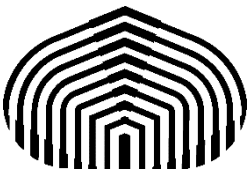


Universidad Simón Bolívar

CI-5437 Inteligencia Artificial

Prof. Carlos Infante

INFORME DEL PROYECTO 1

Simon Puyosa 18-10717

Ana Shek 19-10096

Juan Cuevas 19-10056

Caracas, Febrero del 2024.

Tabla de Contenido

INTRODUCCIÓN

La Inteligencia Artificial (IA) ha emergido como una disciplina transformadora que redefine la manera en que abordamos problemas complejos y tomamos decisiones en diversos campos. Su capacidad para imitar procesos cognitivos humanos y aprender patrones complejos la convierte en una herramienta invaluable en la resolución de problemas.

Dentro del ámbito de la Inteligencia Artificial, se destacan ciertos algoritmos de búsqueda informada que son clave para resolver diversos problemas. Entre estos, A* e IDA* sobresalen por su habilidad para emplear funciones heurísticas que guían la búsqueda hacia soluciones óptimas. La eficacia de estos algoritmos depende en gran medida de la calidad de la heurística utilizada, ya que una heurística bien desarrollada puede mejorar significativamente el rendimiento del algoritmo.

Una de las heurísticas más destacadas es el PDB (Pattern Database) que aborda problemas reduciendo su complejidad al proyectarlos en un espacio de estados de menor dimensión. Se resuelve cada subproblema de forma independiente. Para que la heurística PDB sea válida, debe cumplir con las características de admisibilidad y consistencia en la suma de las soluciones de cada subproblema. Si se cumplen estas condiciones, entonces la heurística PDB se considera aditiva; de lo contrario, se clasifica como no-aditiva, y se puede optar por la solución que maximice los resultados.

A continuación, el presente trabajo se centra en los algoritmos de búsquedas de profundidad, el análisis del modelo de espacio de estados y los diversos algoritmos de búsqueda heurística, como A* e IDA*, aplicados a una serie de problemas concretos, entre los que se incluyen el N-puzzle, Top-Spin, Torres de Hanoi y Cubo de Rubik. En este estudio se explorará la eficacia y la aplicabilidad de estos algoritmos en la resolución de problemas complejos, ofreciendo así una comprensión más profunda de su funcionamiento y sus posibles aplicaciones prácticas.

Los experimentos fueron realizados en las siguientes plataformas:

- Un laptop con procesador AMD Ryzen 3 3200U, disco SSD, 20GB de memoria RAM y WSL 2 Ubuntu.
- Un laptop con procesador AMD Ryzen 5 5500U, disco SSD, 8GB de memoria RAM y WSL 2 Ubuntu.
- Una computadora con procesador Intel Core i7-8700, disco SSD, 16GB de memoria RAM y Linux Mint 21.1 x86_64

Y las implementaciones de los algoritmos que se requieren del informe están en el siguiente enlace: [proyecto-1-ci5437/global at main · anna-ayn/proyecto-1-ci5437 (github.com)](), donde:

- ucs1 es UCS sin pruning
- ucs2 es UCS con pruning
- bfs1 es BFS sin pruning
- bfs2 es BFS con pruning
- iddfs1 es IDDFS sin pruning
- iddfs2 es IDDFS con pruning
- astar es A*
- idastar es IDA*

Representaciones de los problemas en PSVN

1. **N-puzzles**
   - ❖ 15-puzzle

El 15-puzzle es un juego que consiste en un tablero cuadrado de 4x4 casillas, con 15 fichas numeradas del 1 al 15 más una casilla vacía. El objetivo del juego es ordenar las fichas de forma ascendente moviendo las fichas a las casillas adyacentes a la casilla vacía.

La representación en PSVN para este problema define un dominio llamado “tile” con 16 elementos enumerados de 1 a 15 más el espacio en blanco “b”. Se representa el problema como un vector de 16 posiciones, una para cada casilla del tablero. Además, cada movida posible tiene un costo 1.

- ❖ 24-puzzle

El 24-puzzle es otro juego similar a 15-puzzle solo que éste es un tablero cuadrado de 5x5 casillas, con 24 fichas numeradas de 1 al 24 más una casilla vacía. Así mismo, el objetivo es tener todas las fichas ordenadas de forma ascendente.

La representación en PSVN también es similar al de 15-puzzle, tiene mismo dominio pero con 24 elementos enumerados de 1 al 24 más el espacio en blanco “b”. Se representa el problema como un vector de 24 posiciones, una para cada casilla del tablero y cada movida en el juego tiene un costo 1 también.

2. Cubo de Rubik

Para este problema, hemos usado el PSVN que ya trae consigo en el zip de psvn-for-ci5437. La representación en PSVN está modelado usando un encoding de "stickers". Se define un dominio "colour" con 6 valores para los colores del cubo (RED, GREEN, BLUE, YELLOW, ORANGE, WHITE) y se usan 48 variables para representar el estado, 8 por cada cara del cubo (U, F, R, B, L, D).

El nombre de cada variable indica en qué cara y posición está. Por ejemplo "FUR" es la esquina superior derecha de la cara F. Además, se definen los movimientos posibles (giros de cada cara) y cómo transfieren los stickers de una posición a otra. Y el estado objetivo del cubo de rubik es tener los stickers de cada color agrupados en una cara.

3. Top-Spin (12-4, 14-4, 17-4)

El Top Spin Puzzle es un rompecabezas que consiste en una plataforma circular con M fichas numeradas del 0 al M - 1, en este caso M es 12, 14 o 17. Y tiene una pequeña rueda giratoria que puede rotar en sentido antihorario las N fichas en posiciones consecutivas (en este caso N = 4). El objetivo es tener los números ordenados.

Las representaciones en PSVN para M = 12, 14 y 17 con N = 4 son generados con el archivo *genTopSpinBasic.cpp* del zip, el cual define M elementos, uno para ficha y 4 variables (X1, X2, X3, X4). Y se define las operaciones con costo 1 para rotar 4 fichas en sentido antihorario y se tiene 12 estados objetivos el cual cada una representa las diferentes maneras de tener las fichas ordenadas.

4. Torre de Hanoi (4 astas con 12/14/18 discos)

El juego de la torre de Hanoi con 4 astas y N discos de tamaños diferentes que se apilan en uno de las astas en orden decreciente de tamaño consiste en lograr transferir toda la pila de discos a otra asta siempre y cuando se mueva un disco a la vez y nunca colocar un disco sobre otro más pequeño.

Las representaciones en PSVN con D = 12, 14 y 18 y 4 astas son generados con el archivo *genHanoi.cpp* del zip. Cada PSVN modela 48 variables de estado binarias (12 por cada asta), indicando presencia/ausencia de cada disco. Cada operación para mover un disco de una asta a otra cuesta 1 y es representado por un estado inicial donde el asta en que está es un 1 y el asta objetivo (el asta donde se quiere mover el disco) es un *0, y el estado luego de aplicar la operación consiste en convertir el 1 por *0 y el *0 por 1 del estado inicial.

El estado goal es tener todos los discos en la asta 1.

Árboles de Búsqueda

Se utilizaron los algoritmos Uniform Cost Search (UCS), Breadth First Search (BFS) e Iterative Deepening Depth-First-Search (IDDFS) para el estudio de los árboles de búsqueda de los problemas y su factor de ramificación sin eliminación de duplicados y con eliminación parcial de duplicados (poda de ancestros).

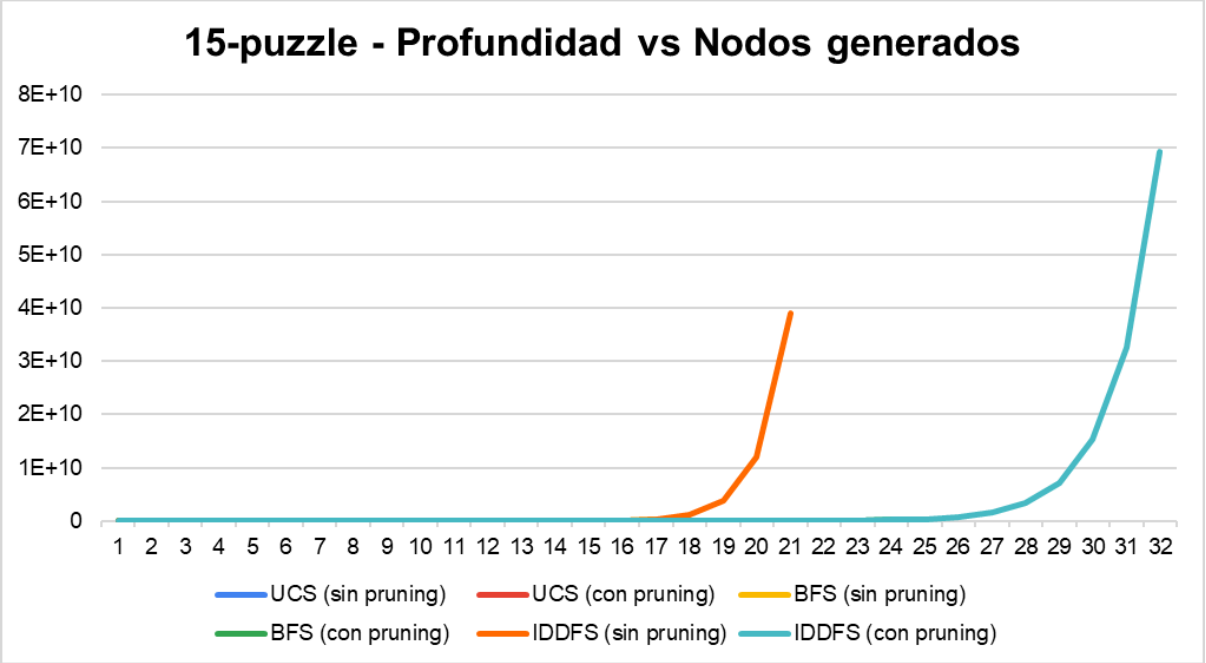
En el siguiente enlace se encuentran los resultados para cada puzzle usando los 3 algoritmos mencionados: Algoritmos de Busqueda UCS, BFS e IDDFS - Hojas de cálculo de Google

Para la poda de ancestros, se utilizó los siguientes flags en el makefile:

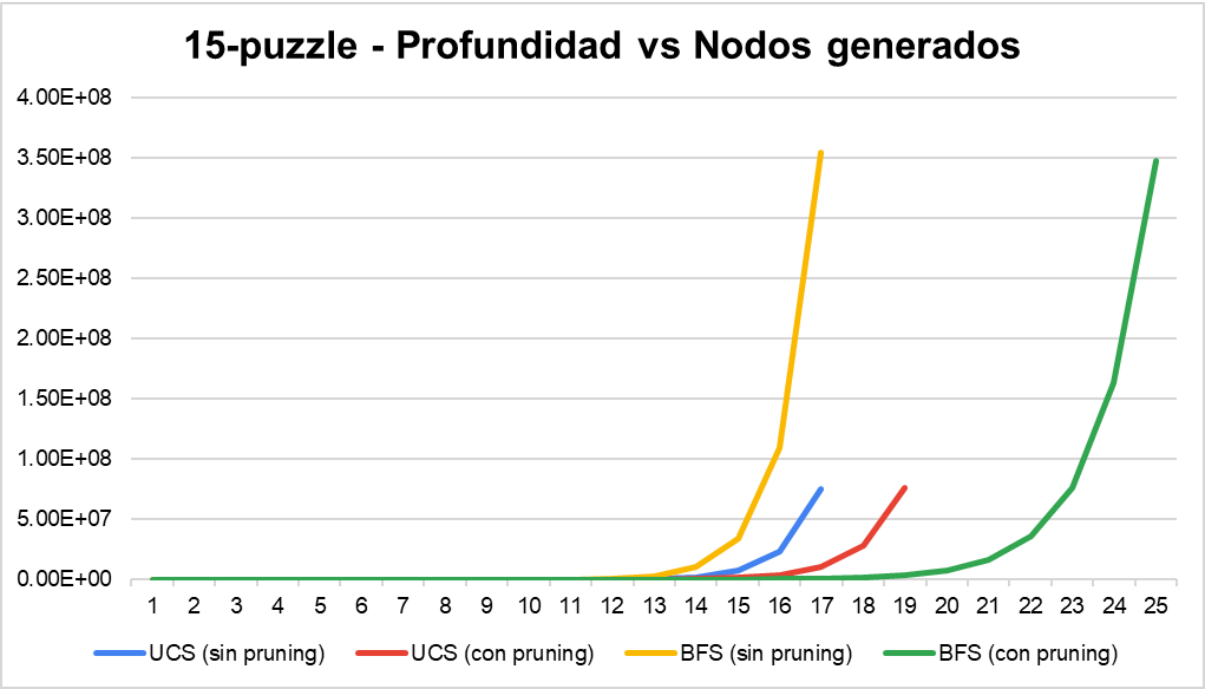
```
OPT = -Wall -Ofast -Wno-unused-function -Wno-unused-variable -march=native
PSVNOPT = --state_map --no_abstraction --backwards_moves
--fwd_history_len=1 --bwd_history_len=1
```

1. **N-puzzles**
   - 15-puzzle

     UCS vs BFS vs IDDFS

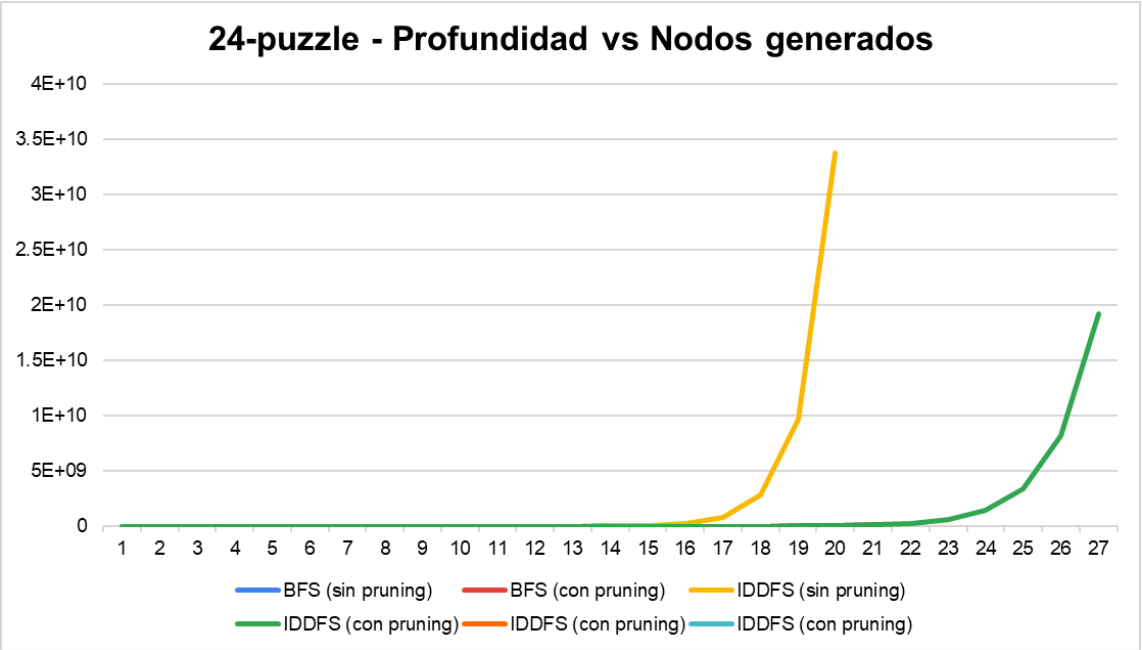


Como en el anterior gráfico no se pudo visualizar bien las líneas de UCS y BFS ya que el IDDFS crece más rápido en comparación con los otros algoritmos de búsqueda de profundidad, tenemos este siguiente gráfico UCS vs BFS:

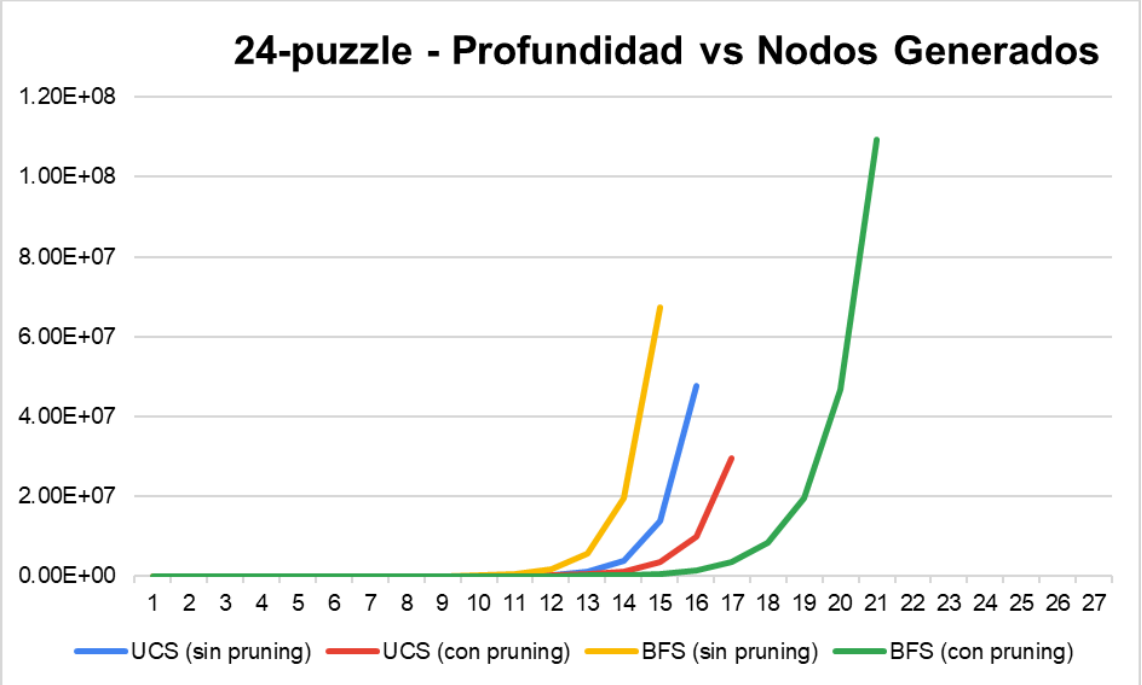


- 24-puzzle

  UCS vs BFS vs IDDFS

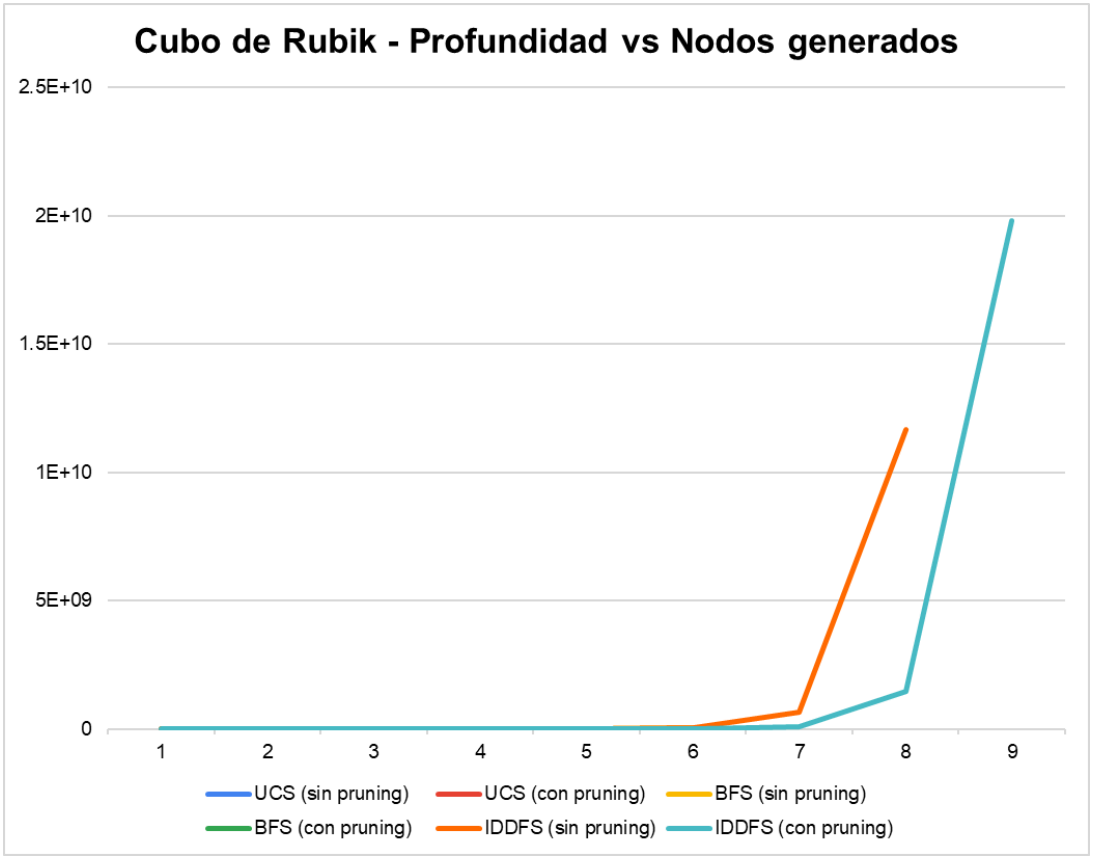


  UCS vs BFS

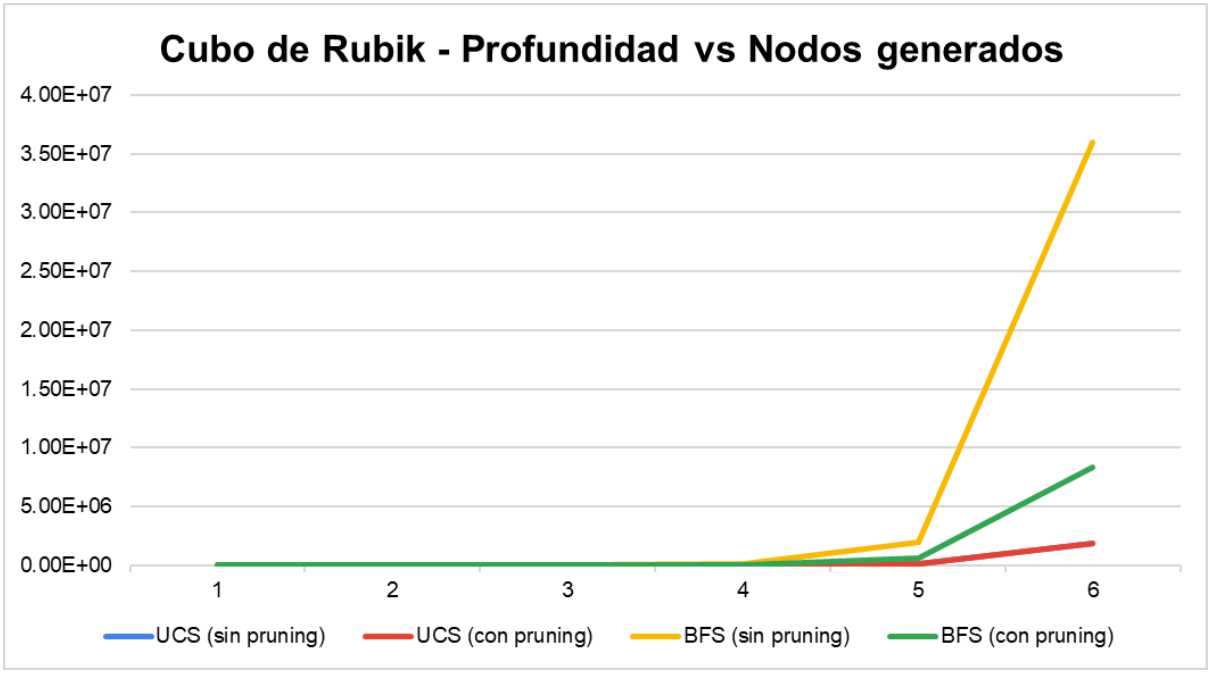


2. Cubo de Rubik

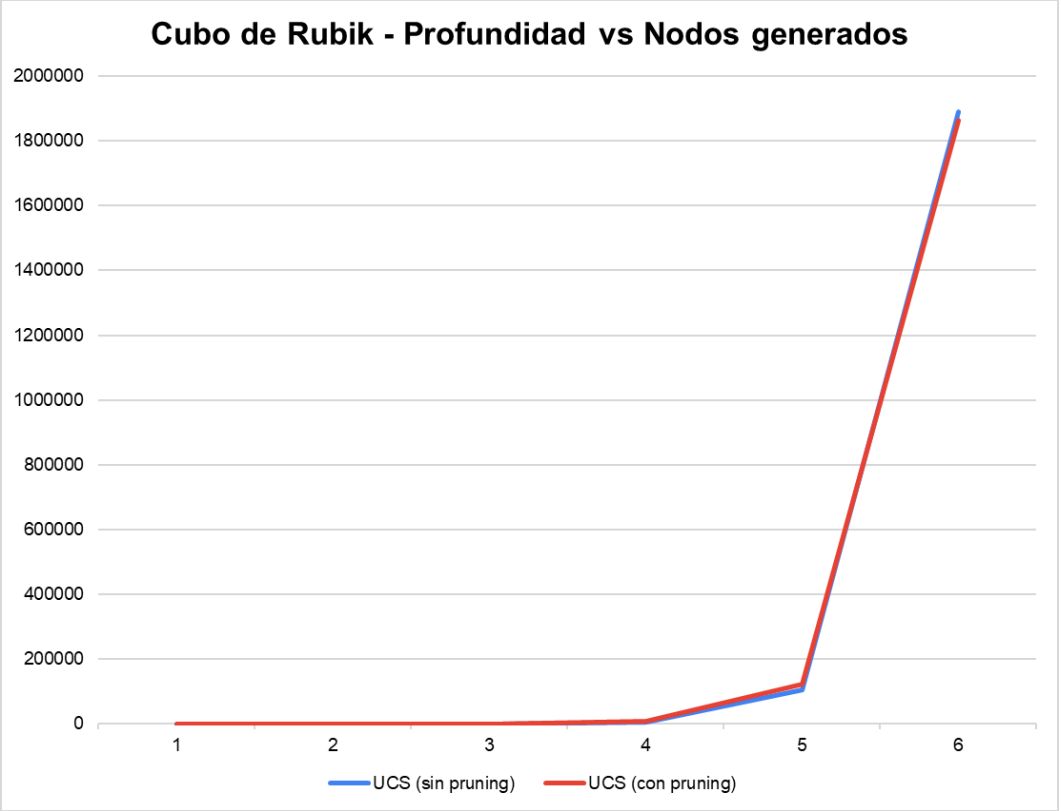
UCS vs BFS vs IDDFS



UCS vs BFS



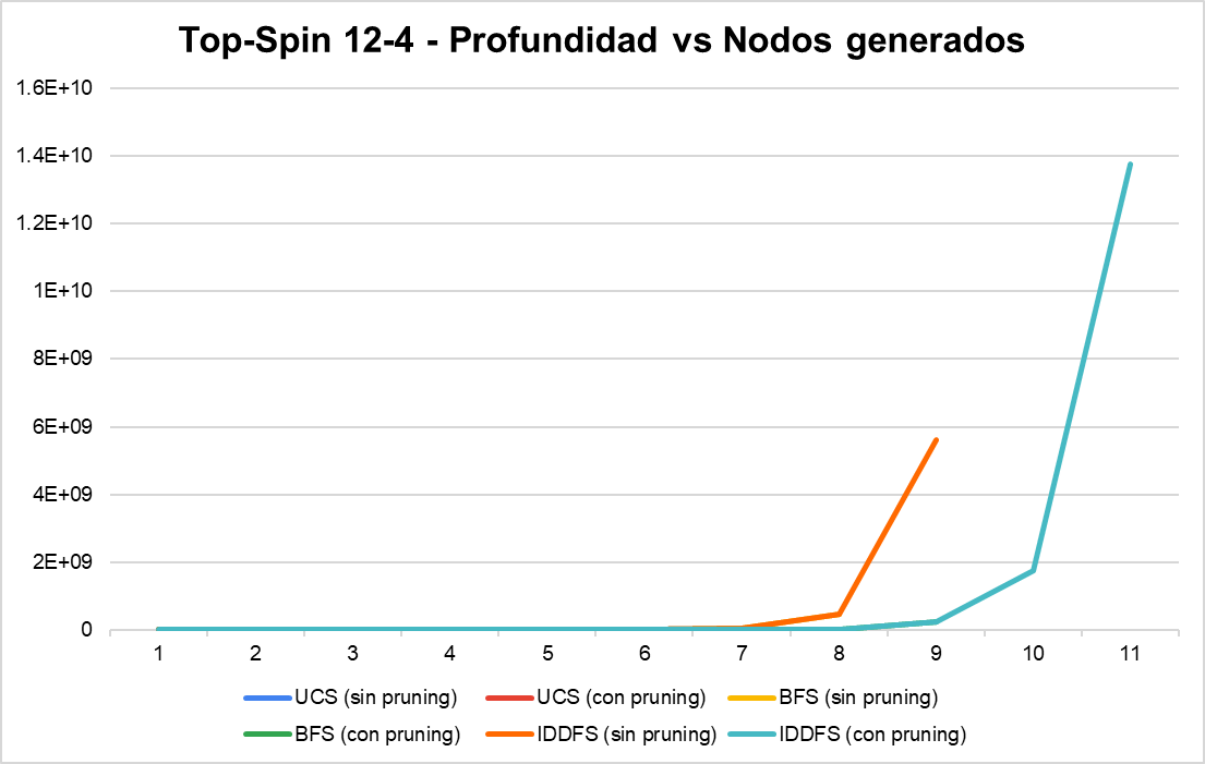
UCS sin pruning vs UCS con pruning



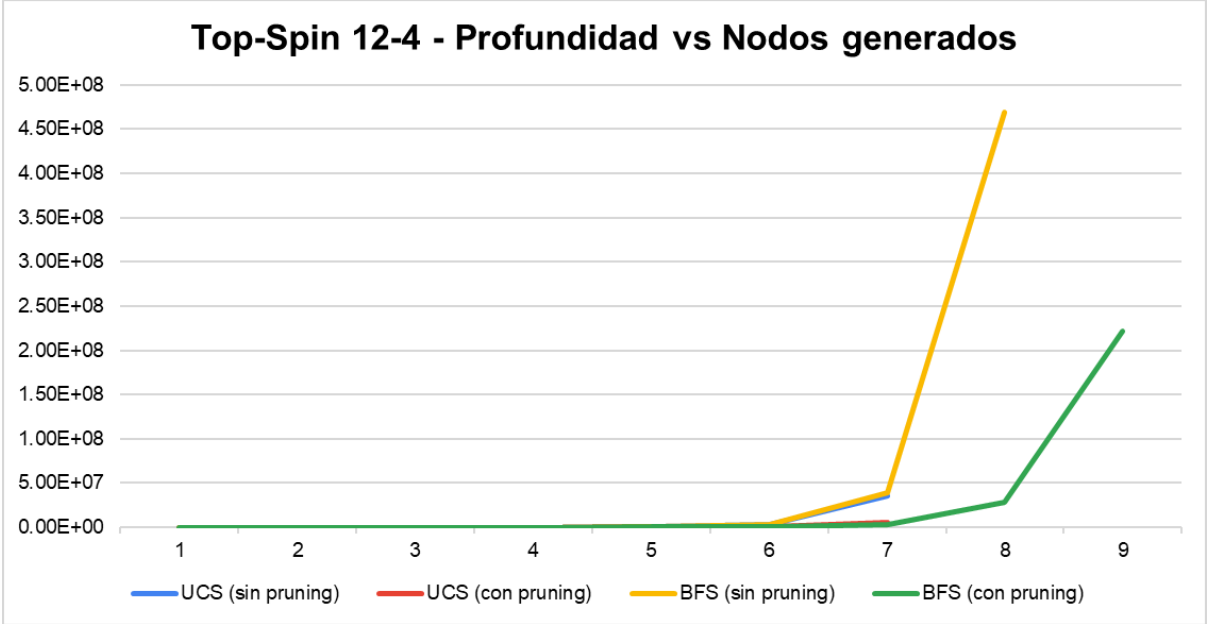
3. Top-Spin

- 12-4

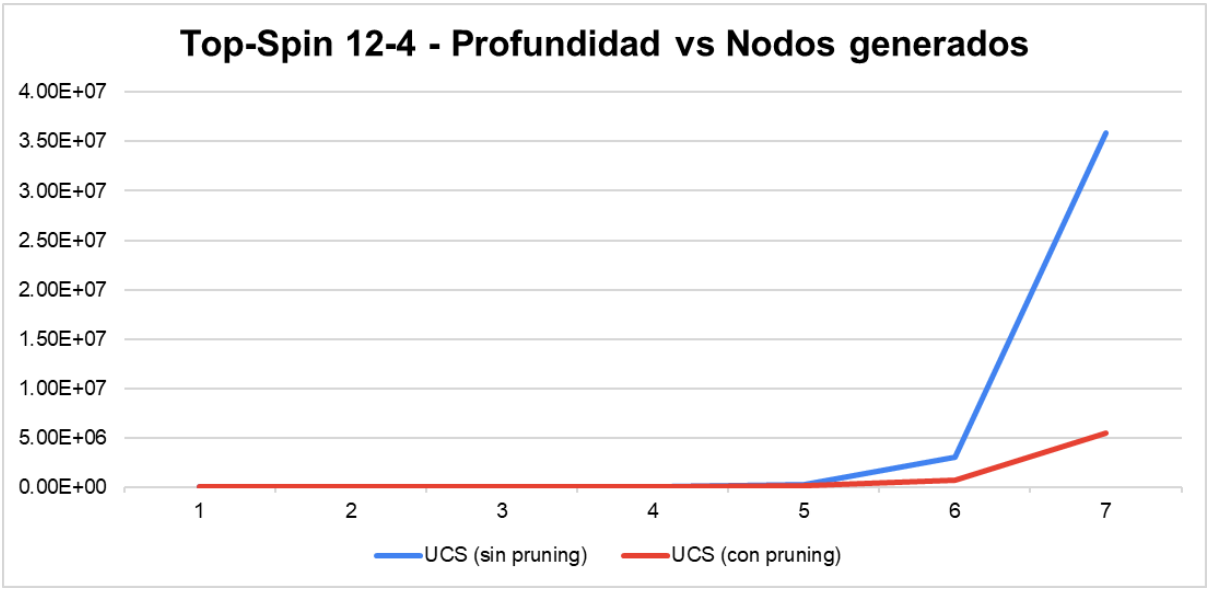
UCS vs BFS vs IDDFS



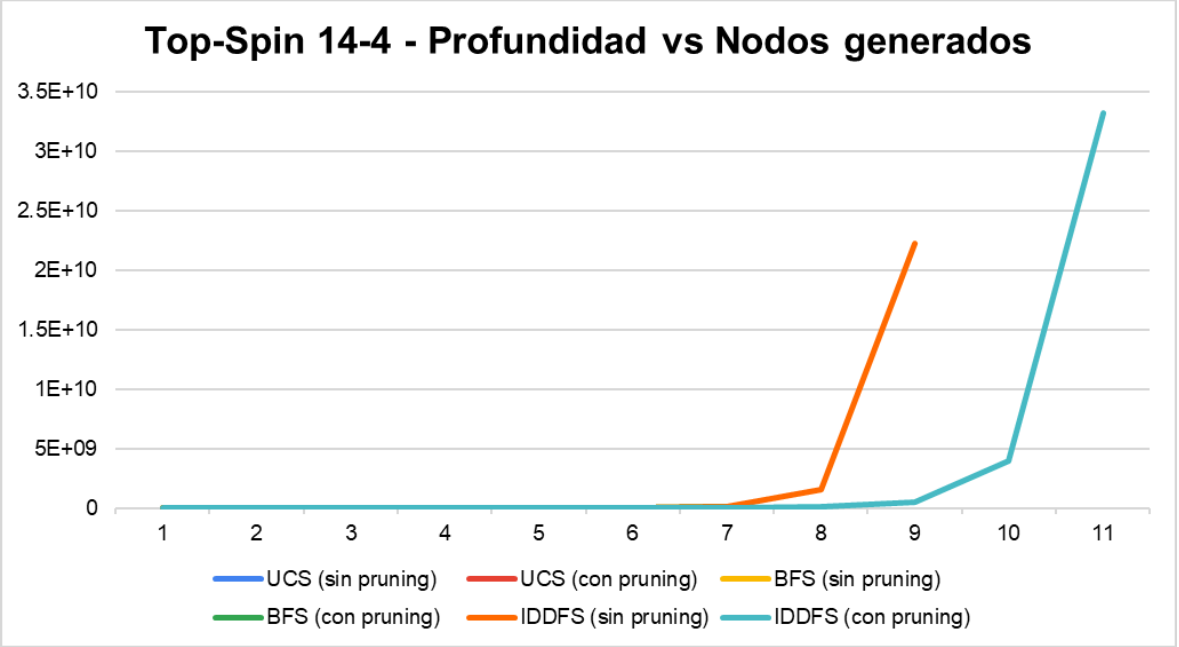
UCS vs BFS



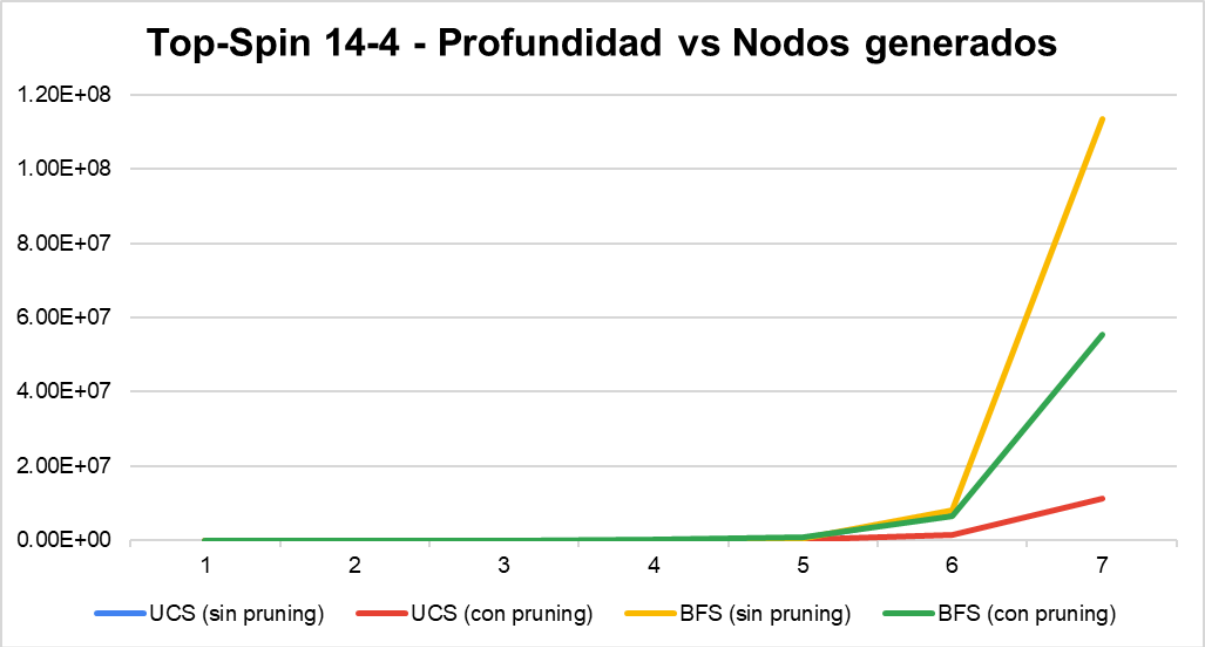
UCS sin pruning vs UCS con pruning



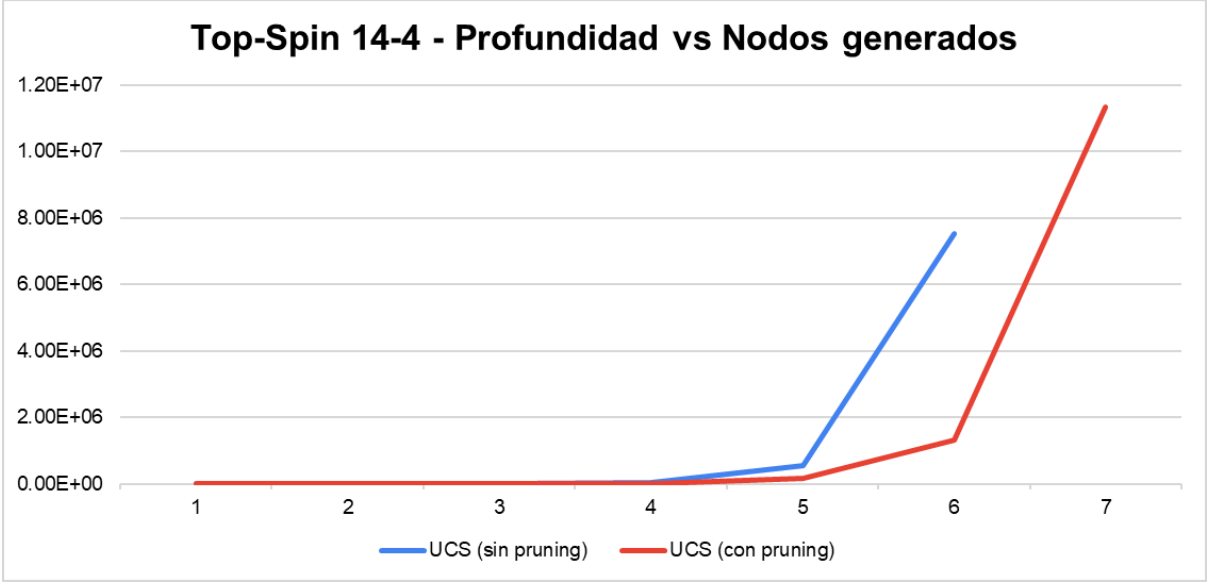
- 14-4



UCS vs BFS

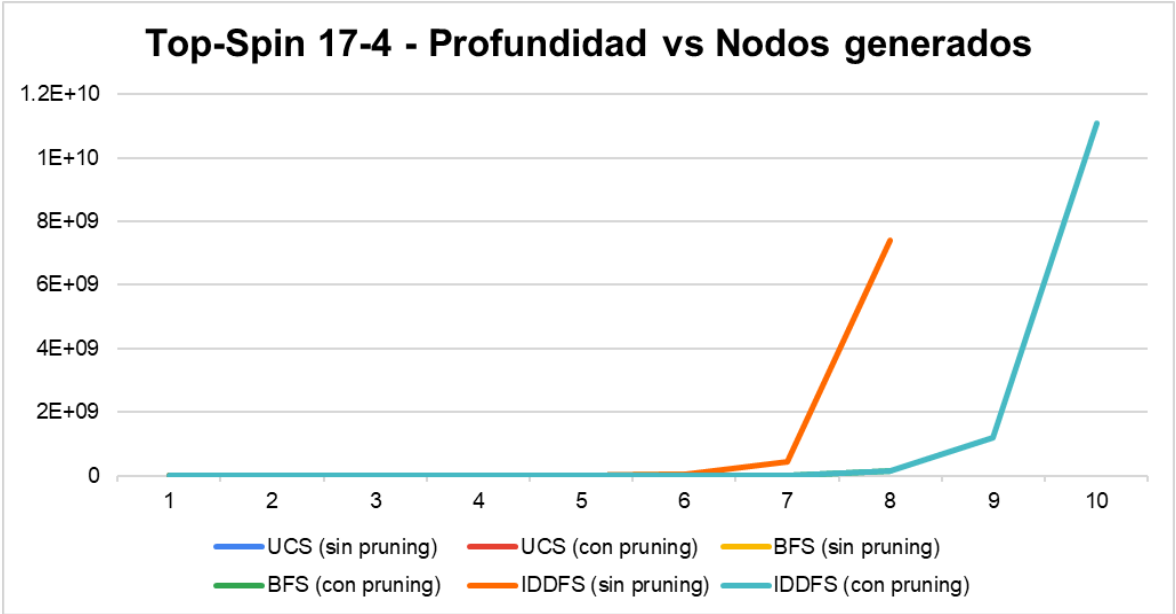


UCS sin pruning vs UCS con pruning

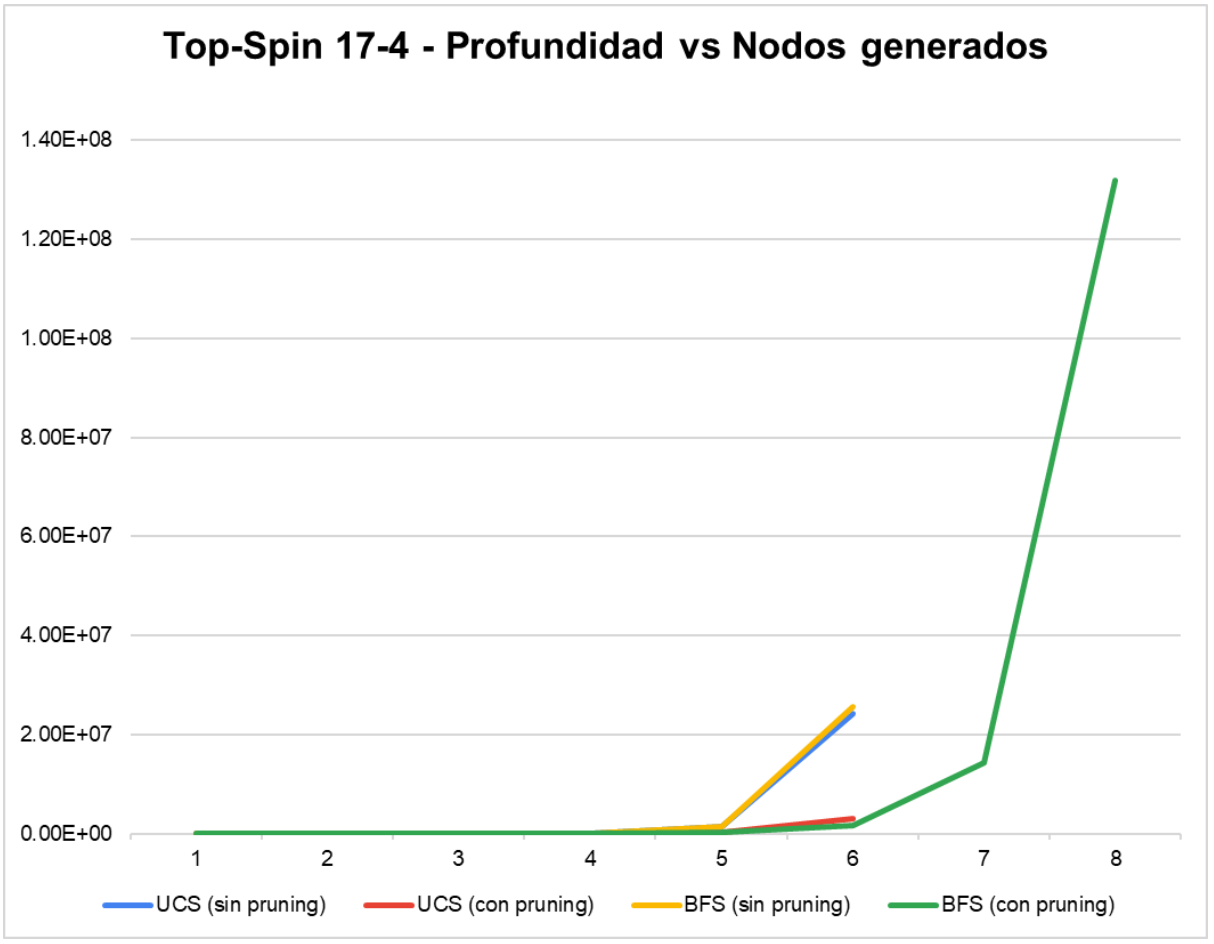


- 17-4

  UCS vs BFS vs IDDFS



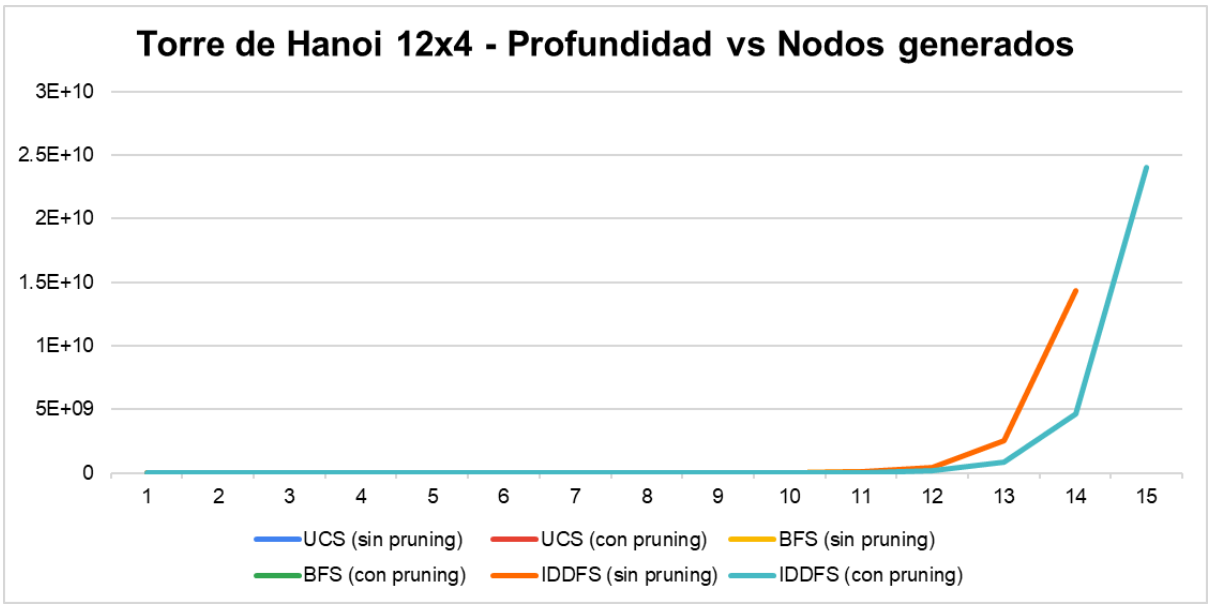
UCS vs BFS



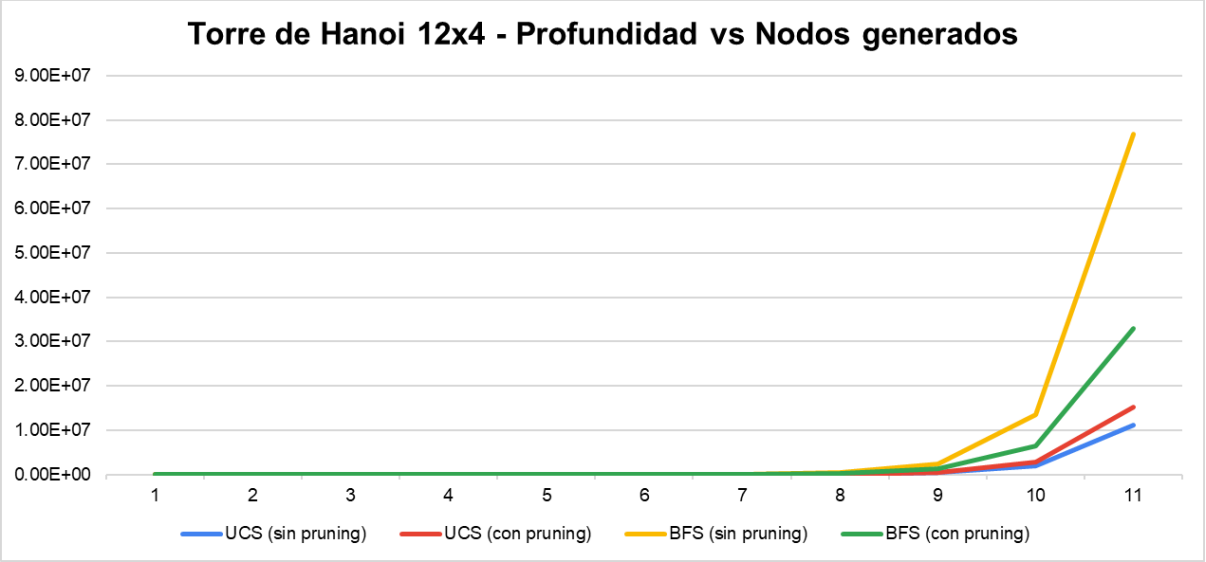
4. Torre de Hanoi

- 12 discos y 4 astas

UCS vs BFS vs IDDFS

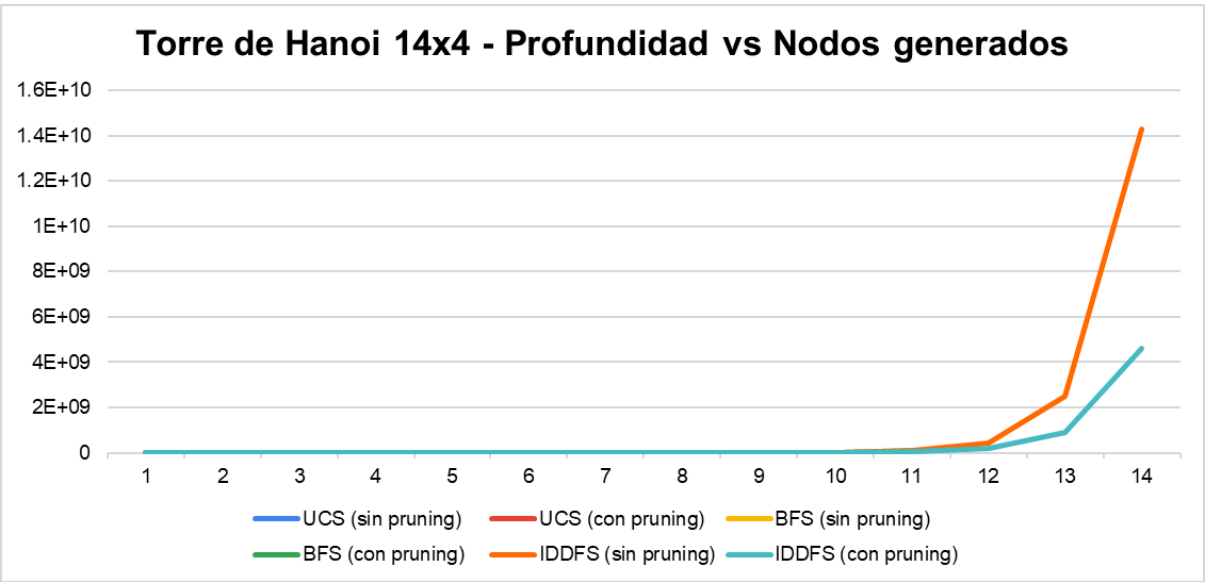


UCS vs BFS

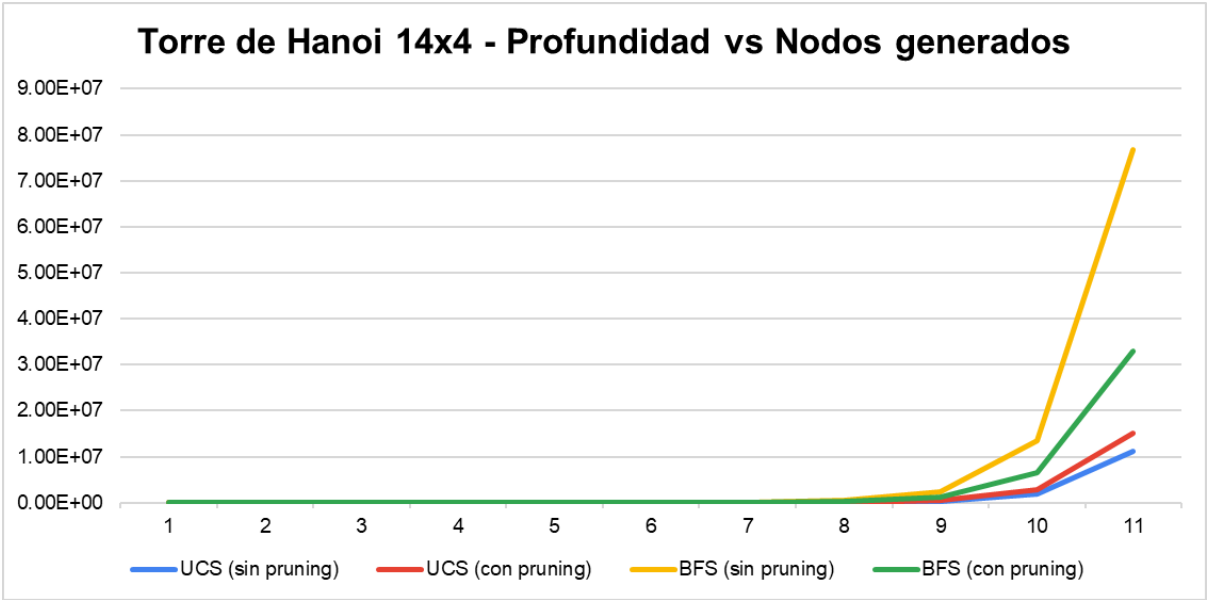


- 14 discos y 4 astas

  UCS vs BFS vs IDDFS

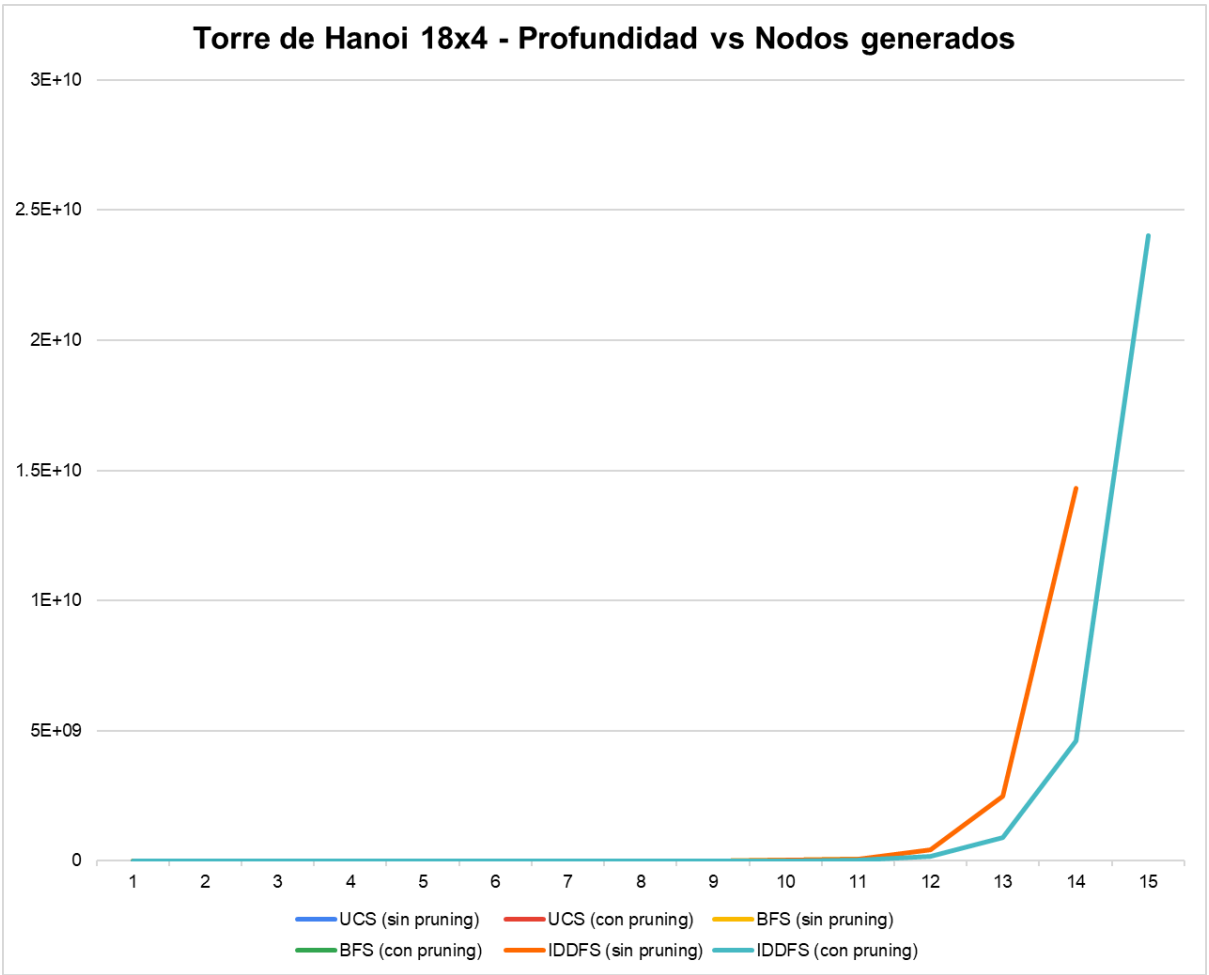


UCS vs BFS

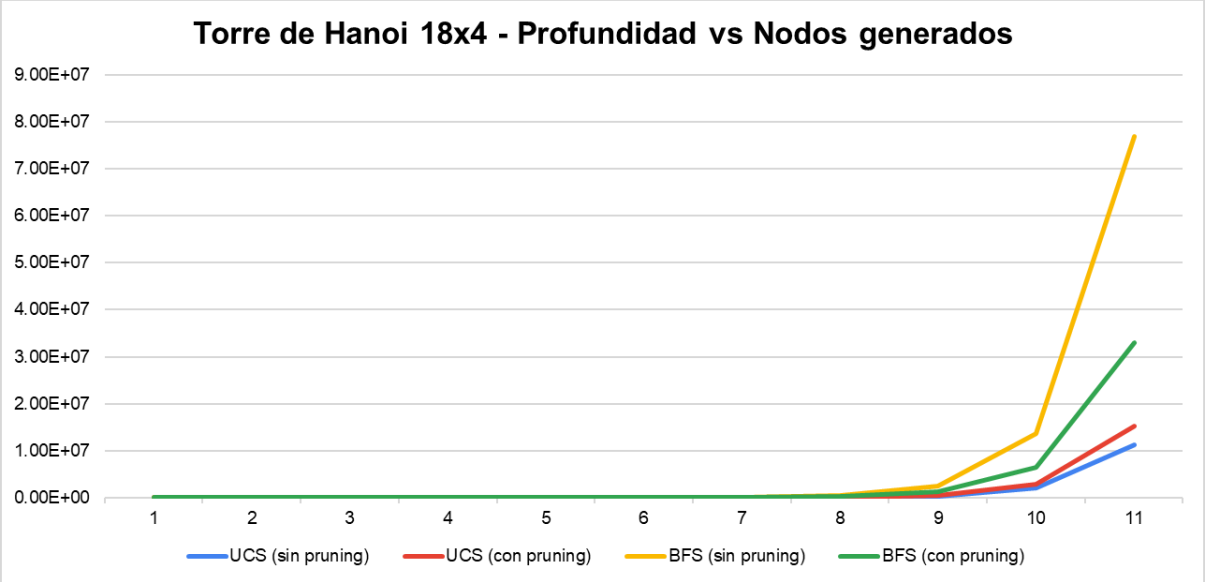


- 18 discos y 4 astas

  UCS vs BFS vs IDDFS



UCS vs BFS



Análisis de resultados

El análisis de resultados muestra que el uso de la eliminación de duplicados para cada problema reduce el factor de ramificación en cada nivel de profundidad, en comparación con los algoritmos que no utilizan esta técnica. Además, aunque UCS con poda genera más nodos en los primeros niveles, este número se estabiliza a medida que la búsqueda se profundiza y encuentra menos estados que UCS sin poda.

Así mismo, se observa una marcada reducción en el número de estados generados en cada nivel de profundidad al emplear la eliminación de duplicados para BFS e IDDFS y también para UCS luego de los primeros niveles. Esta reducción es especialmente notable en el caso del 15-puzzle, donde, por ejemplo, el número de estados generados a una profundidad de 15 se reduce de 3234207 a 39719 con la técnica de pruning en BFS. De igual manera se tiene para todos los casos que la técnica de pruning es considerablemente mejor a la sin pruning con distintos niveles de mejoras, esto debido a la ausencia de estados inválidos que restrinjan los movimientos.

Al examinar el desempeño de diversos algoritmos de búsqueda, se puede notar que IDDFS tiene la capacidad de llegar a profundidades mayores que BFS y UCS. La razón detrás de esto es que el uso de memoria de BFS y UCS es significativamente mayor que el de IDDFS, lo que puede limitar su alcance antes de llegar al tiempo de ejecución deseado. No obstante, en algunos casos, IDDFS puede requerir un poco más de tiempo para alcanzar cierta profundidad en comparación con los otros algoritmos mencionados.

Heurísticas

Se implementaron las heurísticas:

- Distancia Manhattan (15-puzzle)
- Diferente additive PDBs (15-puzzle y 24-puzzle)
- Max de diferentes PDBs (Cubo de Rubik, Top Spin, Torre de Hanoi).

1. N-puzzles

- 15-puzzle: Se utilizaron las siguientes abstracciones, el cual cada color identifica a una abstracción correspondiente:

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

  - ➢ Rojo: test1.txt
  - ➢ Verde: test2.txt
  - ➢ Azul: test3.txt

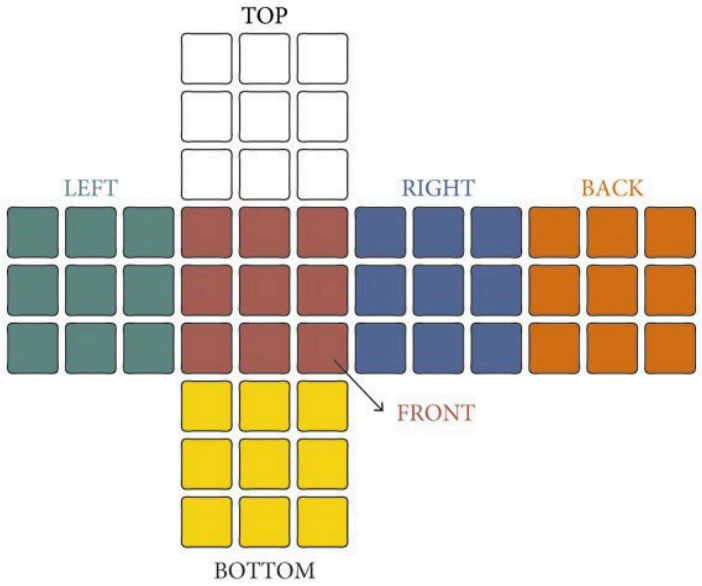
- 24-puzzle: Se utilizó la heurística para diferentes additives PDBs
  Cada color identifica a una abstracción correspondiente:

	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

  - ➢ Rojo: test1.txt
  - ➢ Anaranjado: test2.txt
  - ➢ Verde: test3.txt
  - ➢ Amarillo: test4.txt
  - ➢ Azul: test5.txt
  - ➢ Morado: test6.txt

2. Cubo de Rubik

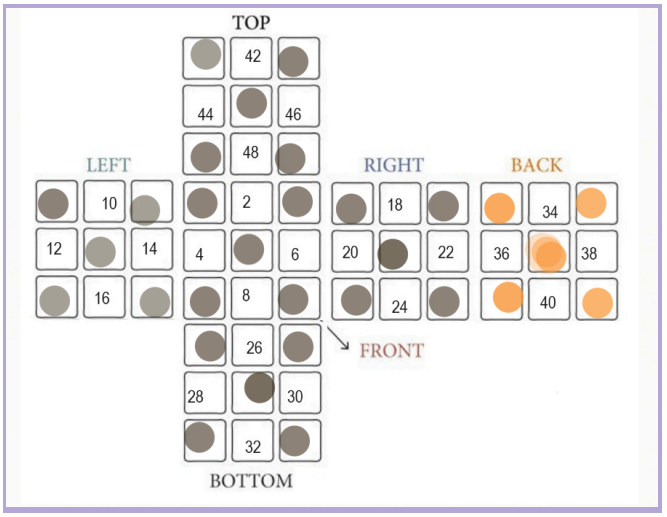
Para realizar las abstracciones, nos guiamos de un dibujo de un cubo de rubik desarmado:



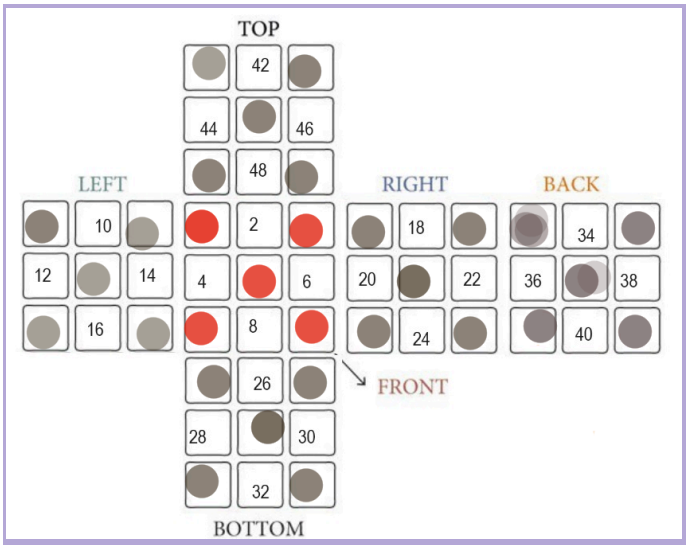
Entonces, para el corner PDB se dividieron en 5 abstracciones (ya que eran demasiados grandes y no corría usando una sola abstracción)

Las casillas coloreadas por un color gris o con otro color son las variables que NO fueron agregadas a la lista de variables a eliminar por proyección. Mientras que las casillas que están numeradas con un número par son las que fueron agregadas a la lista de variables a eliminar por proyección. Esto es para tener una abstracción similar a la foto de abajo:

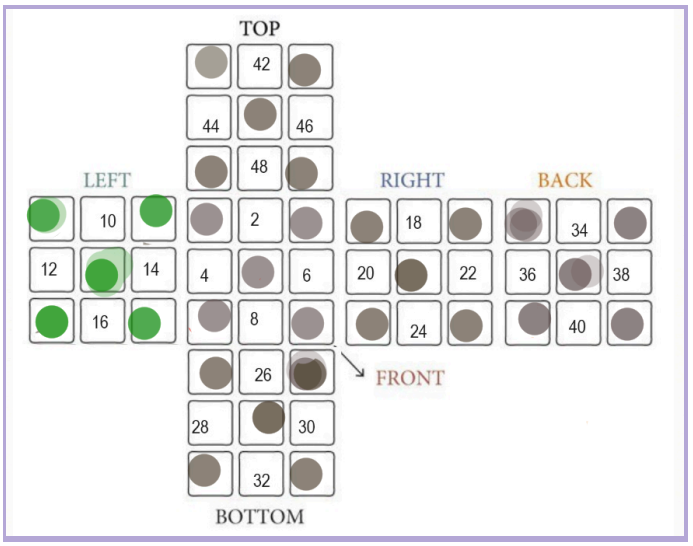
➢ corner_1.txt



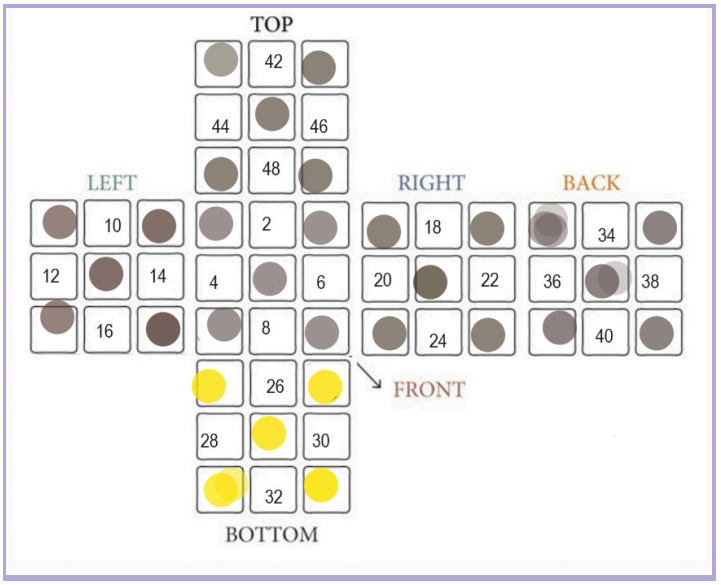
➢ corner_2.txt



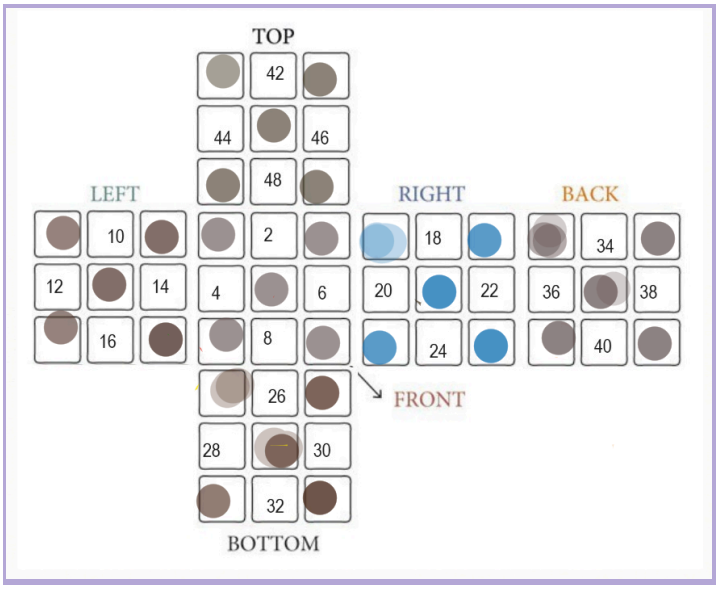
➢ corner_3.txt



➢ corner_4.txt



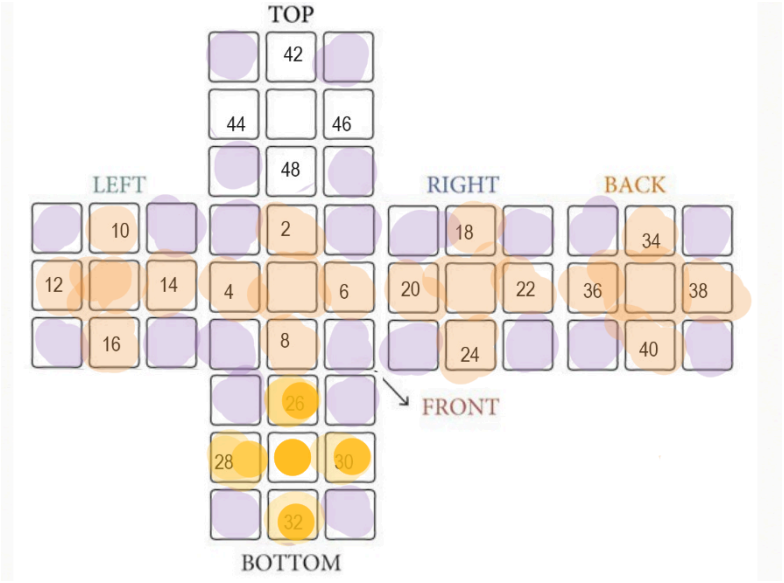
➢ corner_5.txt



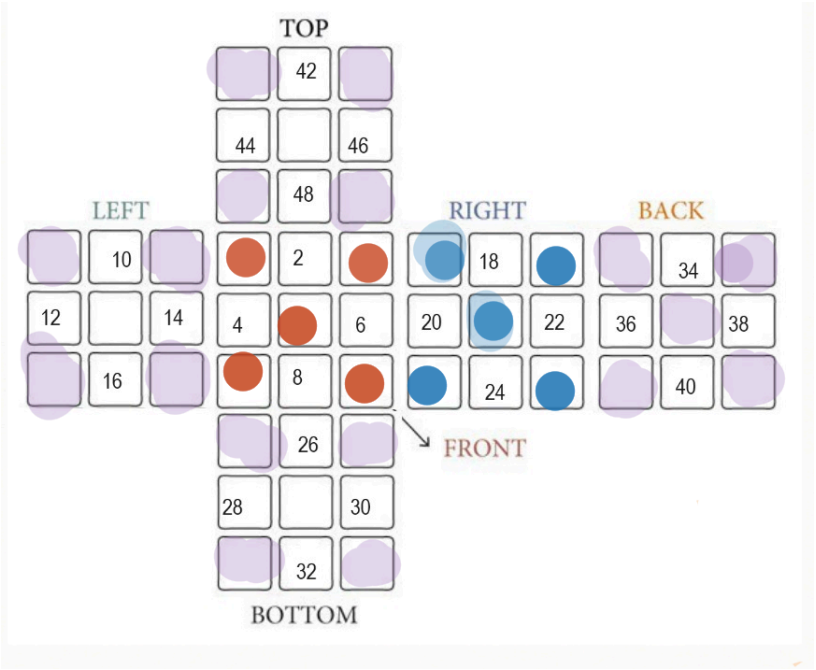
Y para los 2 edges PDBs:

Las casillas con color lila son las variables que fueron agregadas a la lista de variables a eliminar por proyección.

➢ edge_1.txt



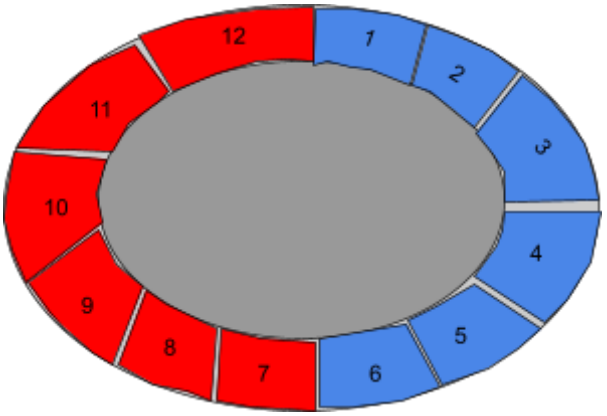
➢ edge_2.txt



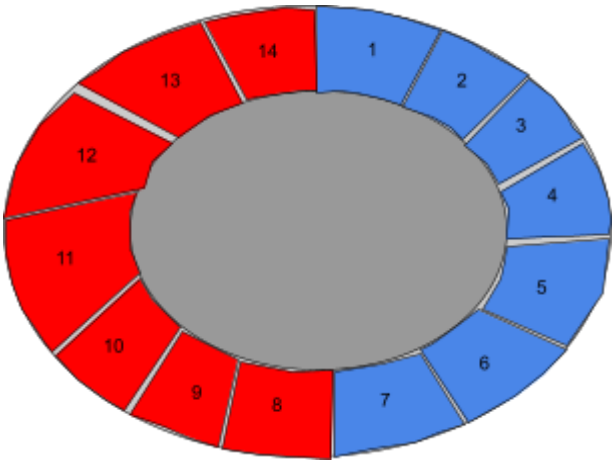
3. Top-Spin

Cada color identifica a una abstracción correspondiente

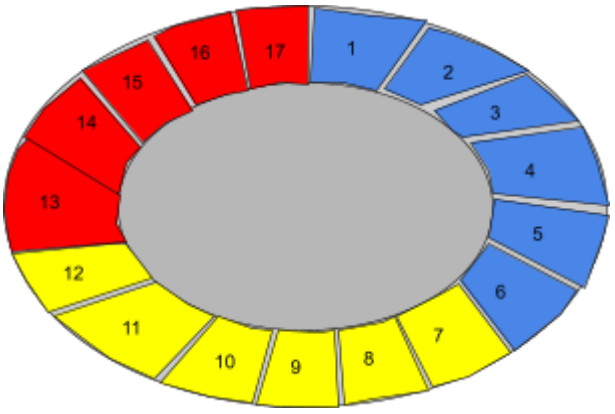
- **Top-Spin 12x4**



- **Top-Spin 14x4**



- **Top-Spin 17x7**



➢ Rojo: group_1.txt

➢ Azul: group_2.txt

➢ Amarillo: group_3.txt

4. Torres de Hanoi

Se mostrará una tabla en donde cada columna es una asta y cada fila es un disco. Se mostrarán en colores las respectivas abstracciones.

- **Torres de Hanoi 12x4**

1	13	25	37
2	14	26	38
3	15	27	39
4	16	28	40
5	17	29	41
6	18	30	42
7	19	31	43
8	20	32	44
9	21	33	45
10	22	34	46
11	23	35	47
12	24	36	48

- **Torres de Hanoi 14x4**

1	15	29	43
2	16	30	44
3	17	31	45
4	18	32	46
5	19	33	47
6	20	34	48
7	21	35	49
8	22	36	50
9	23	37	51
10	24	38	52
11	25	39	53
12	26	40	54
13	27	41	55
14	28	42	56

- **Torres de Hanoi 18x4**

1	19	37	55
2	20	38	56
3	21	39	57
4	22	40	58
5	23	41	59
6	24	42	60
7	25	43	61
8	26	44	62
9	27	45	63
10	28	46	64
11	29	47	65
12	30	48	66
13	31	49	67
14	32	50	68
15	33	51	69
16	34	52	70
17	35	53	71
18	36	54	72

➢ Rojo: group_1.txt

➢ Azul: group_2.txt

Algoritmos informados

Se estudiaron la búsqueda de soluciones óptimas con los siguientes algoritmos informados: A* con eliminación retardada de duplicados (DDD) e IDA* con eliminación parcial de duplicados.

El tiempo máximo de ejecución para los casos de prueba en cada problema fue de 15 minutos. En el siguiente enlace se encuentran los resultados para cada puzzle usando los 3 algoritmos mencionados: Algoritmos informados A* e IDA* - Hojas de cálculo de Google

1. **N-puzzles**

- **15-puzzle**

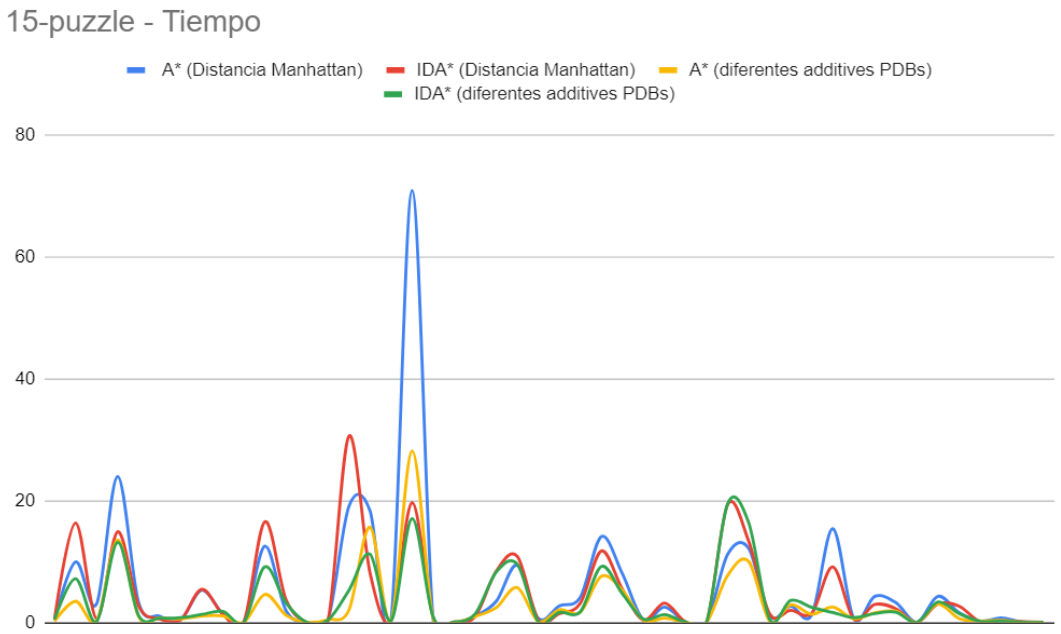
  Para este puzzle, se mató el proceso al emplear el algoritmo de A* con la heurística de distancia Manhattan en el 49-avo estado inicial, esto es debido al consumo de recursos, particularmente de la memoria, que utiliza el algoritmo A*.

a. Gráfica de cantidad de estados iniciales para un determinado distancia donde encuentra al estado objetivo:

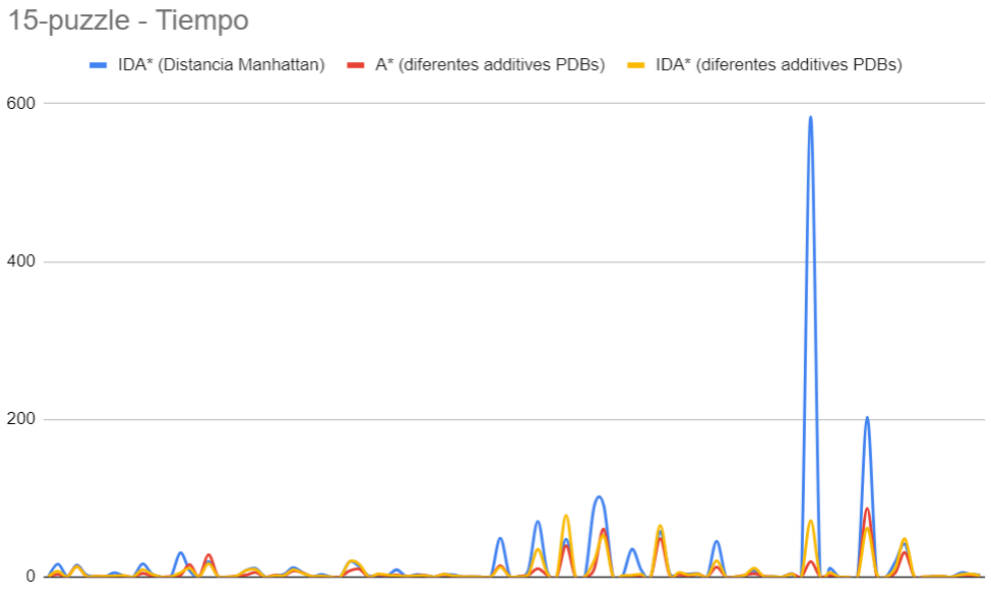
De esta gráfica, podemos observar que las pruebas de casos que fueron proporcionadas en la carpeta benchmarks tienen una profundidad entre 44 y 66. Y hay más estados iniciales donde encuentran el estado objetivo a una distancia igual a 54 y 57.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.

Para los primeros 48 estados iniciales:



Para los otros 52 estados:



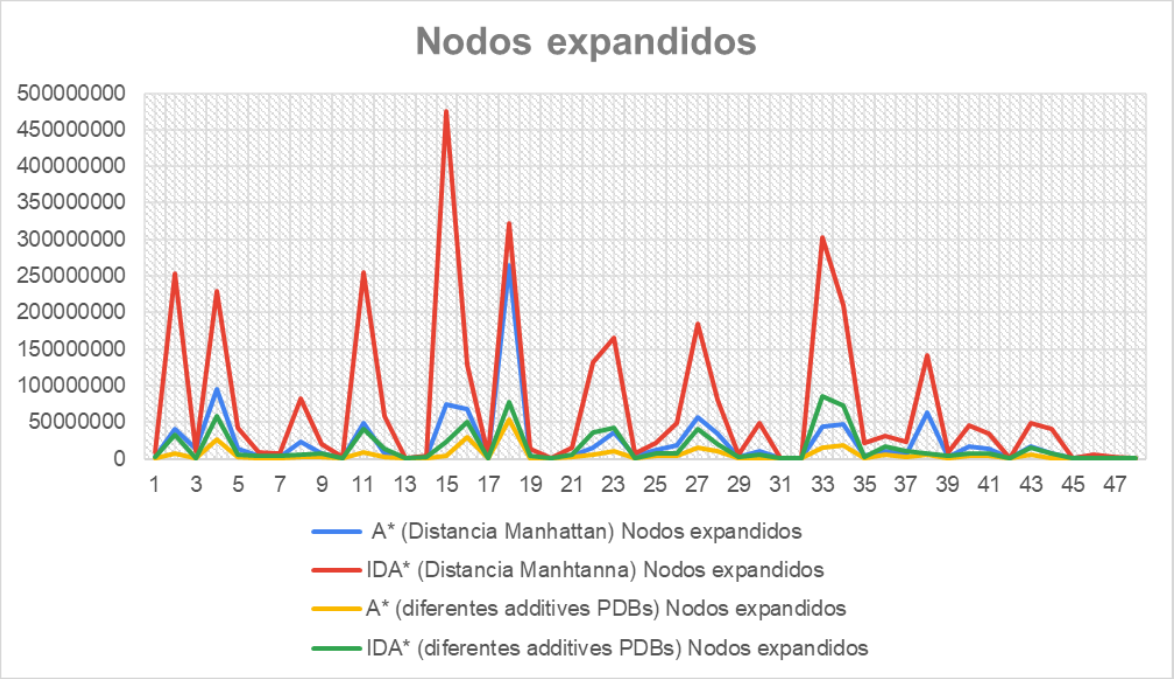
De estos gráficos, podemos observar que IDA* con la heurística Distancia de Manhattan tiene un tiempo mucho mayor que en los otros casos, donde el caso que tarda más es con el siguiente estado inicial de un 15-puzzle:

14	10	2	1
13	9	8	11
7	3	6	12
15	5	4	b

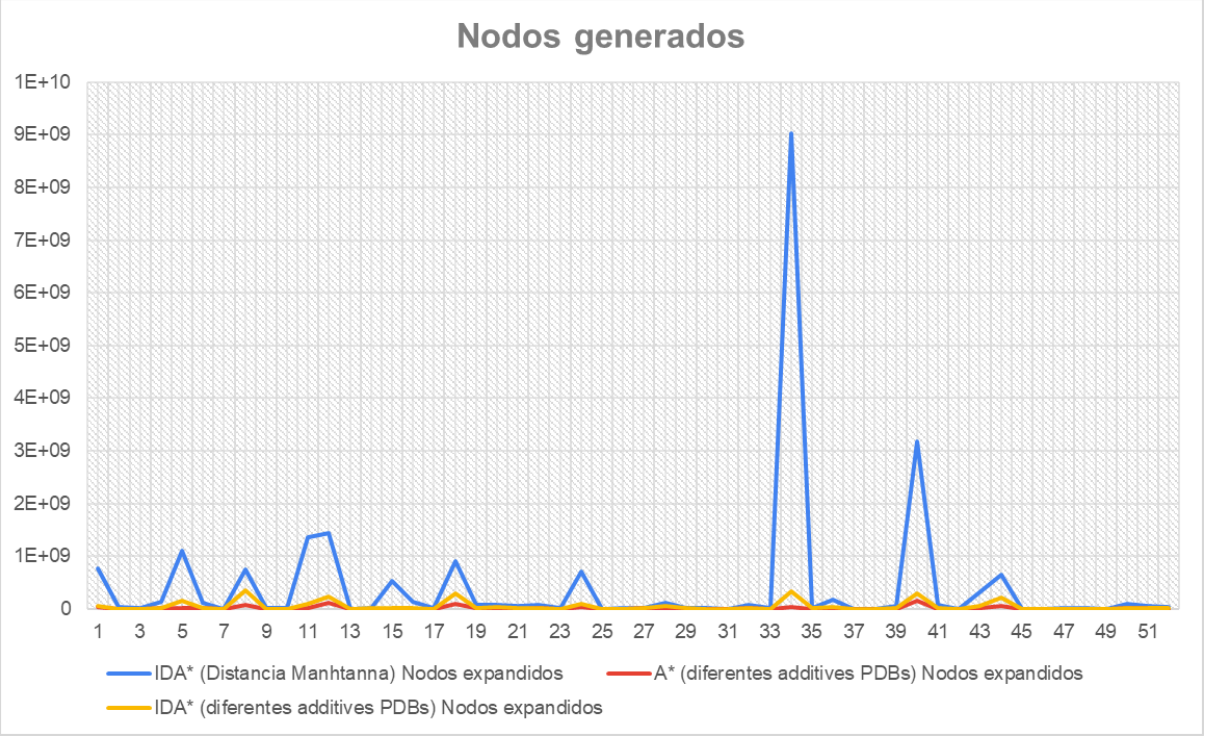
Con un tiempo de 583.411981 segundos, aproximadamente casi 10 minutos en encontrar el estado objetivo.

c. Gráfico de nodos expandidos

Para los primeros 48 pruebas de casos:

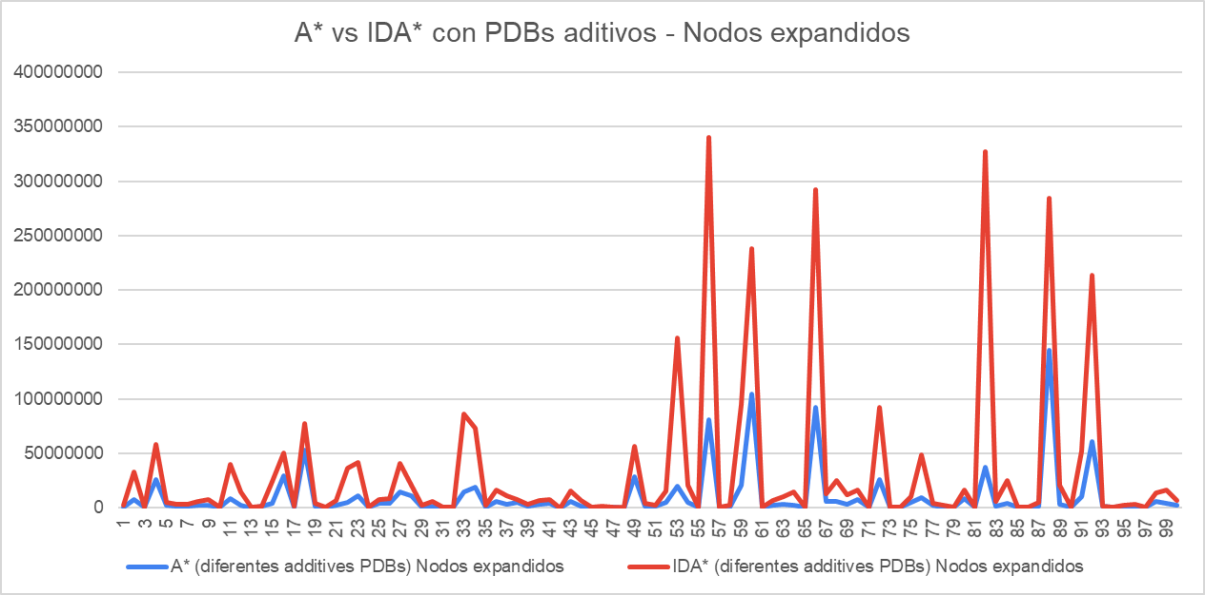


Y para los casos restantes:



Podemos observar que IDA* con Distancia de Manhattan tiene mayor cantidad de nodos expandidos, lo cual es muy probable que esta sea la razón por la cual tarda mucho tiempo en alcanzar un estado objetivo para un determinado estado inicial.

Otro detalle importante es que empleando la heurística con PDBs additives para IDA* e A* es más eficiente que utilizar la distancia de Manhattan, ya que en la mayoría de los casos expande menos cantidad de estados.

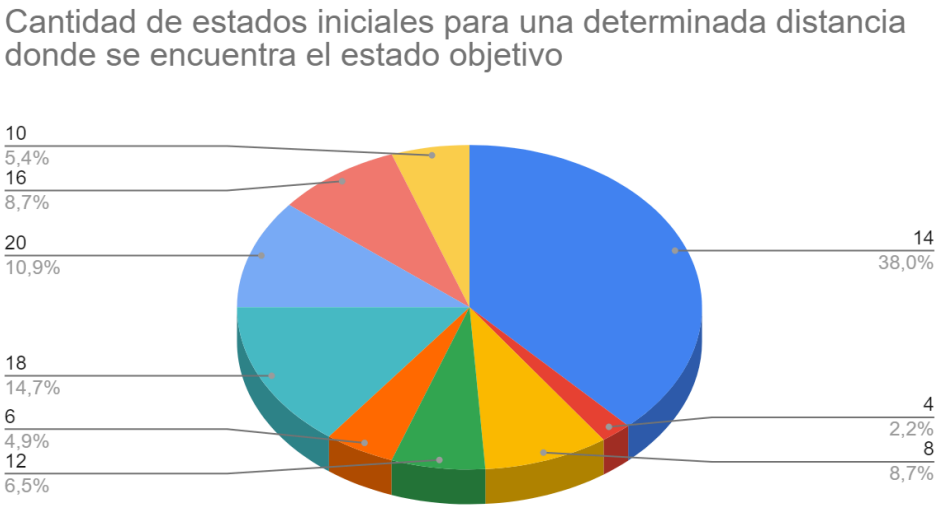


Y entre A* e IDA*, ambos empleando la heurística de PDBs aditivos, podemos ver del gráfico que A* expande menor cantidad de estados que IDA*.

- **24-puzzle**

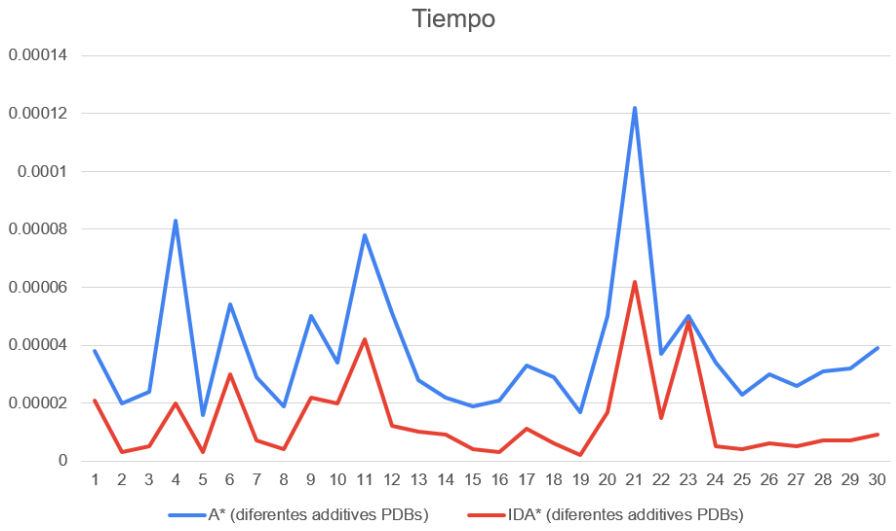
  Todos los casos de pruebas dados en la carpeta benchmarks o bien no lograron alcanzar el estado objetivo antes del tiempo límite establecido o bien el proceso fue matado por falta de recursos. Y para ello, se generaron 30 pruebas de casos fáciles para que pudieran correr y hacer el análisis de resultados.

a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



Se observa que dentro de estos casos de pruebas generados, el rango de distancias donde alcanza el estado objetivo es entre 4 y 20, donde gran parte de estos pruebas de casos fáciles están a una distancia 14 del estado objetivo.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.

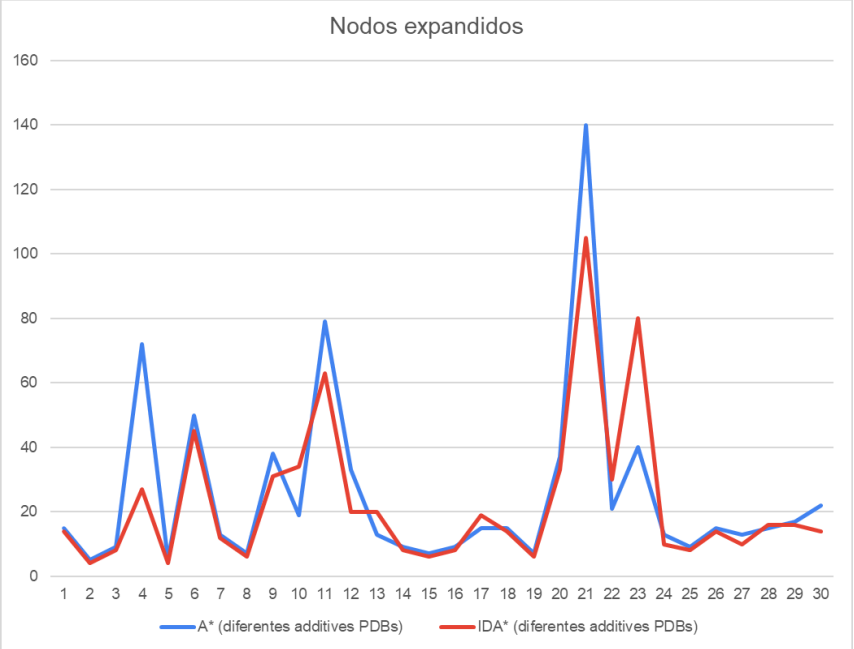


La línea azul, que representa a A* con diferentes PDBs aditivos muestra fluctuaciones a lo largo del tiempo, se observan picos notables alrededor de los casos de pruebas #4, 11 y 21 que superan casi los 0,00008 segundos, donde el caso #21 es el que tiene mayor tiempo, superando los 0,00012 segundos.

Para la línea roja, que indica a IDA* con la misma heurística dicha anteriormente, también fluctúa a lo largo del tiempo pero no tanto como la línea azul. El pico más significativo de la línea roja ocurre también en el caso #21.

También se puede presenciar que para cualquiera de estos 30 casos de pruebas, IDA* siempre toma menor tiempo en alcanzar el estado objetivo que A*.

c. Gráfica de nodos expandidos



Podemos ver que en los primeros 9 casos, los estados expandidos en IDA* es menor que la de A*. Además, en ambos algoritmos, donde se expande más nodos para encontrar el estado objetivo es en el caso #21, que supera más de 100.

También cabe destacar que en el caso #23, existe una gran cantidad de nodos expandidos en IDA* que A*, mientras que en el caso #4, A* tiene más nodos expandidos que IDA*.

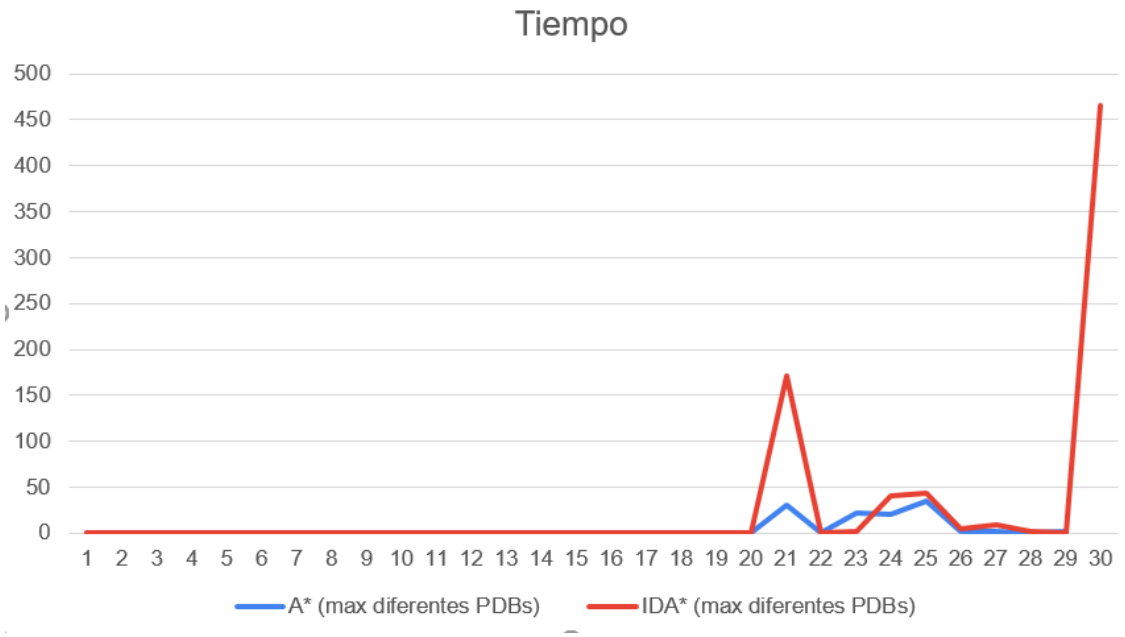
2. Cubo de Rubik

Para este problema, solo se pudo conseguir el estado objetivo para los primeros 30 estados iniciales que fueron dados en la carpeta benchmarks del proyecto. Esto es en el caso de IDA* con máx de corner PDBs y 2 edge PDBs, que luego del 30-avo caso, los más casos de pruebas terminaron con tiempo de ejecución excedido. Mientras que A* con la misma heurística se logró solamente para los primeros 29 estados iniciales, a partir del 30-avo caso el proceso murió por falta de recursos de la computadora.

a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.

La distancia en que encuentra el estado objetivo para estos primeros 30 casos están entre 3 y 14, donde la mayoría de ellos logran su estado objetivo con una profundidad igual a 8, 12 y 13.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.



Podemos ver que para el último caso, el tiempo de IDA* alcanza casi 500 segundos y es el que tarda más tiempo. Otro detalle que podemos ver es que la línea azul que representa a A* permanece relativamente constante para cada estado inicial, con pequeñas fluctuaciones pero sin cambios significativos en su valor, a diferencia de la línea roja que muestra una variación significativa.

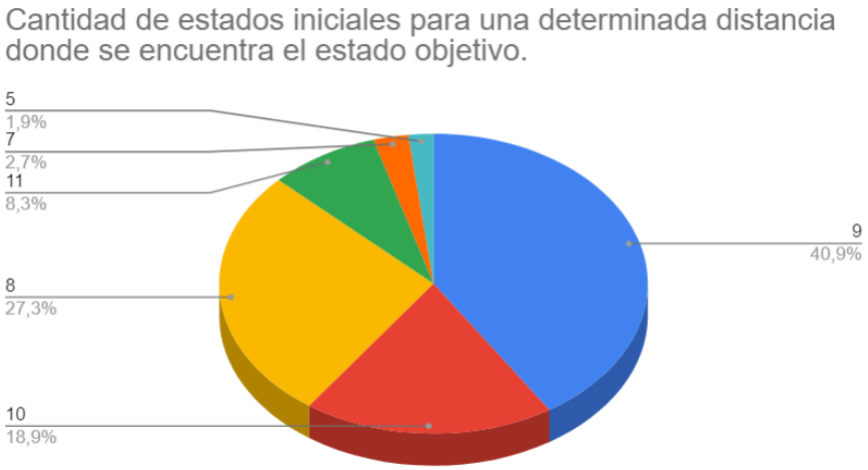
c. Gráfica de nodos expandidos

El gráfico es similar al gráfico anterior del tiempo entre A* e IDA*. Del mismo modo, el tiempo de IDA* muestra una variación significativa entre los casos #20 y #22, y entre #23 y #26, y termina con un número muy grande de nodos expandidos para el caso final (#30) con casi 70.000.000 estados.

3. Top-Spin

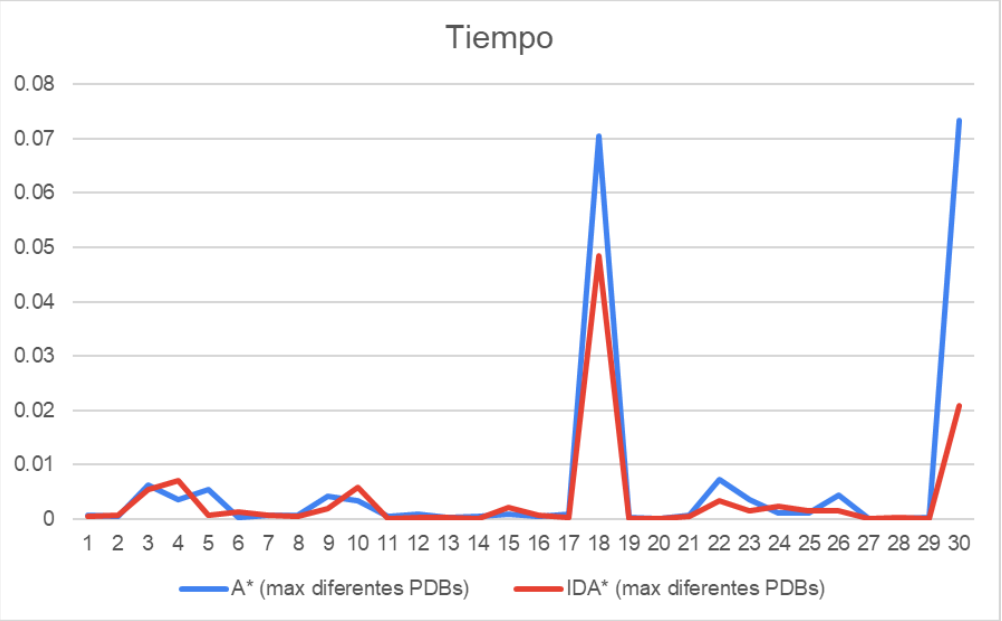
- **Top-Spin 12-4**

a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



La distancia al estado objetivo para los 30 casos de pruebas generados, varían entre 5 y 11, el cual, la mayoría de los casos lograr encontrar el estado objetivo a una distancia igual a 9 y son muy pocos casos en donde logran encontrarlo a una distancia entre 5 y 7.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.

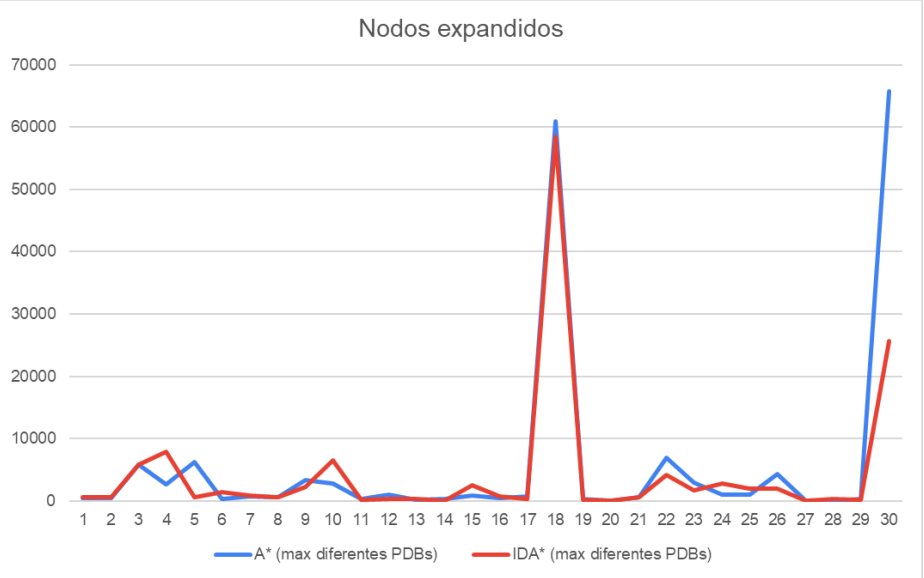


En los primeros 17 casos de pruebas y entre los casos #19 y #29 se observa que el tiempo no supera los 0,01 segundos para ambos algoritmos, en los otros casos si lo superan, especialmente el caso #30 donde tomás más de 0,07 segundos (A* con el máximo de diferentes PDBs).

De igual manera, se observa picos significativos en el caso # 28 para ambos algoritmos.

Pero gran parte de los pruebas de casos, podemos apreciar que IDA* toma menos tiempo que emplear A*.

c. Gráfica de nodos expandidos

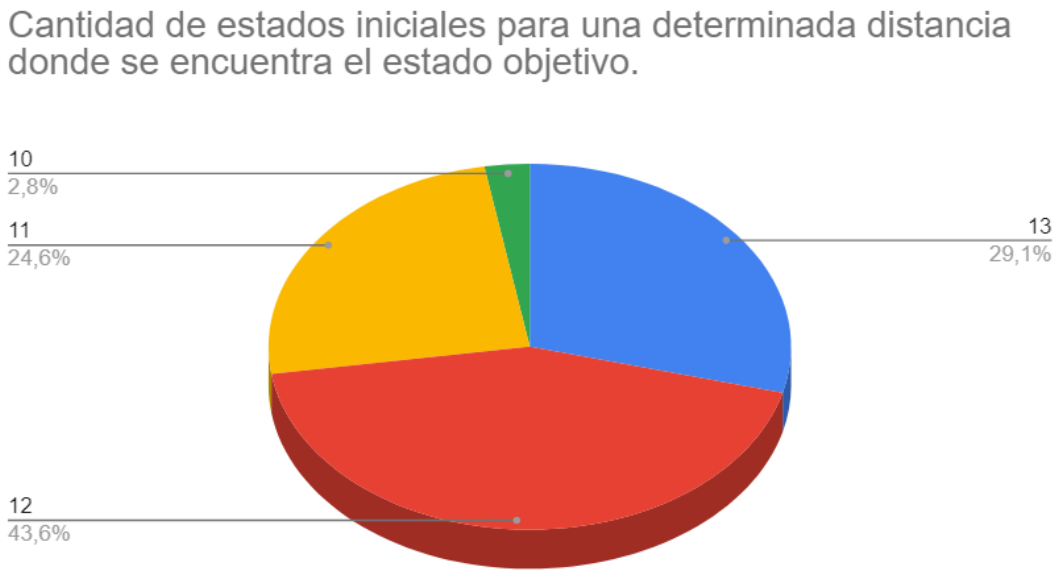


De la misma manera que en el gráfico anterior, se observa que en los primeros 17 casos de pruebas y entre los casos #19 y #29 se expanden menos de 10.000 estados en ambos algoritmos. Mientras que en el caso #18, se observa una cantidad significativa de nodos expandidos, ambos algoritmos alcanzan alrededor de 60.000 nodos expandidos.

Para el último caso de prueba, existe una diferencia muy grande entre la cantidad de nodos expandidos por ambos algoritmos, donde A* ha expandido casi 70.000 estados mientras que IDA* se ha expandido a menos de 30.000 estados.

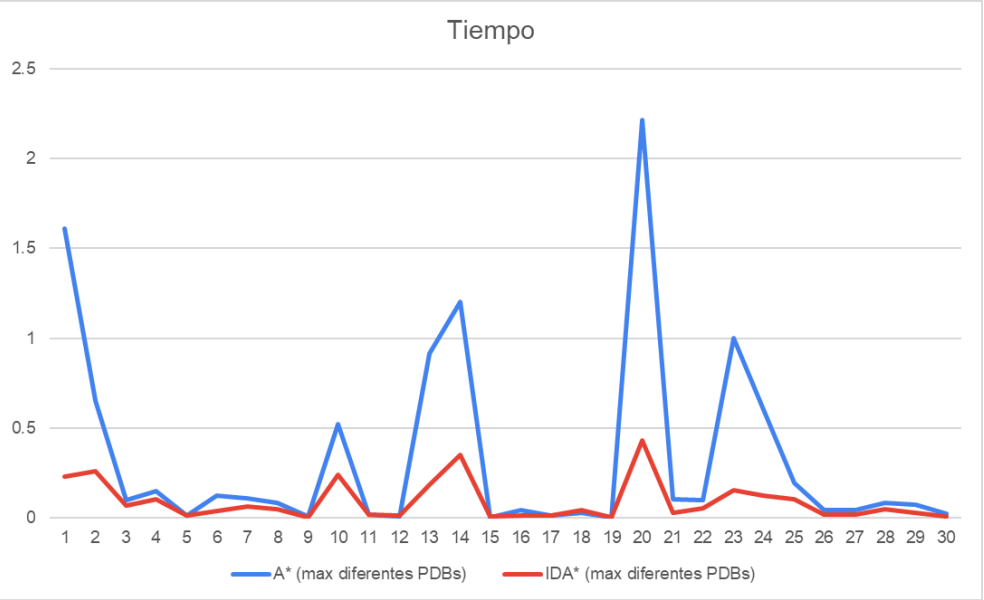
- **Top-Spin 14-4**

a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



Para este problema que cuenta con 30 casos de pruebas generados, se observa que la distancia con que alcanza al estado objetivo varía entre 10 y 13, el cual, gran parte de estos casos están a una distancia igual a 12 del estado objetivo, mientras que muy pocos casos logran alcanzarlo en una distancia igual a 10.

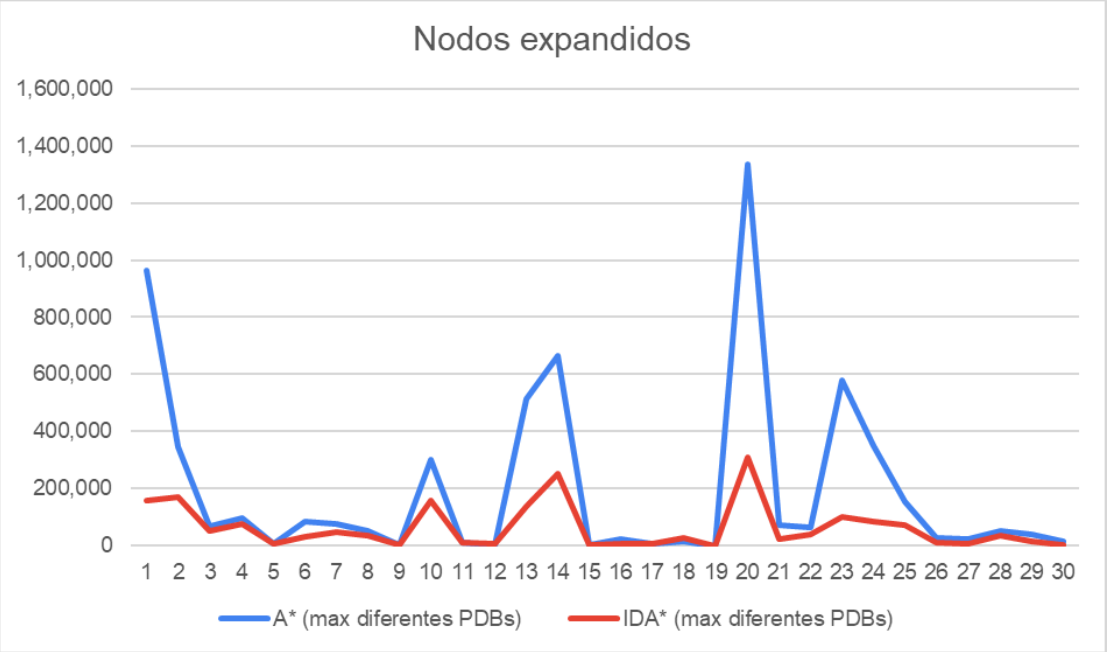
b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.



Con respecto al tiempo, se observa que el tiempo que toma en cada caso de prueba están en un rango entre 0 y 2,5 segundos (sin incluirlos) y se puede apreciar que IDA* siempre es más rápido que A*, ya que en cualquier caso de prueba, la línea roja siempre está por debajo de la línea azul y también siempre está debajo de los 0,5 segundos.

Además, en los casos #1, #14, #20 y #23 tienen picos significativos para el algoritmo A*, donde el #20 es el que tiene el pico más alto que cualquier otro caso de prueba.

c. Gráfica de nodos expandidos



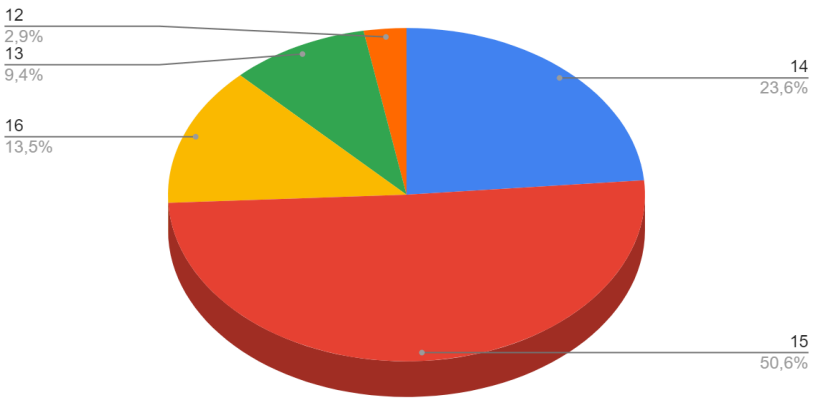
Este gráfico es similar al anterior gráfico que representa el tiempo que tomaba en cada estado inicial para ambos algoritmos.

De la misma manera, se puede ver que IDA* es más rápido que A* en cualquier caso de prueba y que usando A* en el caso #20 se ha expandido casi 1.400.000 de estados para lograr encontrar el estado objetivo.

- **Top-Spin 17-4**

En este problema, se usaron 60 pruebas de casos en total, el cual 3 de ellas no se pudieron correr con ninguno de los algoritmos informados. Y cabe resaltar que solo 2 pruebas de casos pudieron correr con A*, ya que en los otros, el proceso murió.

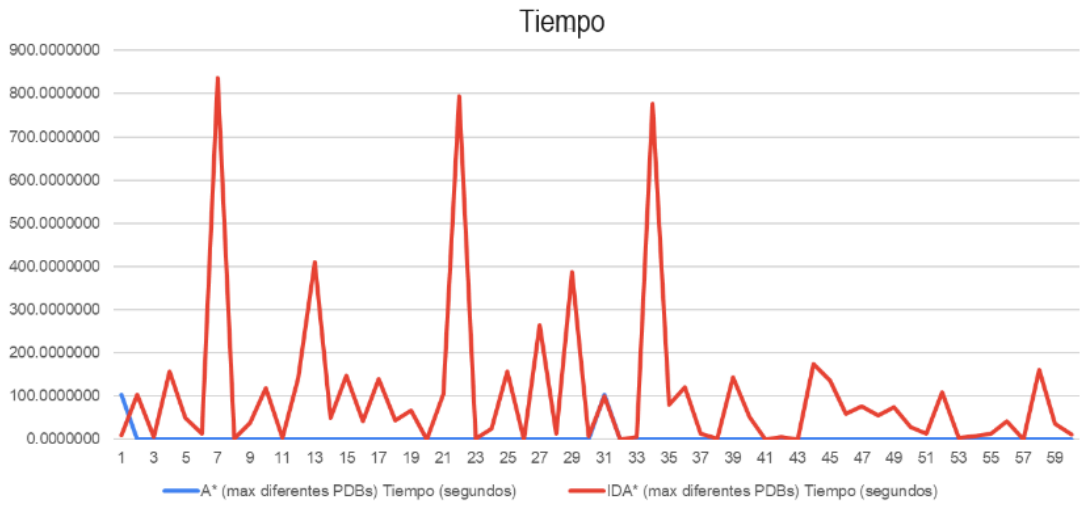
a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



Entre los 57 casos de pruebas generados y que pudieron encontrar el estado objetivo, la distancia con que se encuentra el estado objetivo va entre 12 y 16.

La mitad de los casos encuentran el estado objetivo a una distancia de 15 y existen muy pocos casos que lo encuentran a una distancia de 12.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.

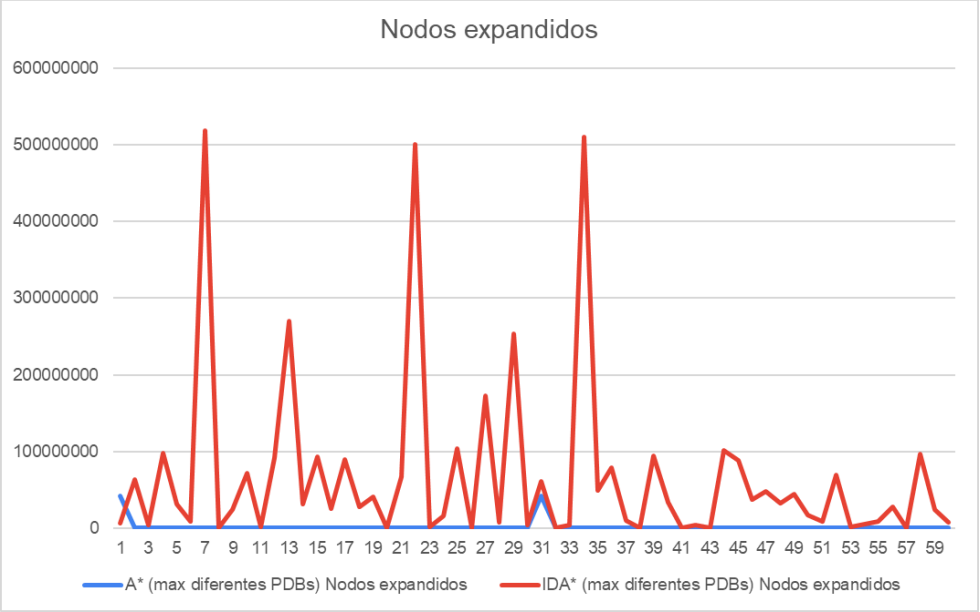


Se observa que la línea azul, que representa A*, comienza alrededor de los 100 segundos y termina bajando hasta llegar a 0. Esto es debido a que solo se pudo correr el primer y el 31avo caso con A*, y en ambos de estos casos llegaron a aproximadamente 103 segundos en alcanzar el estado objetivo.

Para la línea roja, que representa IDA*, se observan varios picos pronunciados, lo cual indica cambios en el tiempo a lo largo de cada estado inicial que se ejecuta, y se tiene que el estado que tarda más en alcanzar el objetivo es el #7 con casi 900 segundos.

También cabe destacar que los casos donde no se pudo encontrar el objetivo con ningún algoritmo informado son el número 20, 43 y 57.

c. Gráfica de nodos expandidos



El gráfico es similar al del tiempo de este problema. Y en los únicos casos donde se pudo correr con A* (los casos #1 y #31) tienen menos de 100.000.000 nodos expandidos

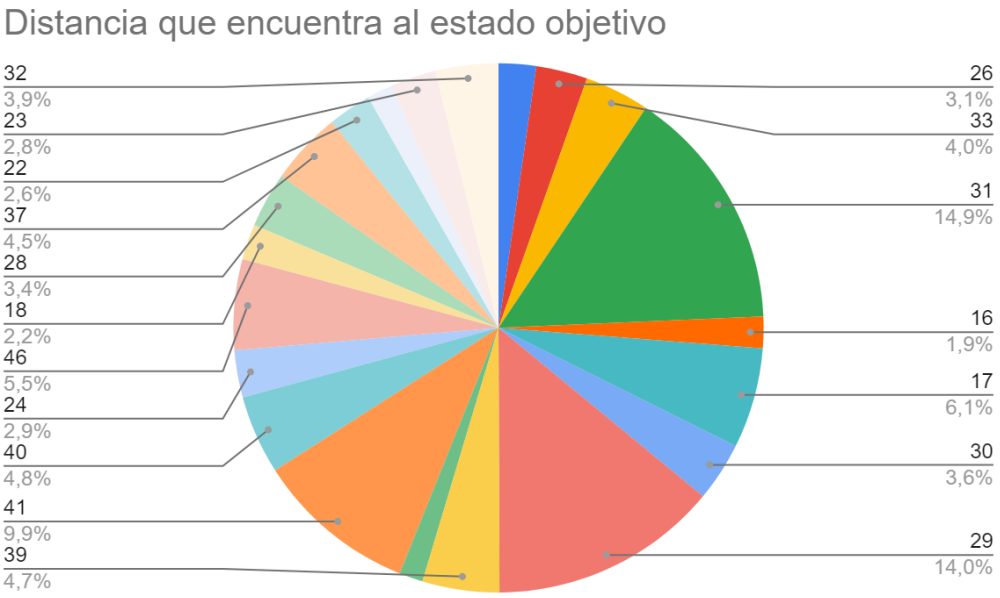
Para la línea roja, que representa IDA*, se observan variaciones significativas y varios picos pronunciados, y se tiene que los estados que expande con mayor cantidad de nodos son el #7, #22 y #34 con alrededor de 500.000.000 nodos expandidos.

4. Torre de Hanoi

- **Torre de Hanoi 12x4**

  Para este problema, existen varios pruebas de casos generados que no lograron alcanzar el estado objetivo usando el algoritmo IDA* porque excedieron el límite de tiempo de ejecución.

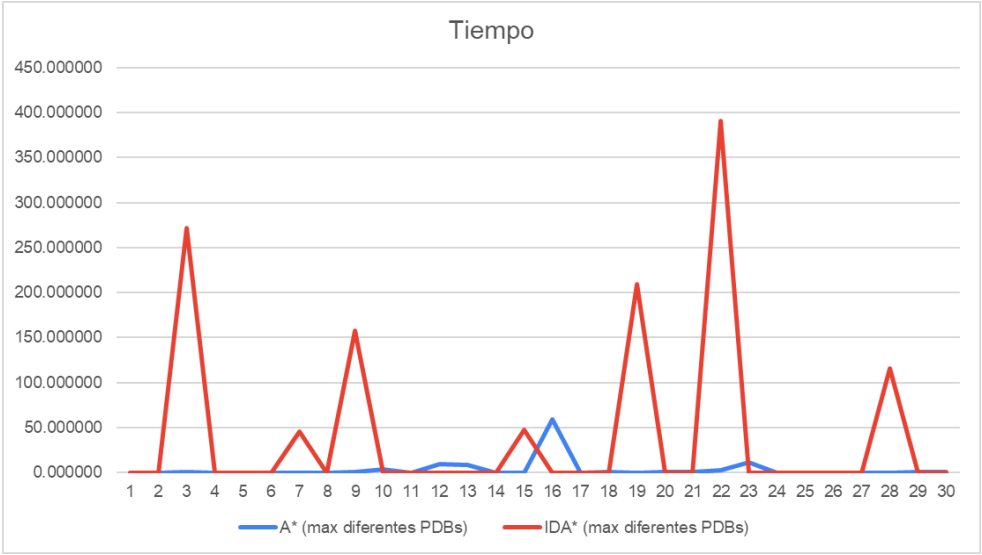
a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



Entre los 30 casos de pruebas generados, la distancia con que se encuentra el estado objetivo va entre 16 y 46.

La mayoría de los casos están a una distancia de 20 y 31 del estado objetivo, y donde hay menos casos es con una distancia de 16.

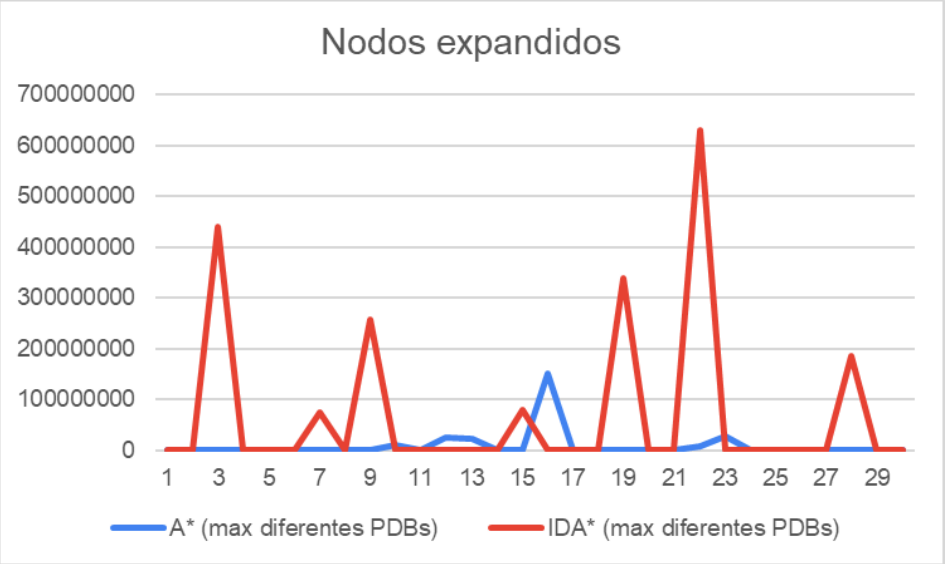
b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.



En los casos con #4, 10, 12, 13, 16, 18, 20, 21, 23, 29 y 30 es donde ocurre el tiempo de límite excedido con el algoritmo IDA*.

También se observa que la línea roja (IDA*) tiene picos más pronunciados que alcanzan hasta 400 segundos (aproximadamente 7 minutos), mientras que la línea azul (A*) permanece casi siempre constante y por debajo o un poco más de los 50 segundos.

c. Gráfica de nodos expandidos



El gráfico es similar a la gráfica del tiempo de este mismo problema, solo que los nodos expandidos oscilan entre 0 y 700.000.000.

Del mismo modo, cabe destacar que en los casos con #4, 10, 12, 13, 16, 18, 20, 21, 23, 29 y 30 es donde ocurre el tiempo de límite excedido con el algoritmo IDA*.

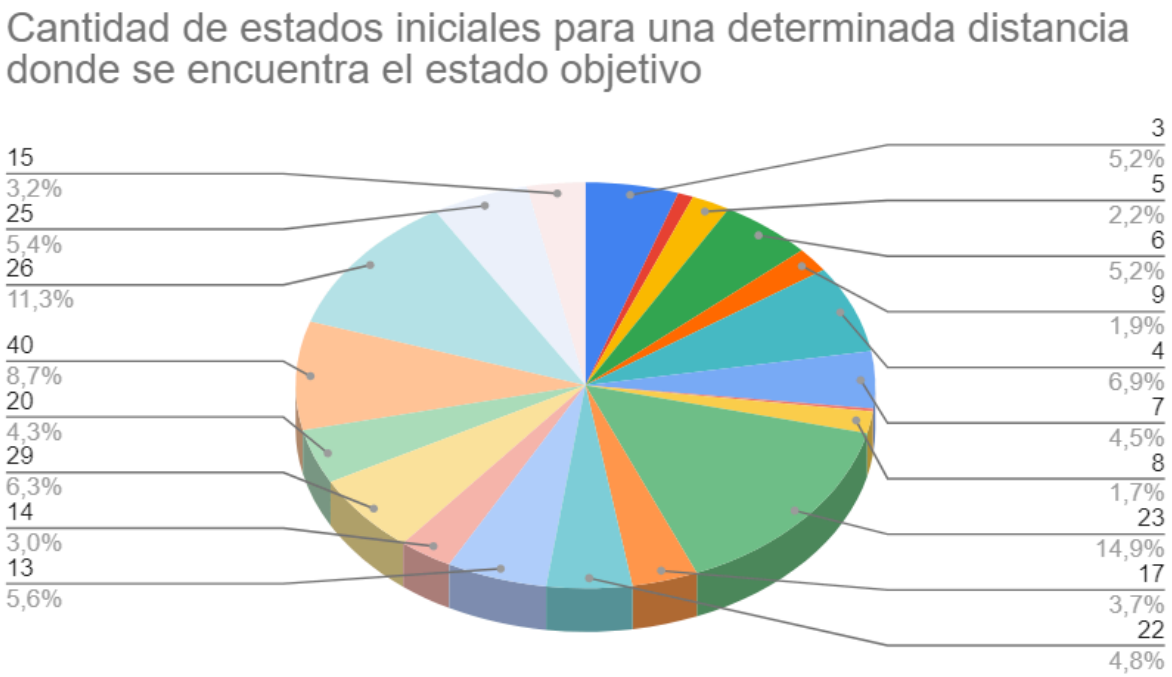
También se observa que la línea roja (IDA*) tiene picos más pronunciados que alcanzan de hasta 700.000.000 estados expandidos, mientras que la línea azul (A*) permanece casi siempre constante y por debajo de los 100.000.000 estados expandidos, excepto para el caso #16 que supera un poco de esa cantidad de estados expandidos.

En este sentido, A* es más rápido y más eficiente que IDA* al emplearlo en este problema.

- **Torre de Hanoi 14x4**

  Para este problema, en los últimos 9 casos de pruebas generados no se logró alcanzar el estado objetivo usando el algoritmo IDA* porque todos ellos excedieron el límite de tiempo de ejecución.

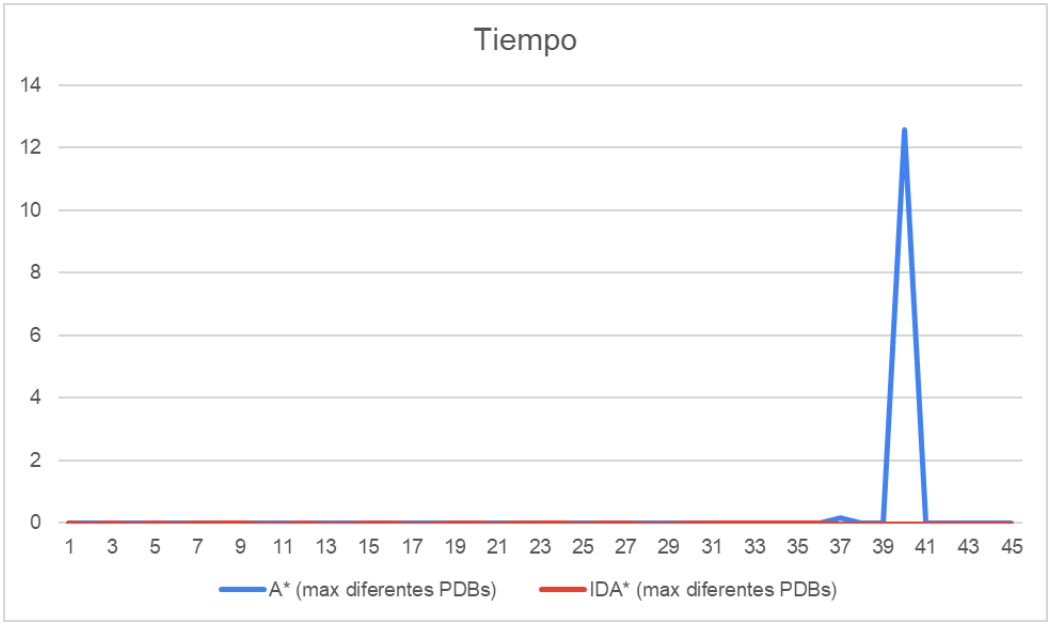
a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.



Entre los 34 casos de pruebas generados donde se pudo conseguir el estado objetivo, la distancia con que se encuentra el estado objetivo de estas pruebas de casos va entre 3 y 40.

La mayoría de los casos están a una distancia de 23 y 26 del estado objetivo, y donde hay menos casos es con una distancia de 4 y 8.

b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.



La línea roja, que representa IDA* (máx diferentes PDBs) se mantiene constante cerca del cero del eje Y a lo largo de todo el gráfico, pero a partir del 37-avo ocurre el tiempo de límite excedido empleando el IDA*.

Mientras que la línea azul, que representa A* (máx diferentes PDBs), también permanece cerca de los cero segundos pero muestra un pico abrupto y significativo alrededor del caso de prueba #40, alcanzando casi hasta 13 segundos antes de volver cerca de los cero segundos.

c. Gráfica de nodos expandidos

El gráfico es similar a la anterior, tenemos nuevamente que la línea roja, que representa IDA* (máx diferentes PDBs) se mantiene constante cerca del cero del eje Y a lo largo de todo el gráfico. Pero a partir del 37-avo ocurre el tiempo de límite excedido empleando el IDA*.

Mientras que la línea azul, que representa A* (máx diferentes PDBs), también permanece por debajo de los 5.000.000 nodos expandidos pero muestra un pico abrupto y significativo alrededor del caso de prueba #40, alcanzando casi hasta 35.000.000 nodos expandidos antes de volver debajo de los 5.000.000 del eje Y.

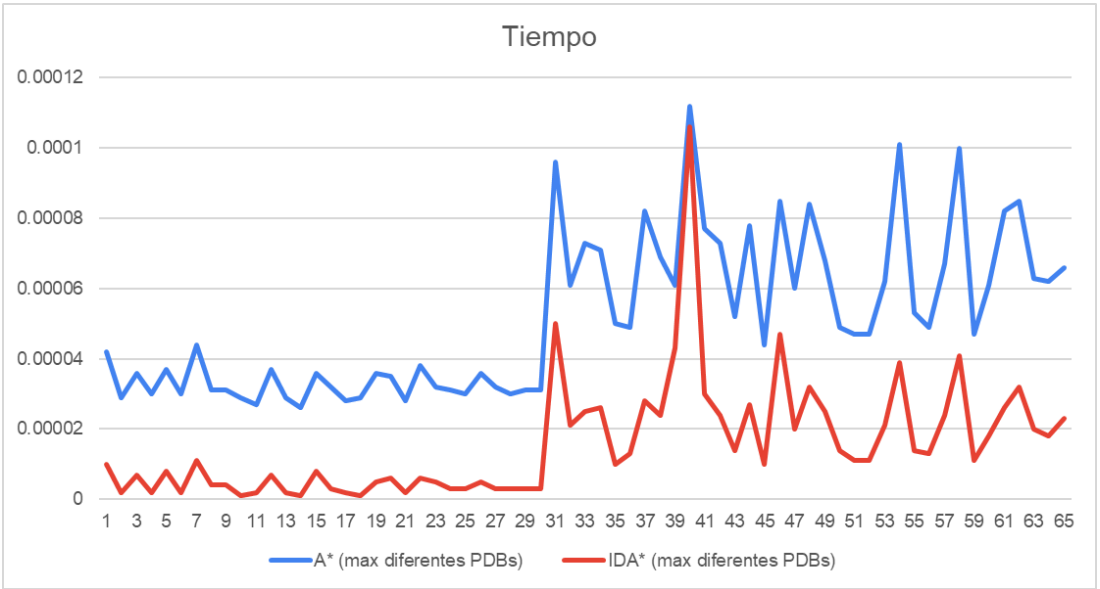
También cabe resaltar que solamente hay dos casos de pruebas que superan los 27 nodos expandidos en A*, mientras que los otros casos tienen menor o igual a 27 nodos expandidos.

- **Torre de Hanoi 18x4**

a. Gráfico de la cantidad de estados iniciales para una determinada distancia donde se encuentra el estado objetivo.

Entre los 65 casos de pruebas generados que incluye tanto casos de pruebas fáciles como difíciles, la distancia con que se encuentra el estado objetivo va entre 3 y 40. La mayoría de los casos están a una distancia de 22 y 26 del estado objetivo, y donde hay menos casos es con una distancia de 3, 6, 7 y 25.

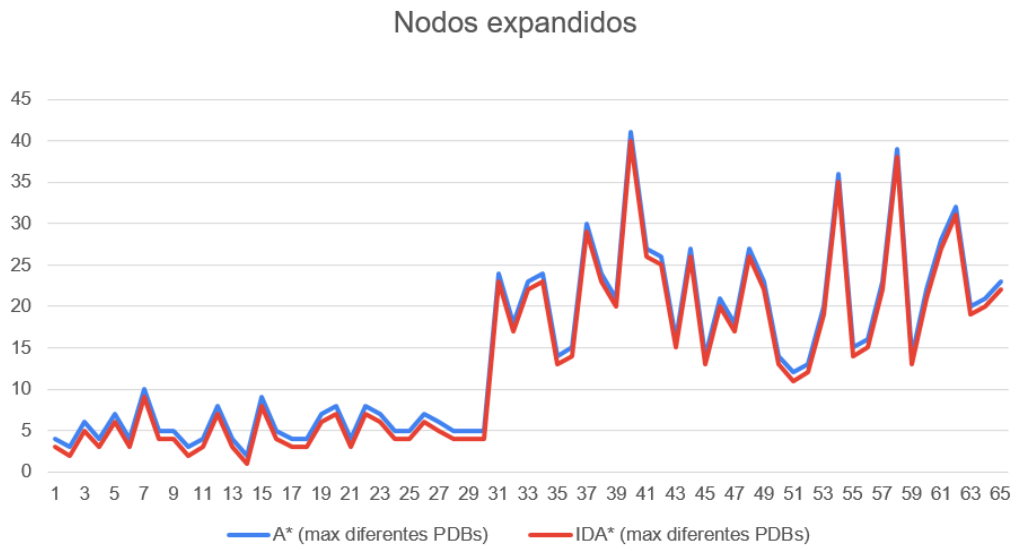
b. Gráfica de cuánto tiempo toma cada algoritmo para llegar al estado objetivo.



Se observa que ambas líneas tienen varios picos y valles, lo que indica cambios en el tiempo a lo largo de cada estado inicial. La línea azul, que representa A* con máx diferentes PDBs, generalmente mantiene un valor más alto que la línea roja, que representa IDA* con máx diferentes PDBs, en la mayoría de las partes del gráfico.

Esto sugiere que para este problema, IDA* es generalmente más rápido que A*, el cual ambos usan la misma heurística.

c. Gráfica de nodos expandidos



Al igual que con el gráfico anterior, se observa que ambas líneas tienen varios picos y valles, lo que indica cambios en la cantidad de nodos generados a lo largo de cada prueba de caso y además la línea azul (A*) se mantiene casi siempre un poco encima de la línea roja, el cual es la principal diferencia de esta gráfica con la anterior donde se observa grandes diferencias entre los tiempos de ambos algoritmos informados.

Casos de prueba

Dado el enunciado de este proyecto, para los casos de prueba de los n-puzzles y Cubo de Rubik hemos utilizado los que están en la carpeta benchmarks. Sin embargo, también se generaron pruebas de casos para el 24-puzzle ya que no se pudieron ejecutar los casos de pruebas que están en la carpeta que se mencionó anteriormente.

Mientras que para los otros problemas, los casos de pruebas fueron generados con el programa global/generator.cc, recibido de la distribución de PSVN, al principio con un seed=30, n=30 y max-depth: 3000. Esto es para generar 30 casos de pruebas aleatorias tanto fáciles como difíciles y hasta una profundidad de 3000.

Sin embargo, solo se pudo usar estos parámetros para usar estos casos de pruebas con A* e IDA* en los problemas de Top-Spin 12-4 y Top-Spin 14-4, ya que o terminaban con tiempo de límite excedido o se mataba el proceso.

Posteriormente, para generar casos de prueba de los otros problemas que faltan, se usaron las siguientes configuraciones del generador de casos:

- seed=5, n=30, max-depth=30: Para generar 30 casos de pruebas aleatorias fáciles para: 24-puzzle, Top-Spin 17x4 y los 3 Torres de Hanoi.
- seed=100, n=15, max-depth=1000: Para generar 15 casos de pruebas aleatorias de nivel medio para Top-Spin 17x4 y los 3 Torres de Hanoi.
- seed=150, n=20, max-depth=1250: Para generar 20 casos de pruebas aleatorias difíciles para los 3 para Top-Spin 17x4 y los 3 Torres de Hanoi.

No se generó pruebas de nivel medio ni difíciles para el problema 24-puzzle ya que ninguno pudo correr con los algoritmos A* ni con IDA*.

Dichos casos nos ayudaron a comprobar la robustez de los algoritmos y la optimalidad de las abstracciones realizadas para los algoritmos informados.

CONCLUSIÓN

En resumen, para los hallazgos de los resultados de los algoritmos de búsquedas de profundidad aplicados en los problemas concretos sugieren que la poda es una estrategia efectiva para mejorar la eficiencia de los algoritmos de búsqueda, al reducir tanto la cantidad de estados generados como el grado de ramificación. Además, a medida que aumenta la profundidad, el número de estados generados de los tres algoritmos crece exponencialmente, lo que puede llevar a un crecimiento no viable en términos de tiempo y recursos computacionales.

Sin embargo, IDDFS supera esta limitación al realizar una búsqueda en profundidad a diferentes niveles de profundidad de manera iterativa. Comienza con una búsqueda en profundidad limitada a una profundidad fija y, si no encuentra la solución, aumenta gradualmente la profundidad permitida. Esto permite que IDDFS alcance profundidades significativamente mayores en comparación con UCS y BFS, al tiempo que limita el número total de estados generados. Por lo tanto, IDDFS logra un equilibrio entre la eficiencia en la utilización de recursos y la capacidad para explorar profundidades mayores en el espacio de búsqueda.

Por otra parte, la exploración de algoritmos informados como A* e IDA* para la resolución de problemas ha revelado la influencia crucial del tipo de problema y la complejidad de los casos de prueba en su rendimiento. La longitud de las soluciones y la elección de heurísticas desempeñan roles significativos en la eficacia de estos algoritmos. Por ejemplo, en casos donde las soluciones son de longitud considerable, IDA* puede enfrentar desafíos debido a la necesidad de explorar extensamente el espacio de búsqueda. Sin embargo, para problemas con soluciones más cortas, su desempeño puede ser superior, como por ejemplo, para los tres problemas de Top-Spin, donde muestra un rendimiento generalmente más rápido y más eficiente que A* en la expansión de nodos, alcanzando el estado objetivo en menor cantidad de estados expandidos.

Cabe resaltar también que la utilización de memoria puede afectar el rendimiento del algoritmo A* de varias maneras. En general, el rendimiento del algoritmo A* puede verse afectado por la cantidad de memoria disponible, ya que un espacio de búsqueda más grande puede requerir más memoria para almacenar los estados explorados y los nodos en la frontera. Si la memoria es limitada, el algoritmo A* puede enfrentar dificultades para manejar espacios de búsqueda extensos, lo que puede resultar en un rendimiento más lento o en la imposibilidad de encontrar una solución óptima.

Además, la selección de heurísticas adecuadas puede mejorar significativamente el rendimiento de los algoritmos. Por ejemplo en el 15-puzzle, se observó que la heurística de la distancia de Manhattan es menos efectiva que las diferentes PDBs aditivas en la mayoría de los casos, ya que en la mayoría de los casos las diferentes PDBs aditivas expanden menos cantidad de estados. Además, se encontró que para este mismo problema, entre A* e IDA* empleando la heurística de PDBs aditivas, A* expande una menor cantidad de estados que IDA*.

En este sentido, la comprensión de estas dinámicas no solo enriquece nuestro conocimiento teórico, sino que también nos proporciona herramientas prácticas para abordar una variedad de problemas de manera más eficiente.

En última instancia, esta investigación subraya la importancia de considerar las características específicas del problema y elegir el enfoque algorítmico más adecuado dentro del marco de la Inteligencia Artificial. Al entender las fortalezas y limitaciones de los algoritmos como A* e IDA*, podemos aprovechar al máximo su potencial en la resolución de problemas reales, avanzando así en el campo de la IA hacia resultados más efectivos y satisfactorios.