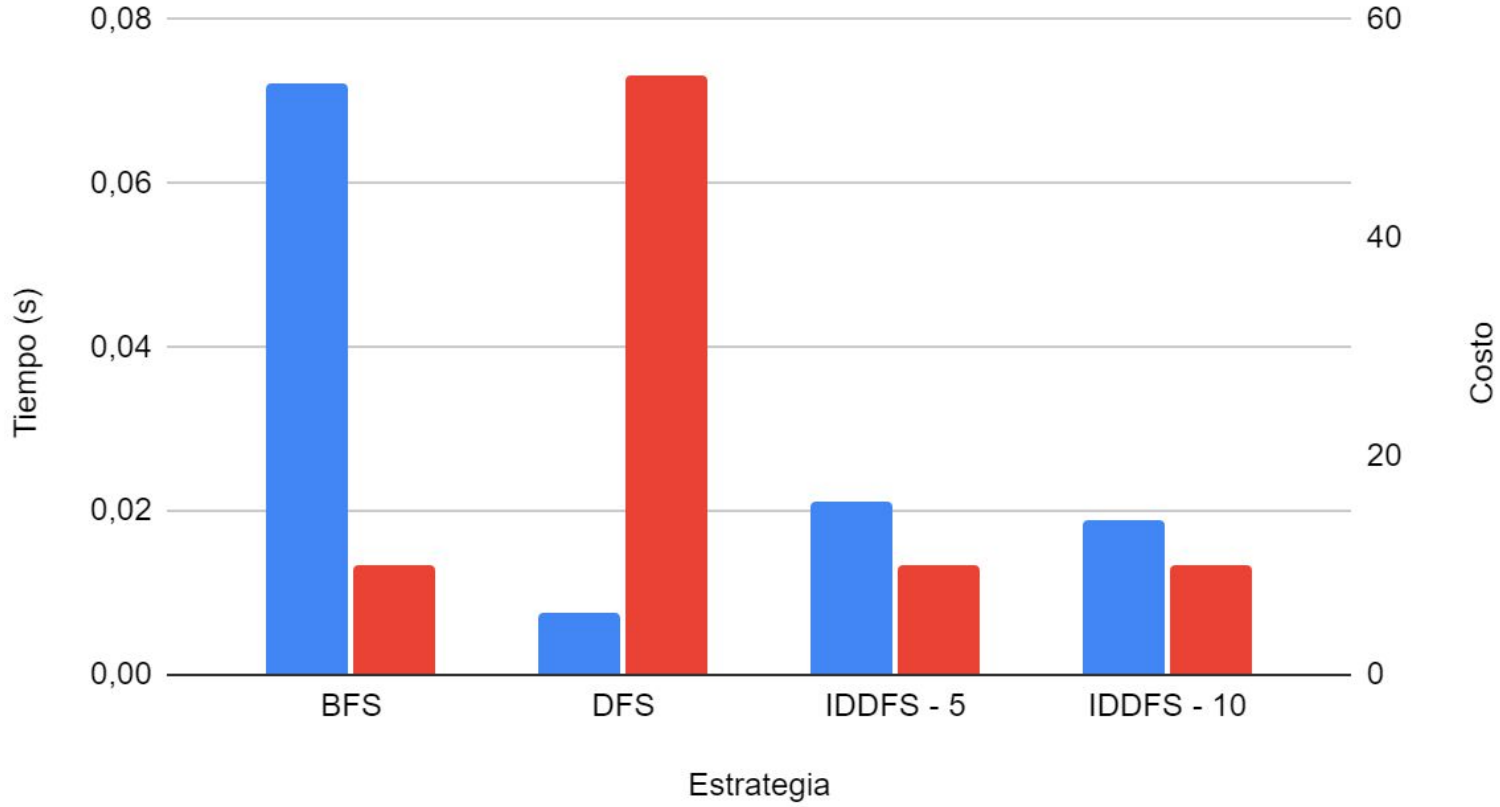
Luego consideramos que el jugador alternaba entre cajas en lugar de centrarse en una sola. **Player-Box Distance** lo obliga a mover una misma caja, favoreciendo que se “acerque” a esta.

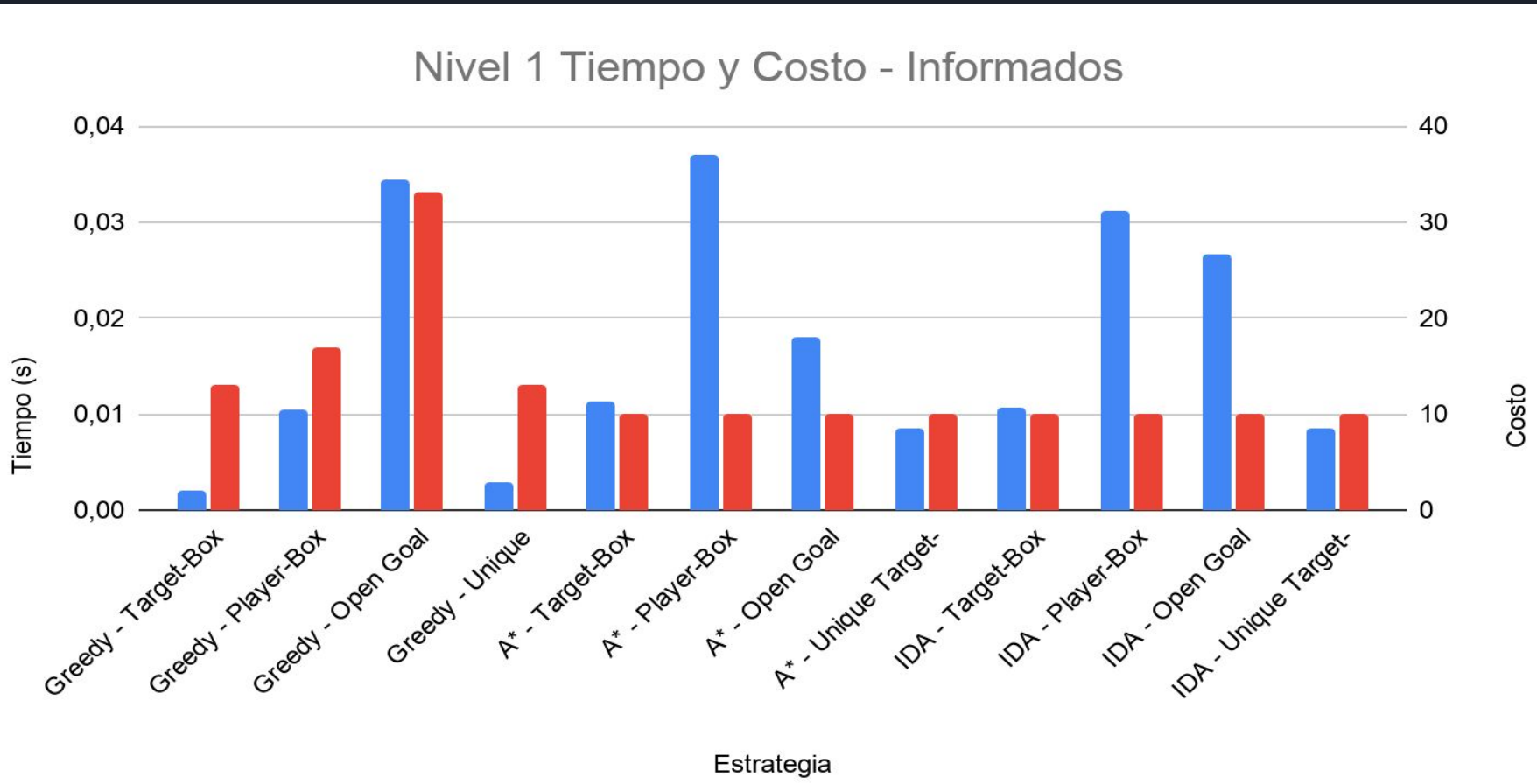
Target-Box Distance busca minimizar la distancia entre cajas y objetivo, algo presente en toda solución. En niveles donde aparecen múltiples targets cercanos a una misma caja, esta heurística empieza a fallar

A partir de esto, **Unique Target-Box** busca asignar de manera óptima cada caja con un objetivo sin repetir. En principio, esperábamos que esta fuese la heurística más performante.

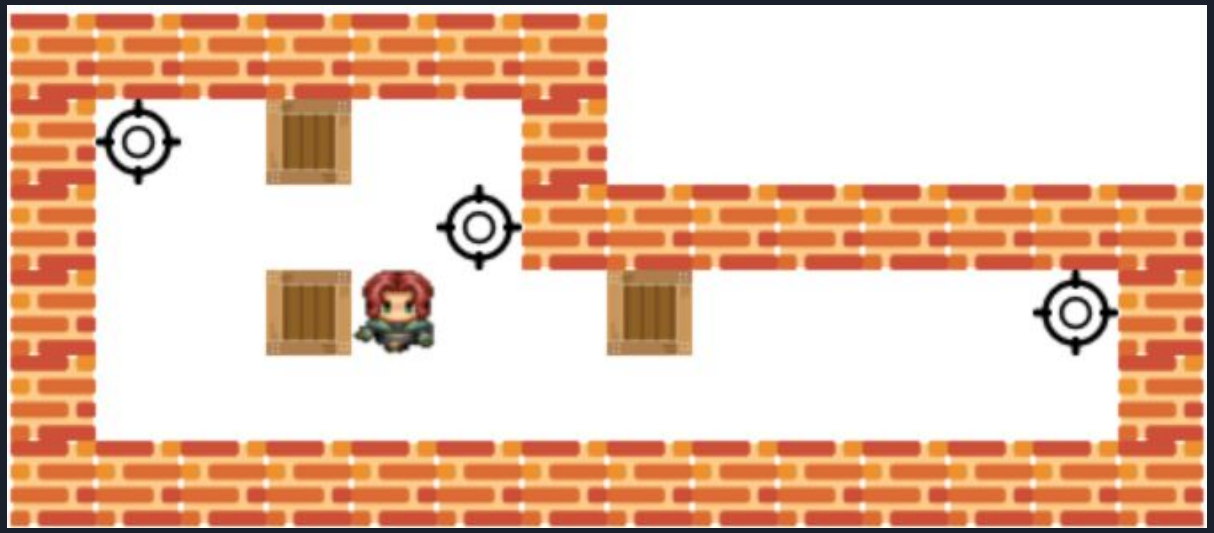
Nivel 1:

Nivel 1 Tiempo y Costo - Desinformado

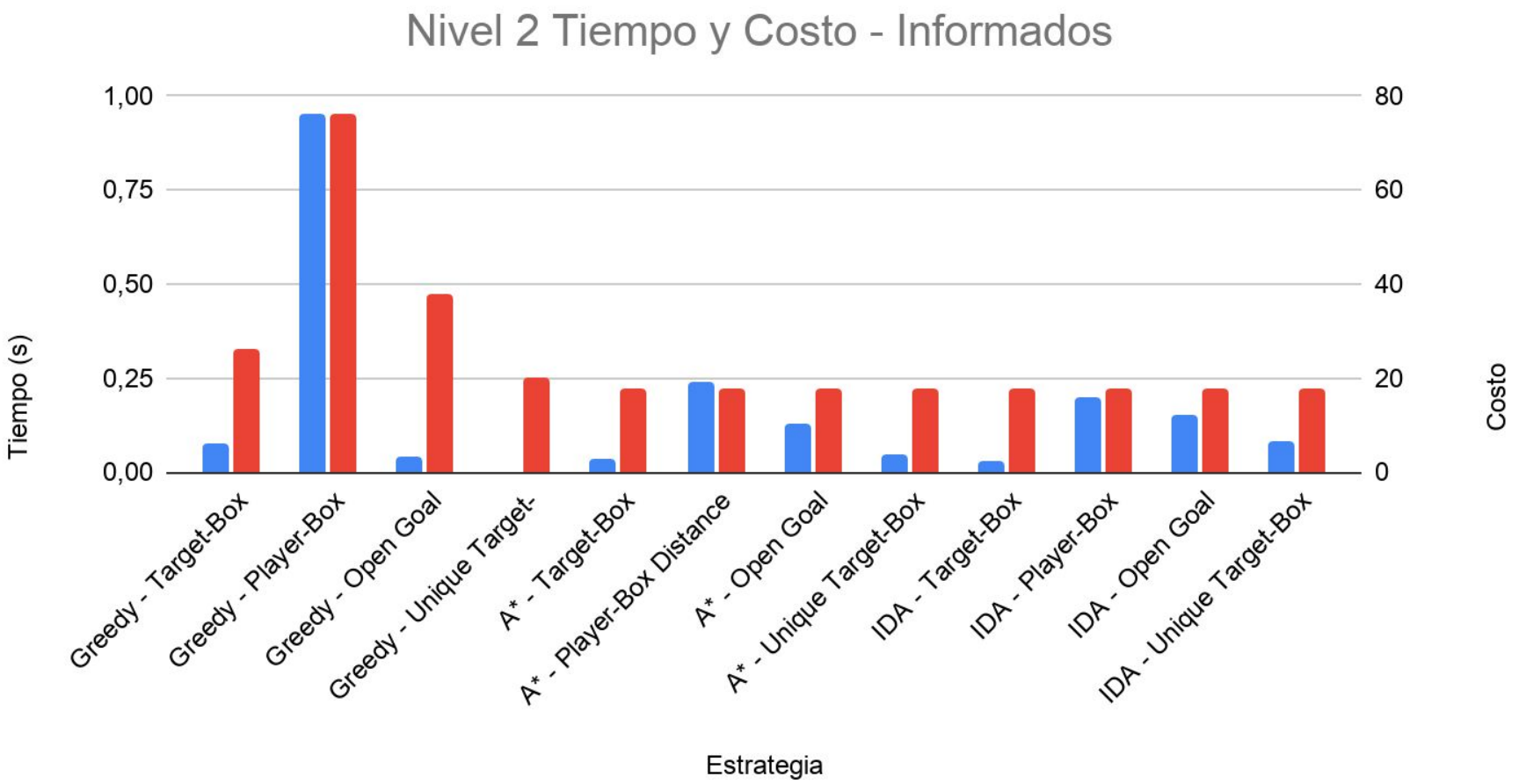



Nivel 1 Tiempo y Costo - Informados
0,04
0,03
0,02
0,01
0,00
Tiempo (s)
40
30
20
10
0
Costo
Greedy - Target-Box
Greedy - Player-Box
Greedy - Open Goal
Greedy - Unique
A* - Target-Box
A* - Player-Box
A* - Open Goal
A* - Unique Target-
IDA - Target-Box
IDA - Player-Box
IDA - Open Goal
IDA - Unique Target-
Estrategia

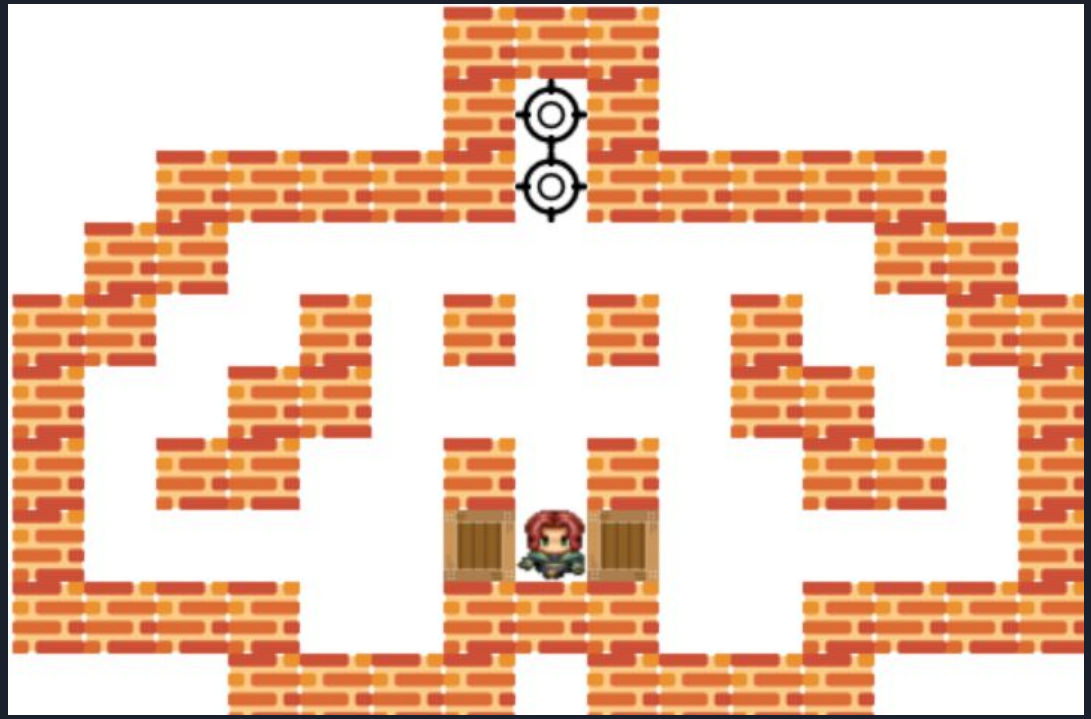
Resultados nivel 2:




Nivel 2 Tiempo y Costo - Desinformados
0,60
0,40
0,20
0,00
Tiempo (s)
150
100
50
0
Costo
BFS
DFS
IDDFS - 5
IDDFS - 10
Estrategia


Nivel 2 Tiempo y Costo - Informados
1,00
0,75
0,50
0,25
0,00
Tiempo (s)
80
60
40
20
0
Costo
Greedy - Target-Box
Greedy - Player-Box
Greedy - Open Goal
Greedy - Unique Target-
A* - Target-Box
A* - Player-Box Distance
A* - Open Goal
A* - Unique Target-Box
IDA - Target-Box
IDA - Player-Box
IDA - Open Goal
IDA - Unique Target-Box
Estrategia

Nivel 3

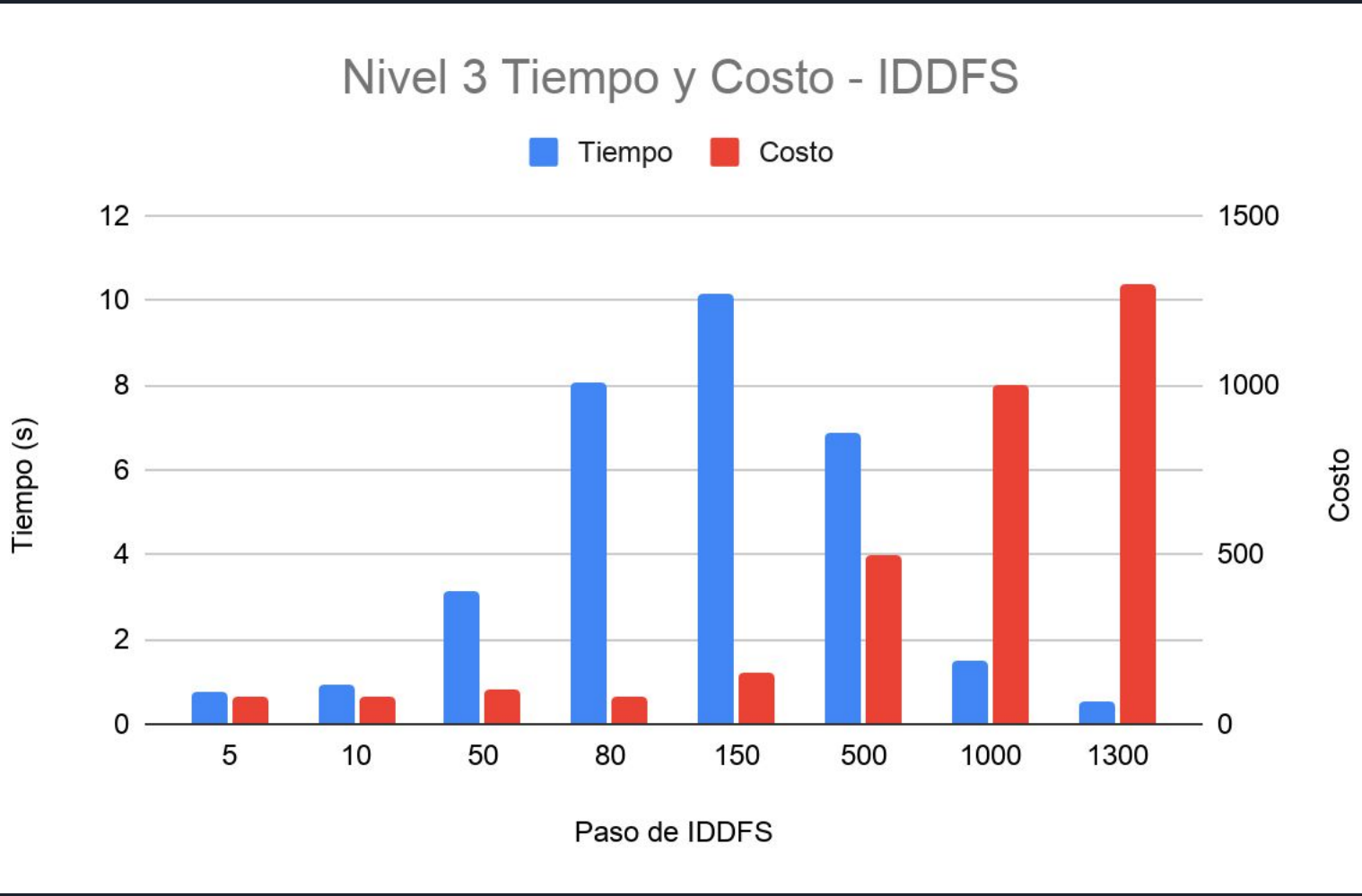



Nivel 3 Tiempo y Costo - Desinformados
Tiempo
Costo
1,5
1,0
0,5
0,0
Tiempo (s)
1500
1000
500
0
Costo
BFS
IDDFS-5
IDDFS-10
IDDFS-1000
IDDFS-1300
DFS
Estrategia

Métodos Desinformados

Como pudo comprobarse en los gráficos anteriores, se cumple lo que la teoría indica. En primer lugar y dado que el problema es de costo uniforme, el método **BFS** termina encontrando la solución **óptima**, sin embargo, este resulta **lento** para hallarla y consume una gran cantidad de espacio. De manera contraria, **DFS** termina siendo más **rápido** y se consume menos espacio, pero la solución resulta completamente **subóptima**, haciendo una gran cantidad de movimientos innecesarios.

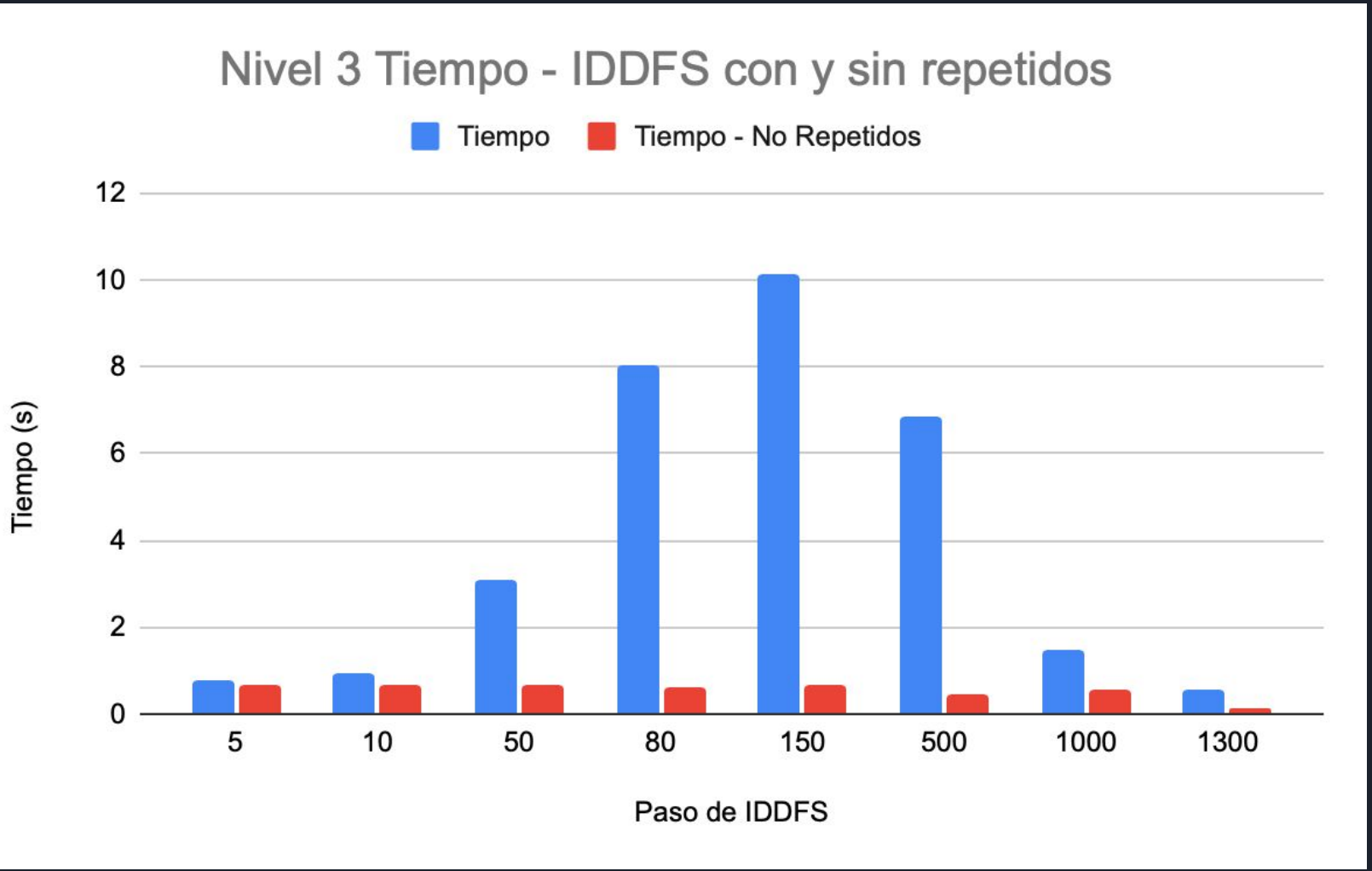
Por último, se puede ver como el comportamiento de IDDFS se asemeja más a BFS cuando el paso es chico y a DFS cuando el paso es mayor.

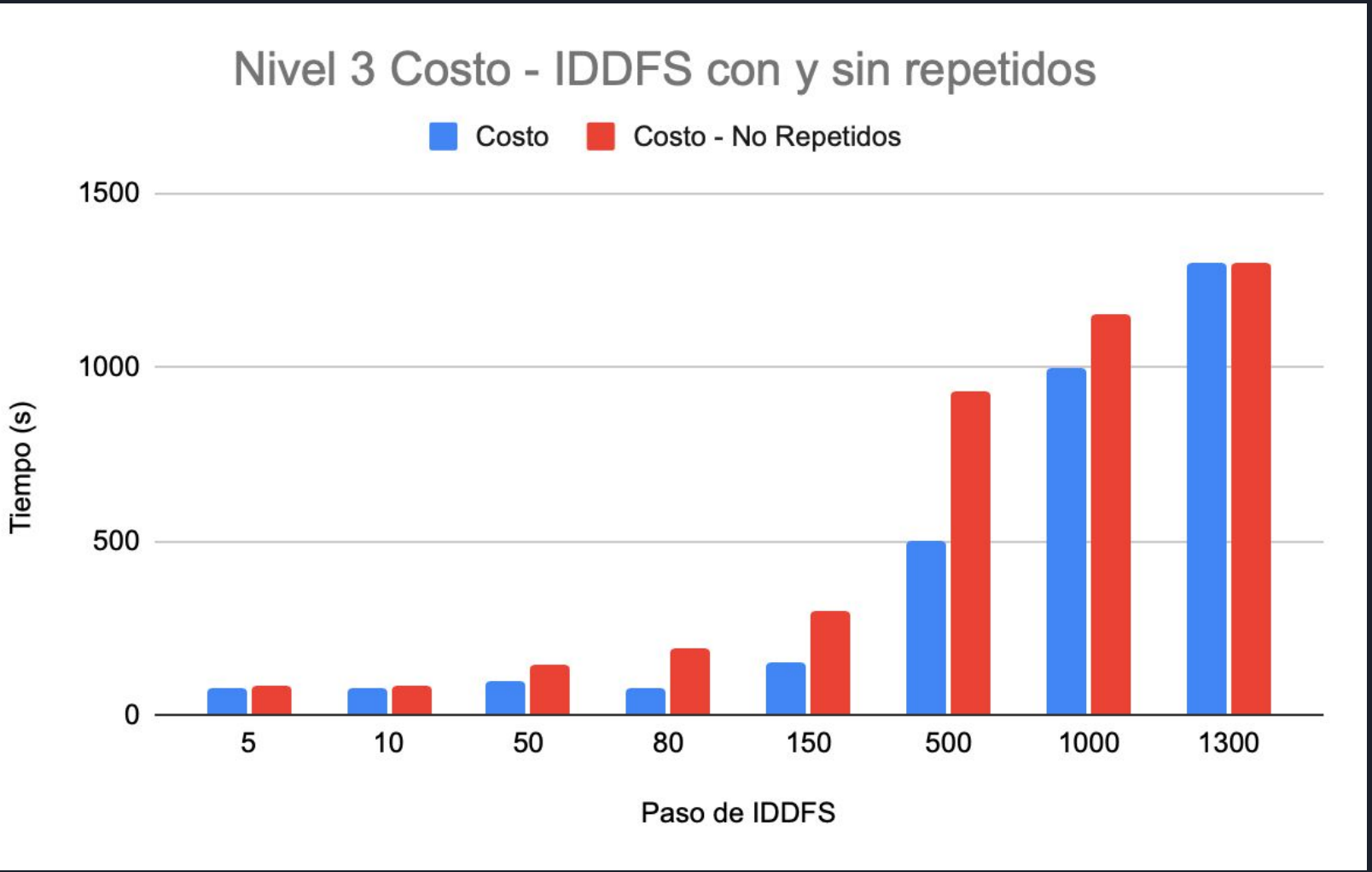

Nivel 3 Tiempo y Costo - IDDFS
Tiempo
Costo
12
10
8
6
4
2
0
1500
1000
500
0
Tiempo (s)
Costo
5
10
50
80
150
500
1000
1300
Paso de IDDFS

IDDFS en función del paso

Para la implementación de IDDFS no es posible implementar la misma limpieza de estados que en BFS y DFS, a la hora de comparar un nodo con el anterior nos interesa saber si tienen el mismo estado y cual tiene menor costo aculado. Esta última consideración nos permite asegurar que se encontrara una solución con la tolerancia dada. El problema de contemplar este escenario es que, para un estado dado, este puede aparecer tantas veces como sea el valor de la tolerancia. Esto implica que, mientras mayor sea el paso elegido, más se ramificará el árbol.

Cuando el paso elegido es muy grande, en este caso 5 veces la solución óptima, el comportamiento se asemeja más a DFS.


Nivel 3 Tiempo - IDDFS con y sin repetidos
Tiempo
Tiempo - No Repetidos
12
10
8
6
4
2
0
Tiempo (s)
5
10
50
80
150
500
1000
1300
Paso de IDDFS


Nivel 3 Costo - IDDFS con y sin repetidos
Costo
Costo - No Repetidos
1500
1000
500
0
Tiempo (s)
5
10
50
80
150
500
1000
1300
Paso de IDDFS

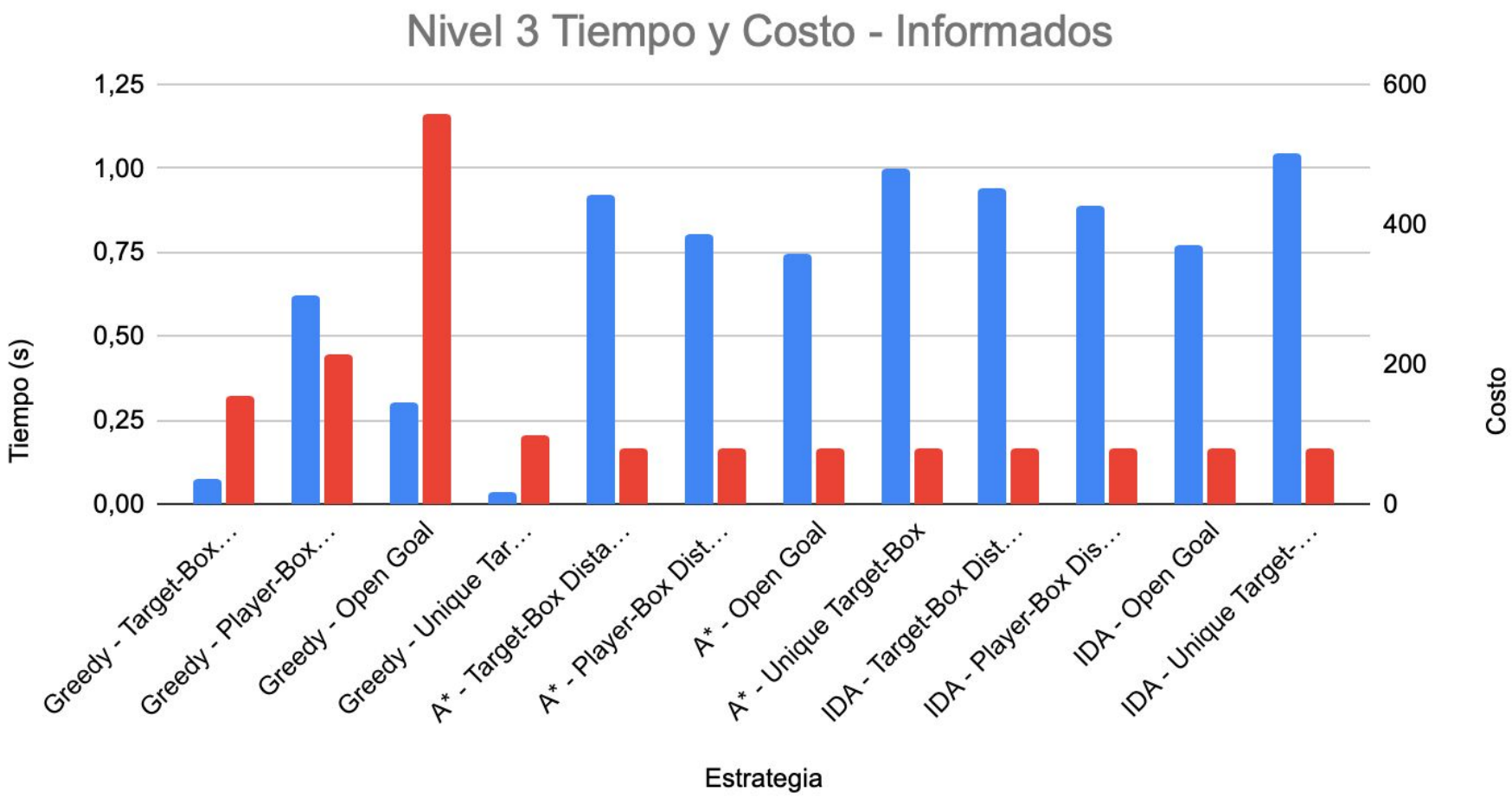
IDDFS con y sin consideración de profundidad

En este caso se realizó un análisis comparativo entre dos implementaciones de IDDFS. Por un lado se realizó el filtrado de nodos teniendo en cuenta el costo acumulado como ya fue mencionado y, en el otro, se filtraron los nodos únicamente por su estado, similar a BFS y DFS.

Se pueden observar dos tendencias claras.

Por un lado, la performance es mucho más estable a lo largo de todos los pasos posibles, equiparable con BFS y DFS.

Por otro lado, la respuesta deja de estar en el rango esperado, sabiendo que la óptima es 79, todos los pasos que se elijan sobre este valor deberían encontrar una solución. De todas formas, se observan casos donde apareció luego del paso, esto nos lleva a pensar que, utilizando esta validación más agresiva, la solución tiene tolerancia de 2 * paso.


Nivel 3 Tiempo y Costo - Informados
1,25
1,00
0,75
0,50
0,25
0,00
Tiempo (s)
600
400
200
0
Costo
Greedy - Target-Box...
Greedy - Player-Box...
Greedy - Open Goal
Greedy - Unique Tar...
A* - Target-Box Dista...
A* - Player-Box Dist...
A* - Open Goal
A* - Unique Target-Box
IDA - Target-Box Dist...
IDA - Player-Box Dis...
IDA - Open Goal
IDA - Unique Target-...
Estrategia

Conclusiones Greedy

Come se están filtrando estados repetidos esta es una búsqueda completa.

Como se puede comprobar en los gráficos la heurística 4 muestra los mejores resultados tanto temporales como espaciales. Esto se debe a que resulta la heurística que mejor modela el objetivo del juego. Aquí puede comprobarse como el desempeño del algoritmo está ligado fuertemente con la heurística usada. Seguida de esta esta la número 3 que si bien es una versión inferior de la 4, se comportan de manera similar. Cual de estas funcione mejor depende principalmente de la distribución inicial de las cajas, dado que si cada caja es cercana a un objetivo distinto, el cálculo del valor es más rápido. La primer heurística termina encontrando una solución relativamente rápido, pero de manera muy ingenua usando muchos movimientos. Por último la segunda resulta bastante más lenta en niveles donde se puede mover mucho la caja y no dejarla necesariamente en ningun objetivo.

En cualquier caso, dado que este algoritmo no busca la solución óptima, en muchos casos termina siendo más rápido que los otros dos métodos informados.

Conclusiones A*

Dado que todas nuestras heurísticas son admisibles, la cantidad de nodos a expandir son finitos y todos tienen un costo uniforme mayor que cero, este algoritmo siempre encuentra solución y es la óptima.

En cuanto a las heurísticas, salvo en el nivel tres, se comportó de manera similar a Greedy. Se analiza que en el caso de este algoritmo, resulta menos susceptible a una mala heurística para un nivel, como se ve en el caso del nivel 2 para el caso de Greedy. Esto se debe a que a la hora de irse por un mal camino, este algoritmo ya ha considerado que la profundidad es mucha y ha desestimado la heurística.

El problema principal de este algoritmo es que para garantizar el camino óptimo, puede llegar a tardar bastante mas.

Conclusiones IDA

Bajo las mismas condiciones que A*, este algoritmo encontrará la solución y será óptima.

Se pudo analizar como en la mayoría de los casos se comportó de modo similar a A* resultando mejor en algunos niveles y peor en otros. Esto se debe a que termina dependiendo mucho del límite el tiempo que le tardará encontrar la solución.

En cualquier caso, lo que hace superador a este algoritmo es que, al estar cambiando de profundidad constantemente se utiliza una cantidad de memoria considerablemente menor, lo que hace que este algoritmo sea implementable en mayor cantidad de situaciones