Volume 2 Issue 1 ISSN 2631-4444

Aplicaciones de Algoritmos de búsquedas de costo mínimo en juegos.

Pedro Rodriguez - 15-11264 and Daniel Pinto - 15-11139

1 Algoritmos utilizados

Cada uno de los juegos presentados en el trabajo, presenta el mismo tipo de problema: ¿Como navegar desde una configuración inicial a una configuración final utilizando la mínima cantidad de transiciones?

Cada instancia presenta al menos 3 soluciones distintas: una utilizando el algoritmo no informado BFS, el cual expande por capas de manera incremental hasta llegar a un estado objetivo. Luego mediante una modificación de A^* basada en Best First Search con eliminación parcial de duplicados, y finalmente mediante una version modificada de IDA^*, tambien basada en Best First Search con eliminacion parcial de duplicados.

2 Specs del equipo usado

Todas las pruebas se corrieron en el mismo equipo, el cual cuenta con los siguientes specs:

- **Procesador:** Ryzen 7 3800x, 3.9GHz.
- **RAM:** 16GB, 3200MHz.
- SSD.

Adicionalmente, para todos los experimentos, tomamos como tiempo máximo de ejecución: 420001ms.

3 15 Puzzle

El juego del 15 puzzle se presenta como una cuadricula 4×4 numerada con casillas del $1-15$, y con un espacio en blanco (Figura 1). El objetivo del juego es ir intercambiando las casillas numeradas con el espacio en blanco, hasta llegar a una configuración ordenada por filas del $1-15$ con el blanco posicionado en la ultima casilla.

Para este algoritmo, se utilizaron como benchmarks, las presentadas en la carpeta benchmarks, apartado 15-puzzle. Adicionalmente, todos los algoritmos corren con la misma cantidad de muestras: 10

3.1 Solución mediante BFS (not prunning)

La Figura (2) muestra el logaritmo de la cantidad de nodos expandidos vs la profundidad del BFS junto a la cantidad de nodos por nivel teorizada en clase:

$$d_n = \frac{1}{5}\left((\phi_1 + 2)^2 \phi_1^{n-2} + (\phi_2 + 2)^2 \phi_2^{n-2}\right)$$

En donde:

Pedro Rodriguez - 15-11264: 15-11264@usb.ve, http://compositionality-journal.org
Daniel Pinto - 15-11139: 15-11139@usb.ve

Figure 1: Un juego de 15 Puzzle

$$\phi_1 = 1 + \sqrt{5}$$
$$\phi_2 = 1 - \sqrt{5}$$

La diferencia entre ambas cifras es de orden 10^1, lo cual prueba de manera experimental la aproximación dada en clase. Mas aún, notamos que ambos log-plots forman una linea recta, lo cual significa que la cantidad de nodos aumenta exponencialmente con la profundidad. Este es el motivo por el cual vemos que todas las ejecuciones de BFS sin pruning **no** encuentran solución: El tiempo promedio obtenido es exactamente 42001ms que es exactamente el tiempo maximo de ejecución.

Los resultados tabulados para la Figura (2) se muestran en la Table (1).

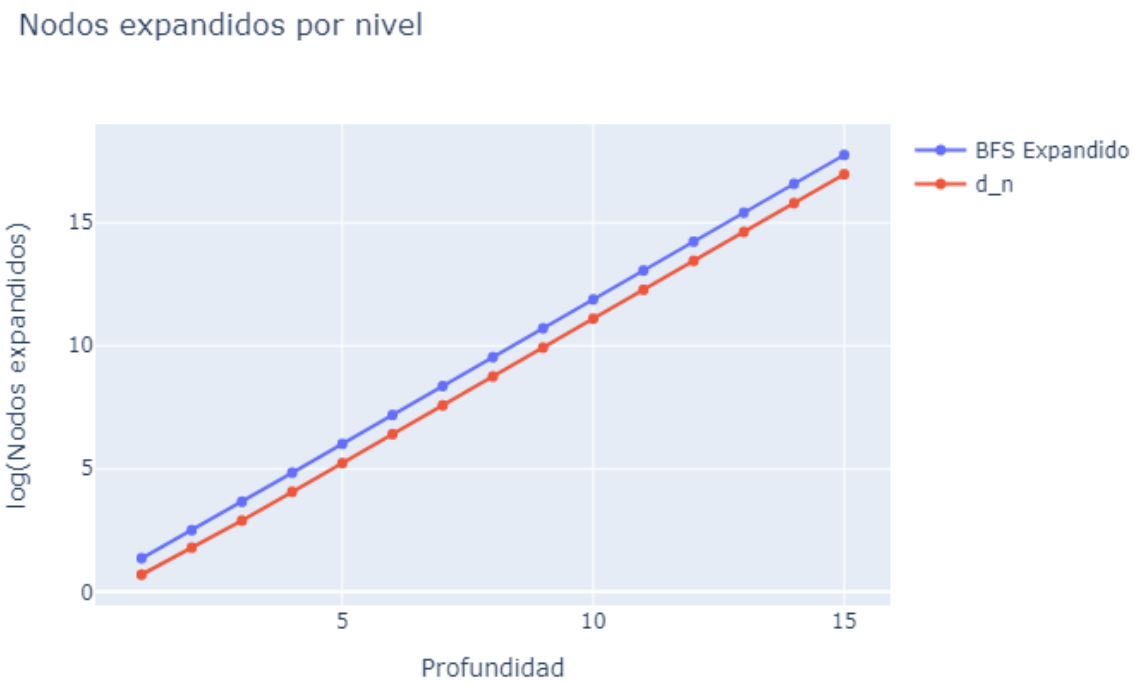

Figure 2: Corrida BFS No pruning

BFS No Pruning

Profundidad	Promedio de nodos expandidos
1	3.9
2	12.2
3	38.8
4	126.1
5	406.4
6	1316.2
7	4257.1
8	13777.9
9	44583.3
10	144277.1
11	466886.5
12	1510880.3
13	4889305.7
14	1.582e7
15	5.120e7
Tiempo Promedio Total (ms)	420001.0

Table 1: Resultados de nodos expandidos y tiempo de corrida promedio para BFS no pruning (10 tests)

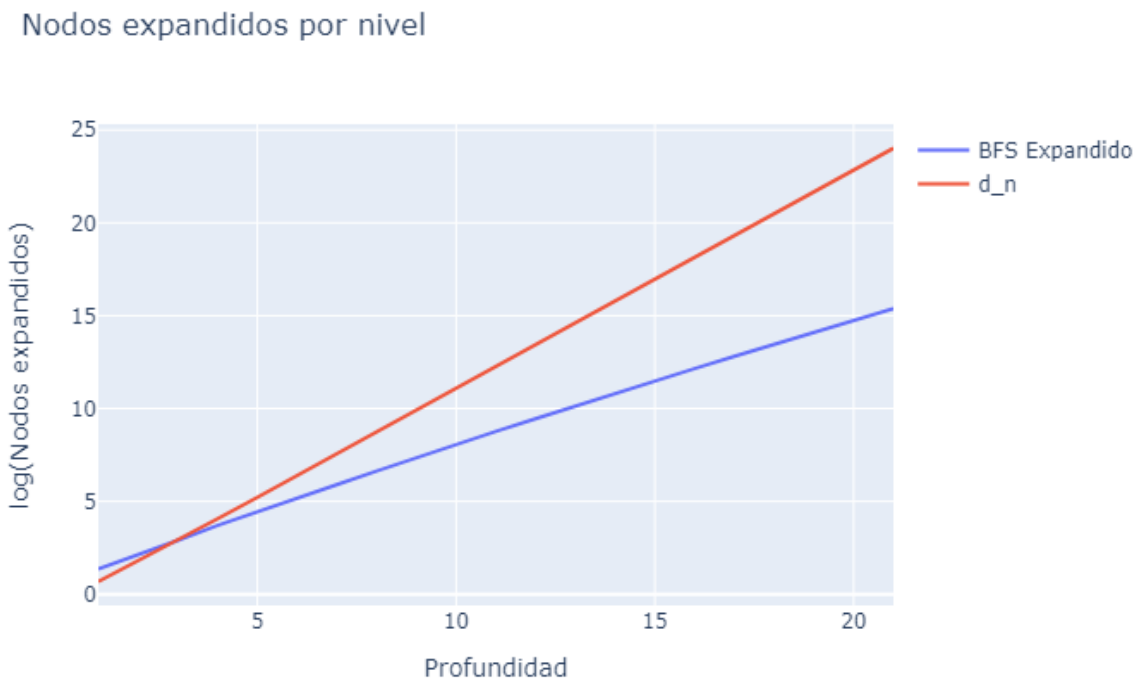

Figure 3: Corrida BFS pruning

3.2 Solución mediante BFS (pruning)

Al agregar prunning al BFS bajo 15 puzzle, notamos en la Figura (3) que la profundidad aumenta considerablemente: pasamos de tener una profundidad máxima de 15 a tener una de 21, para la cual se expanden 4.8 millones de nodos (Véase profundidad a nivel 21 en Tabla (2)).

La ventaja de añadir pruning se hace notoria cuando comparamos la cantidad promedio de nodos expandidos a profundidad 15 (la profundidad máxima de BFS no pruning):

$$\frac{pruning_{15}}{no\_pruning_{15}} = \frac{9.760e4}{5.120e7} \sim e^{-3}$$

Es decir, que BFS no pruning explora 1000 veces la cantidad de nodos que BFS pruning a profundidad 15. De hecho, el beneficio de la poda parcial aumenta a medida que aumentamos la profundidad.

En la Figura (3) observamos que la pendiente de BFS pruning es menor a la de d_n. Dado que estamos en un log-grafo, esto significa que se BFS pruning explora exponencialmente menos nodos mientras la profundidad crece.

Sin embargo, aún con la mejora exponencial que ofrece poda parcial de duplicados, ninguno de los casos de prueba finiquita en el tiempo establecido, debido a que el tiempo promedio de corrida sigue siendo el tiempo máximo.

Los resultados tabulados para la Figura (3) se muestran en la Tabla (2).

BFS Pruning	
Profundidad	Promedio de nodos expandidos
1	3.9
2	8.3
3	18.2
4	40.6
5	86.3
6	176.0
7	367.1
8	761.5
9	1573.9
10	3169.2
11	6400.2
12	12669.5
13	25242.5
14	49599.3
15	97604.1
16	189660.3
17	368376.7
18	706670.8
19	1352667.1
20	2557020.9
21	4813228.0
Tiempo Promedio Total (ms)	**420001.0**

Table 2: Resultados de nodos expandidos y tiempo de corrida promedio para BFS (10 tests)

3.3 Solucion mediante A^* $6-6-3$

Como PDB aditivo, seleccionamos un particionamiento $6-6-3$ mostrado en la Figura (4). La razón por la que seleccionamos este particionamiento es debido que genera dos grupos (PDBs) grandes.

Tener PDBs grandes implica mayor tiempo de compilación, alojar un tamaño exponencial en memoria, pero también implica poseer una mejor heurística: al tener grupos grandes, maximizamos la cantidad de interacciones entre los tiles, y por lo tanto, cualquier solución que tenga como camino parcial alojado en alguno de los grupos va a ser contado por la heurística.

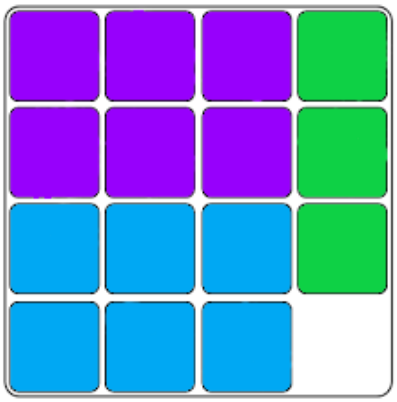

Figure 4: PDB Seleccionado

Debido a que A^* explora nodos a usando una cola de prioridad. Decidimos contar la cantidad de nodos *totales* expandidos:

$$15p\ nodes = 2.357e7$$

Lo cual también genera un tiempo de corrida promedio de $420001ms$. Es decir, que ningún caso acaba.

Esto es un resultado esperado si asumimos que los casos de prueba poseen mucha interacción entre las regiones borde. Puesto que si esto pasa, la heurística seleccionada provee una cota inferior muy laxa y no mejora mucho mas el resultado de BFS con pruning.

Algo notorio de señalar, es la cantidad de nodos totales expandidos. En la Tabla (3), notamos que aúnque ciertamente, A* un tercio de los nodos que BFS no pruning. Tambien expande el doble que BFS pruning.

Que A*/BFS no pruning expandan menos nodos, es debido a que ambos por definicion utilizan poda de resultados parciales, esto quiere decir que hacen un tradeoff, y dejan de expandir resultados utilizando un mecanismo de detección de duplicados. Lo cual suele ser mas costoso que solo transicionar como BFS no pruning (una búsqueda en memoria es constantemente mas costoso que ejecutar una expresión aritmética inlineable). Resultando en menos nodos explorados al finiquitar el máximo de tiempo.

Sin embargo, que A* expanda mas nodos que BFS pruning es un resultado contraintuitivo. Ya que la forma de hacer poda con BFS utiliza solo un diccionario para verificar visitados, mientras que A* también utiliza un diccionario de costes parciales, pero adicionalmente necesita llevar una estructura de nodos por expandir. Lo cual deberia ser asintoticamente mas costoso, y por ende, deberia expandir menos nodos.

3.4 Solución mediante IDA^* $6-6-3$

Para esta solución se utilizo el mismo particionamiento $6-6-3$ mostrado en la Figura (4), y se expandieron un total de $1.214e8$ nodos (vea Tabla (3)). 10 Veces mas nodos que cualquier algoritmo informado, y 1.5 veces mas nodos que BFS no pruning. La razon de esto es que IDA* utiliza un threshold en vez de memoria, lo cual hace las exploraciones menos costosas que A*/BFS con pruning, pero sigue utilizando una heurística para guiar cuales nodos expande.

Sin embargo, al igual que los otros algoritmos, este no encuentra solución, promediando un tiempo de $420001ms$.

3.5 Solucion mediante (ID)A* Manhattan

Con respecto a (ID)A* usando la distancia manhattan como heurística. Vemos que los resultados siguen siendo consistentes con las soluciones de (ID)A* para la heuristica $6-6-3$. Ambos se mantienen a un múltiplo escalar de su contraparte. La razón de que estos expandan mas nodos es porque buscar en memoria un pdb es mas caro que calcular la distancia manhattan.

3.6 Conclusiones 15 Puzzle

Las pruebas con BFS pruning mostraron que el estado objetivo esta a mas de 21 pasos de distancia, mientras que los resultados de A*/IDA* muestran que el particionamiento seleccionado no es óptimo para el conjunto de problemas dado. Una de las razones por las que puede pasar esto, es que el camino de la solución implique muchos intercambios entre las fichas: $7-8-11-12$,

Algoritmo	Nodos totales Expandidos
IDA* $6-6-3$	1.214e8
A* $6-6-3$	2.357e7
A* Manhattan	3.229e7
IDA* Manhattan	3.927e8
BFS no pruning	7.409e7
BFS pruning	1.018e7

Table 3: Cantidad de nodos totales expandidos para 15 Puzzle

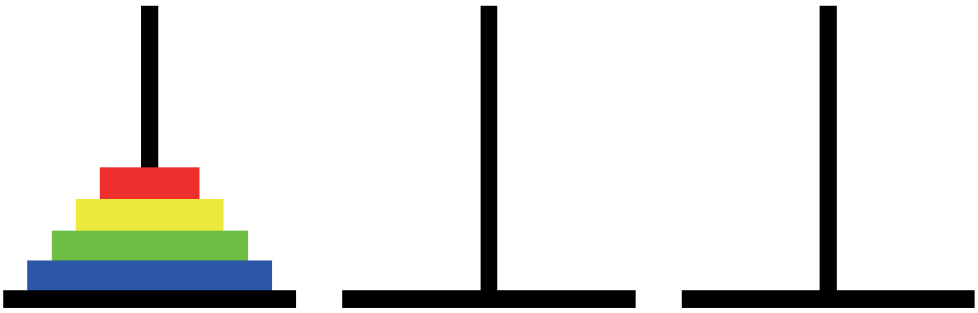

Figure 5: Torres de Hanoi

$5-6-7-9-10-11$, $3-4-7-8-11-12$, las cuales son las regiones borde de los PDB.

4 Torres de Hanoi

Las torres de hanoi es un juego en donde se tiene una serie de discos en longitud creciente ensartados en un asta, y cuyo objetivo es pasar todos los dicos al asta opuesta sin sobreponer un disco de longitud mayor sobre otro de longitud menor.

Para este algoritmo se utilizaron como benchmarks, las presentadas en la carpeta benchmarks, apartado hanoi_XXX. Adicionalmente, todos los algoritmos corren con la misma cantidad de muestras: 5 por nivel de dificultad.

4.1 Solucion mediante BFS (not pruning) 4 hastas 12 discos

Para BFS not pruning, tenemos que pasa todos los tests hasta el Muy Difícil. Por lo tanto nuestro analisis sera con respecto a este.

En la Figura (6) vemos que la cantidad de nodos expandidos sigue el mismo patrón que los demas BFS no pruning: crecimiento exponencial hasta llegar a e^{10} nodos, que es donde encuentra el tiempo máximo.

4.2 Solucion mediante BFS (pruning) 4 astas 12 discos

A diferencia de BFS not pruning, BFS Pruning si acaba para todos los inputs. Y con resultados interesantes.

En la Figura (6) podemos observar la cantidad de nodos expandidos con BFS pruning. A diferencia de los anteriores BFS, este no posee un crecimiento exponencial, sino lineal. Esta es la

BFS No Pruning Hanoi 4-12 (Facil)

Profundidad	Promedio de nodos expandidos
1	6.4
2	22.6
3	83.6
Tiempo Promedio Total (ms)	**55.2**

Table 4: Nodos expandidos para Hanoi 4-12 BFS no Pruning Facil

BFS No Pruning Hanoi 4-12 (Medio)

Profundidad	Promedio de nodos expandidos
1	6.2
2	21.2
3	74.8
4	227.6
5	1308.4
Tiempo Promedio Total (ms)	90

Table 5: Nodos expandidos para Hanoi 4-12 BFS no Pruning Medio

BFS no Pruning Hanoi 4-12 (Dificil)

Profundidad	Promedio de nodos expandidos
1	6.8
2	38.8
3	136.8
4	580.6
5	1777.4
Tiempo Promedio Total (ms)	120.0

Table 6: Nodos expandidos para Hanoi 4-12 BFS no Pruning Dificil

BFS No Pruning Hanoi 4-12 (Muy Difícil)

Profundidad	Promedio de nodos expandidos
1	7.0
2	42.0
3	252.0
4	1512.0
5	9072.0
6	54432.0
7	326592.0
8	1959552.0
9	1.175e7
10	7.0546e7
Tiempo Promedio Total (ms)	420001.0

Table 7: Nodos expandidos para Hanoi 4-12 BFS no Pruning Muy Dificil

BFS Pruning Hanoi 4-12

Profundidad	Promedio de nodos expandidos
1	6.4
2	10.8
3	9.0
Tiempo Promedio Total (ms)	**24.2**

Table 8: Nodos expandidos para Hanoi 4-12 BFS Pruning Facil

BFS Pruning Hanoi 4-12 (Medio)

Profundidad	Promedio de nodos expandidos
1	6.2
2	9.8
3	12.4
4	11.0
5	19.0
Tiempo Promedio Total (ms)	25.4

Table 9: Nodos expandidos para Hanoi 4-12 BFS Pruning Medio

razón por la que logra acabar el caso muy difícil.

Esto se puede deber al dominio: al haber 4 astas y la imposición sobre los discos, esto impone varios estados repetidos.

4.3 Solucion mediante A* 4 astas 12 discos

A* utiliza como heuristica el maximo de 2 PDB. Debido a que la representacion usada son tuplas binarias, el mapeo queda fuera de la discusión puesto que no hay manera de mapear los estados sin cambiar la semántica de las transiciones. Por lo tanto, se seleccionaron 2 proyecciones las cuales representan tener una menor cantidad de discos.

Al usar poda parcial de duplicados, los resultados se mantienen, llegando a resolver problemas muy dificiles en un tiempo promedio de 6876.8ms, 100 veces mas rapido que BFS pruning, expandiendo un total de 144043 en promedio.

La mejora es lo suficientemente significativa, como para terminar algunos tests nivel imposible (20000) con un tiempo promedio de 271933.6ms expandiendo 5321372.4 nodos en promedio.

BFS Pruning Hanoi 4-12 (Dificil)

Profundidad	Promedio de nodos expandidos
1	6.8
2	18.6
3	26.2
4	31.4
5	31.4
Tiempo Promedio Total (ms)	30.4

Table 10: Nodos expandidos para Hanoi 4-12 BFS Pruning Dificil

BFS Pruning Hanoi 4-12 (Muy Difícil)

Profundidad	Promedio de nodos expandidos
1	7.0
2	22.0
3	53.4
4	99.6
5	211.6
6	327.0
7	573.8
8	905.4
9	1217.6
10	1956.8
11	2823.8
12	3502.8
13	4904.8
14	7089.6
15	9511.0
16	11185.8
17	13873.2
18	18672.6
19	24705.6
20	29610.0
21	32389.2
22	36650.4
23	39554.8
24	51240.2
25	61503.0
26	67124.2
27	69505.0
28	74743.4
29	70421.6
30	59187.2
31	71836.4
32	51172.4
33	52833.2
34	52197.8
35	54368.0
36	61900.2
37	73172.0
38	87633.8
Tiempo Promedio Total (ms)	80986.2

Table 11: Nodos expandidos para Hanoi 4-12 BFS Pruning Muy Dificil

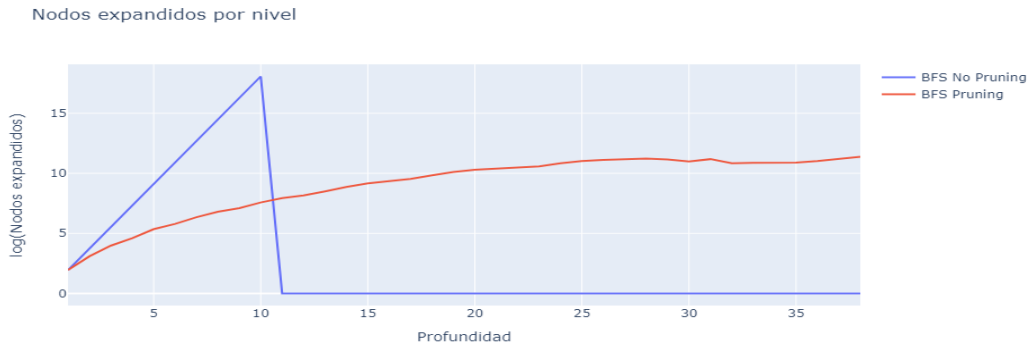

Figure 6: Expansion de nodos para Hanoi 4-12 BFS Muy Dificil

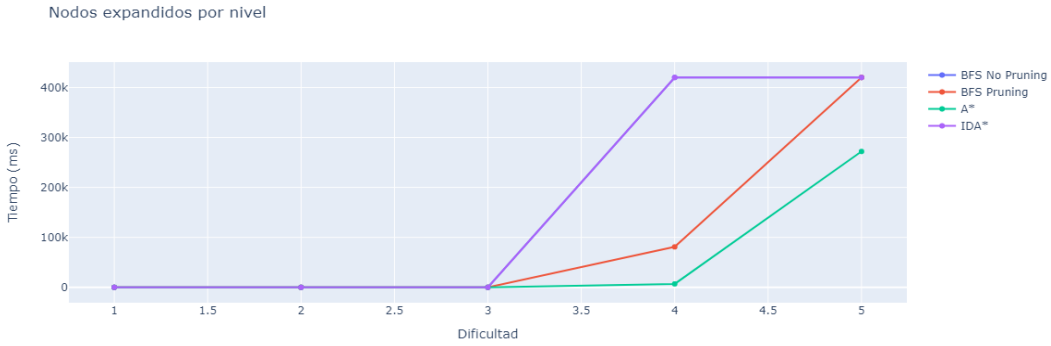

Figure 7: Grafo comparativo tiempos de corrida

4.4 Solucion mediante IDA* 4 astas 12 discos

A diferencia de A* y BFS con pruning, IDA* al trabajar unicamente con heuristica y iterated deepening no aprovecha la poda parcial de duplicados y posee expansion exponencial en memoria, resolviendo solo el problema de nivel medio con un tiempo promedio de 0ms y 13.6 nodos expandidos.

4.5 Solución mediante BFS (no) prunning Hanoi 4 astas 14 discos

La Figura (8) posee el mismo comportamiento que la de la Figura (6). Este comportamiento es esperado, puesto que añadir dos discos adicionales no afecta el comportamiento exponencial del BFS no pruning ni afecta las transiciones realizadas. Por lo tanto remitimos el análisis a la sección Hanoi de 12 discos.

4.6 Solución mediante (ID)A* Hanoi 4 astas 14 discos

Al aumentar la cantidad de discos, crece exponencialmente el espacio de búsqueda, y por lo tanto los casos de pruebas imposibles que apenas corrían, han parado de correr. Por lo tanto, el análisis se realizara basado en los casos de prueba Muy difíciles.

En la Figura (9) vemos un comportamiento similar al de la Figura (7), lo cual refuerza la hipótesis de que añadir mas discos no modifica el comportamiento de los algoritmos bajo la misma abstracción. En otras palabras: proyectar discos mantiene la cantidad de nodos expandidos y tiempo de ejecución.

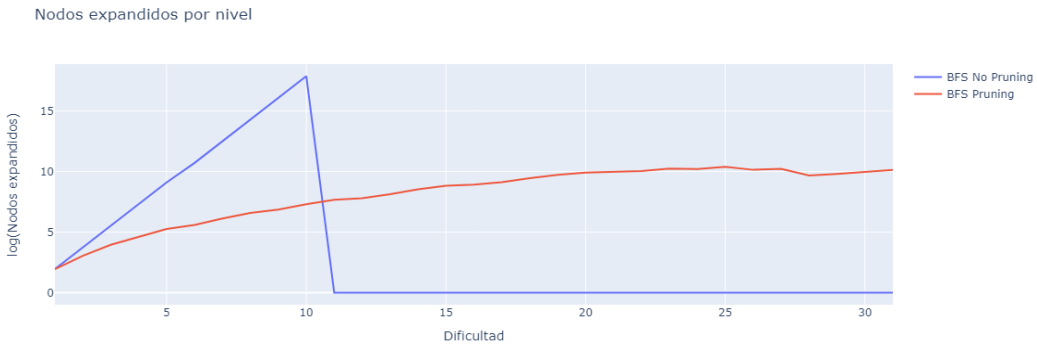

Figure 8: Comparacion de nodos expandidos por BFS (no) pruning Hanoi 14

BFS No Pruning Hanoi 4-14 (Dificil)

Profundidad	Promedio de nodos expandidos
1	7.0
2	42.0
3	251.5
4	1505.666
5	9006.5
6	45114.166
7	270301.333
8	1619519.666
9	9703367.833
10	5.813e7
Tiempo Promedio Total (ms)	350054.666

Table 12: Nodos expandidos por BFS no pruning Hanoi 14 discos

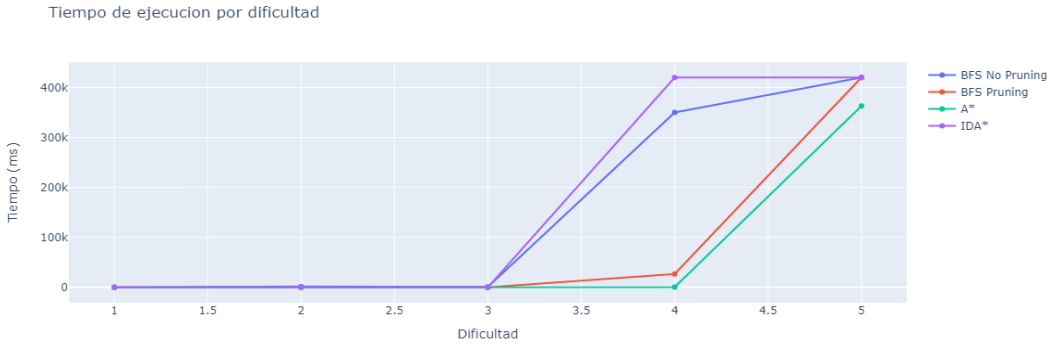

Figure 9: Comparacion de tiempos para Hanoi 14 (Muy dificil)

BFS Pruning Hanoi 4-14 (Muy Dificil)

Profundidad	Promedio de nodos expandidos
1	7.0
2	21.0
3	52.166666666666664
4	99.5
5	193.16666666666666
6	266.8333333333333
7	457.5
8	720.5
9	953.5
10	1478.1666666666667
11	2131.5
12	2432.1666666666665
13	3403.0
14	5102.833333333333
15	6780.0
16	7422.5
17	9227.0
18	12697.333333333334
19	16953.0
20	20204.833333333332
21	22074.5
22	23187.0
23	28268.666666666668
24	27021.0
25	32435.666666666668
26	25448.333333333332
27	27744.166666666668
28	15865.333333333334
29	17997.5
30	21445.666666666668
31	25093.5
Tiempo Promedio Total (ms)	26678.666

Table 13: Nodos expandidos por BFS pruning Hanoi 14 discos Muy Dificil

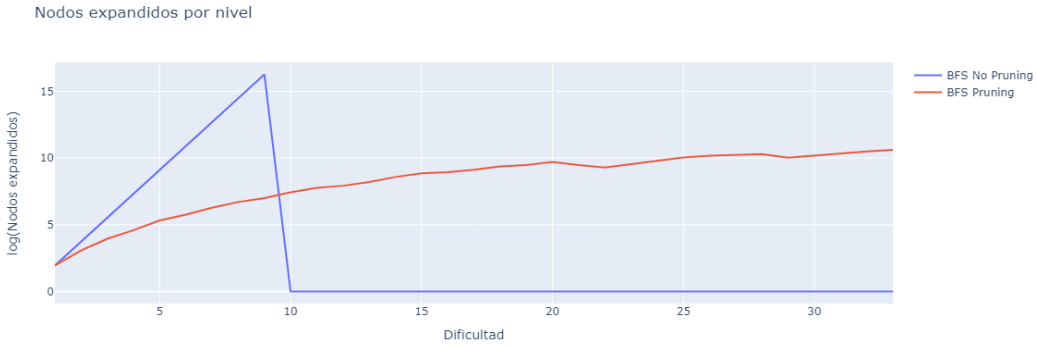

Figure 10: Comparación de nodos expandidos por BFS (no) pruning Hanoi 18

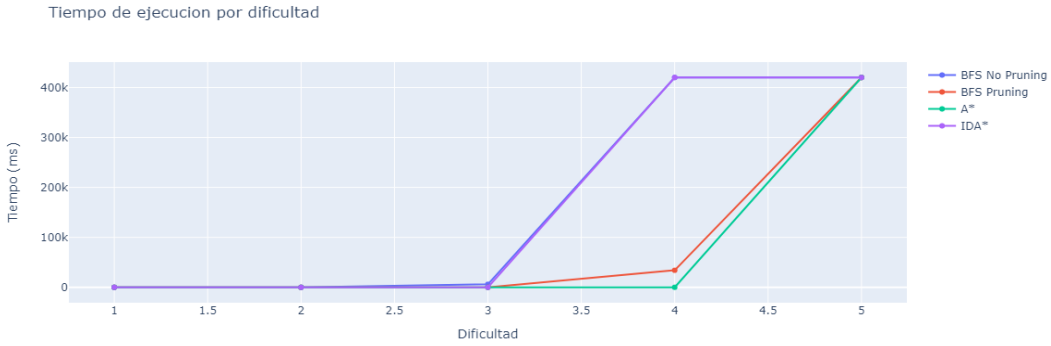

Figure 11: Comparacion de tiempos para Hanoi 18 (Muy dificil)

4.7 Solución mediante BFS (no) prunning Hanoi 4 astas 18 discos

La Figura (10) relfeja el mismo comportamiento que Las Figuras (8) y (6). Lo cual, de nuevo, confirma la hipótesis de que añadir mas discos no modifica el comportamiento de los algoritmos bajo la misma abstracción. En otras palabras: proyectar discos mantiene la cantidad de nodos expandidos y tiempo de ejecución. Y por lo tanto, referimos a Hanoi 12 para ver el analisis de los datos.

4.8 Solución mediante (ID)A* Hanoi 4 astas 18 discos

Finalmente, para Hanoi de 18 discos, vemos el mismo resultado que para los 14 y 12 discos, con la única diferencia que como el espacio de búsqueda ha sido expandido, las soluciones que no implementan poda de duplicados acaban divergiendo de manera aún mas rápida.

5 Topspin

El Topspin es un juego en donde se tiene una serie de esferas numeradas conectadas en un carril, el cual posee un mecanismo que permite intercambiar 4 esferas adyacentes. El objetivo del juego, como el del 15 Puzzle, es acabar organizando las esferas en un orden ascendente.

Para este algoritmo se utilizaron las benchmarks presentadas en la carpeta benchmarks, apartado topspin. Adicionalmente, todos los algoritmos corren con la misma cantidad de muestras: 10.

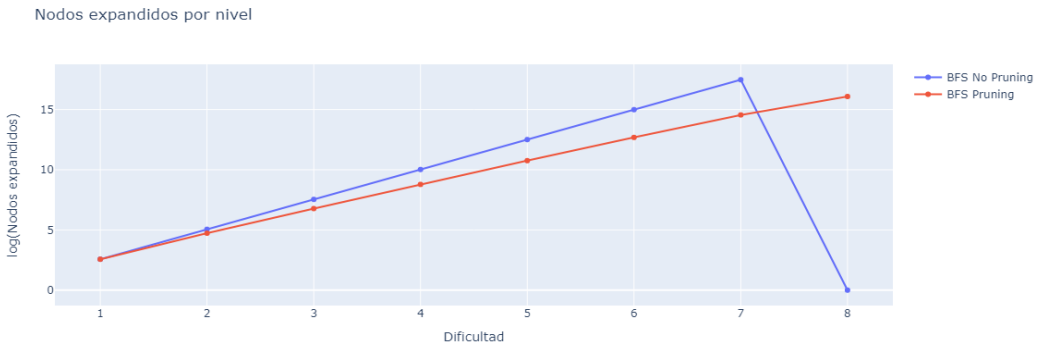

Figure 12: Comparacion de nodos expandidos por BFS (no) pruning TopSpin 12

5.1 Solución mediante BFS (no) prunning para 12 esferas

El problema de las 12 esferas, es en esencia, un problema de permutaciones en donde se aplica una accion a izquierda. Esto quiere decir que es poco común encontrar estados repetidos, ya que necesitariamos aplicar la permutacion inversa para devolvernos a un estado existente. Lo cual se vuelve menos probable a medida que el camino incrementa.

Formalmente, esto es debido a que las unicas dos acciones que tenemos es ciclar: , e intercambiar adyacentes. Por lo tanto, cualquier configuracion se puede expresar como una serie de ciclos: σ_i e intercambios: φ:

$$state_n = \sigma_1\varphi\sigma_2\varphi\sigma_3\varphi\ldots\sigma_n\varphi$$

Para revisitar el estado $n-1$, debemos aplicar φ, lo cual es seleccionar 1 de $n+1$ posibles acciones:

$$state_{n-1} = \sigma_1\varphi\sigma_2\varphi\sigma_3\varphi\ldots\sigma_n\varphi\varphi = \sigma_1\varphi\sigma_2\varphi\sigma_3\varphi\ldots\sigma_n$$

Para revisitar el estado $n-2$, debemos aplicar $\varphi sigma_{-n}$, lo cual es equivalente a seleccionar 1 de $(n+1)^2$ posible acciones:

$$state_{n-2} = \sigma_1\varphi\sigma_2\varphi\sigma_3\varphi\ldots\sigma_n\varphi\varphi sigma_{-n} = \sigma_1\varphi\sigma_2\varphi\sigma_3\varphi\ldots$$

A medida que este camino se vuelve mas y mas largo, menos son las probabilidades de toparnos con un camino antiguo visitado.

Es por esto que la Figura (12) no muestra una mejora significativa al escoger poda parcial de duplicados. La mejora que muestra, es justamente la de no elegir φ en cada iteración, lo cual es un cambio constante constante con respecto a la cantidad de nodos.

5.2 Solución mediante (ID)A* para 12 esferas

La heurística seleccionada para resolver el juego topspin es el máximo de 3 PDBs, donde el primero y el segundo de estos tienen como estrategia hacer un mapeo de una de las mitades de las esferas del juego (inferior o superior) a la ficha inicial de la secuencia, y la tercera unifica cada par de esferas contigüas en la de menor valor. Representaciones gráficas de estas estrategias pueden observarse en la Figura (13) y Figura (14).

BFS No Pruning Topspin 12 (Muy Difícil)

Profundidad	Promedio de nodos expandidos
1	13.0
2	156.0
3	1872.0
4	22464.0
5	269568.0
6	3234816.0
7	3.8817792e7
Tiempo Promedio Total (ms)	406142.4

Table 14: Nodos expandidos por BFS no pruning TopSpin 12 esferas

BFS Pruning Topspin 12 (Muy Difícil)

Profundidad	Promedio de nodos expandidos
1	13.0
2	113.0
3	874.0
4	6506.0
5	46869.0
6	322407.0
7	2085176.0
8	9647596.0
Tiempo Promedio Total (ms)	372746.4

Table 15: Nodos expandidos por BFS pruning TopSpin 12 esferas

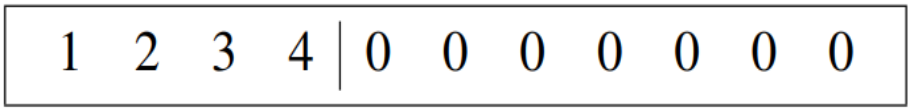

Figure 13: Heurística 1 utilizada para problemas topspin

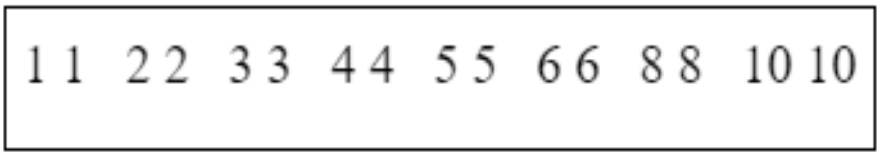

Figure 14: Heurística 2 utilizada para problemas topspin

La Figura (15) Muestra que los tiempos de BFS toman un tiempo considerablemente mayor a los algoritmos informados, enanizandolos. Esto es un indicio de que la heuristica es lo suficientemente estricta como para seleccionar el mejor candidato en casi cada ocasion. Esta hipotesis se confirma al analizar la Figura (16), en la cual observamos que para los problemas de tamaño imposible, A* mejora sobre IDA*, incluso cuando A* realiza poda parcial de duplicados: Al explorar mas a lo profundo con una buena heurística, A* logra converger a la solución mas rápido que expandiendo a lo ancho.

6 Experimentos fallidos

Tanto los Topspin 14/17, El 24 Puzzle y los rubiks poseen PDBs tan grandes que nunca terminan de compilar, o en su defecto, generan representaciones de +8GB en disco, lo cual genera un error a tiempo de ejuccion al tratar de cargar los PDB.

Al tratar de reducir estos PDB, a abstracciones mas pequeñas, como lo es reduciendo el tamaño del particionamiento del 24 Puzzle, obtenemos heuristicas muy laxas que nunca acaban de ejecutar el problema.

7 Conclusiones

Después de haber aplicado BFS no/pruning, A*, e IDA* 4 diferentes juegos, con distintos tipos de heurística, podemos afirmar que no existe un algoritmo definitivo para problemas de exploración.

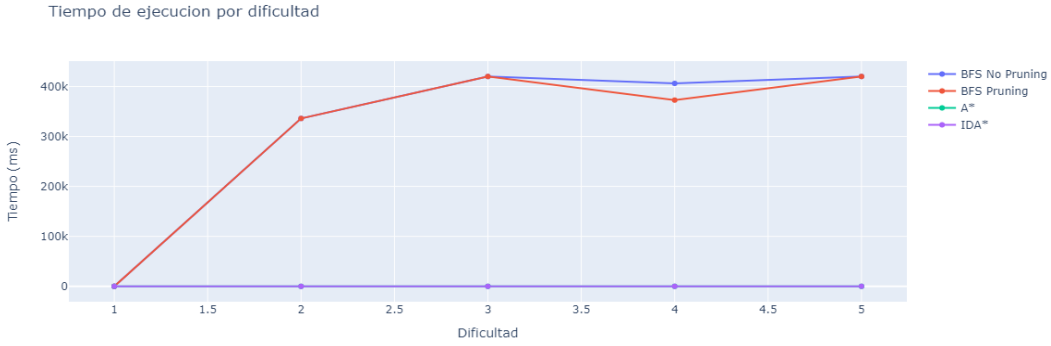

Figure 15: Comparacion de tiempos para TopSpin 12

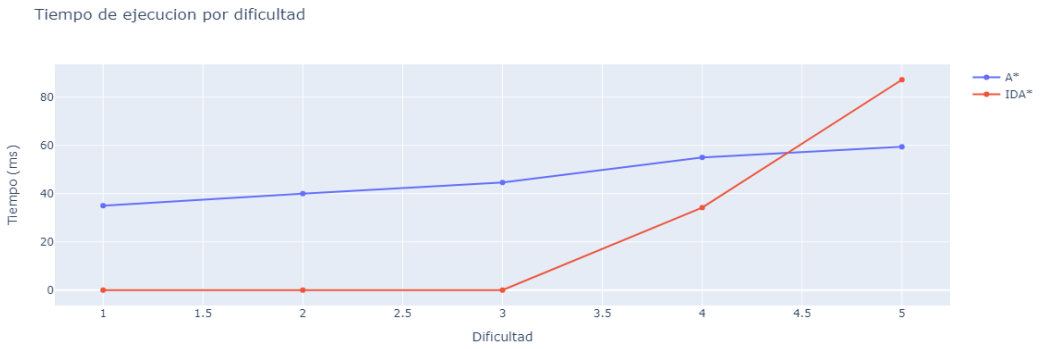

Figure 16: Comparacion de tiempos para TopSpin 12 Informados

La elección entre algoritmos informados y no informados parece estar sesgado hacia los algoritmos informados: aún con una heurística admisible y con que expandan menos nodos, estos expanden en mejor profundidad hacia la dirección del objetivo. Sin embargo, el tradeoff es claramente el uso de PDBs. Dado que los problemas tienen un espacio de busqueda exponencial, el tamaño de los PDB, incluso vistos como versiones simplificadas del problema crecen exponencialmente. Lo cual significa que muchas veces es imposible correr las versiones informadas por limitaciones de hardware.

Entre algoritmos informados, la elección entre IDA* y A* parece ser un poco mas clara. Con una heurística "promedio", IDA* se comporta mejor, puesto que limita la profundidad de exploración, expandiendo de manera mas uniforme y justa los estados. Mientras que con una heurística consistente, A* definitivamente explora nodos de manera mas profunda y por lo tanto, converge mas rápidamente.

Con respecto a los PDB. El tradeoff que se hace es memoria por tiempo: Si el problema no se simplifica lo suficiente, el PDB tiende a sobrepasar las capacidades físicas, y por lo tanto, nunca buildearse. Si el problema se sobre-simplifica, la heurística suele comportarse de manera muy laxa, y el tiempo de ejecución diverge. Elegir un buen PDB requiere una combinacion de: entendimiento profundo del dominio del problema, y ensayo/error.