Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5437 - Inteligencia Artificial I
Ene-Mar 2024
Prof. Carlos Infante

Gabriela Panqueva 18-10761
Daniela Ramírez 16-10940

Informe Proyecto 1

1. Introducción

Este informe se centra en la resolución de un problema común en varios juegos: navegar desde un estado inicial a uno final con el mínimo número de transiciones. Se exploran tres soluciones distintas: el algoritmo BFS, el algoritmo A* y el algoritmo IDA*. El documento detalla las decisiones de diseño y los resultados obtenidos en el primer proyecto del curso CI-5437 Inteligencia Artificial I.

2. Especificaciones de los equipos usados

Para la ejecución de las pruebas se usaron dos equipos, las especificaciones de dichas computadoras son las siguientes:

Computadora 1: 12th Gen Intel(R) Core(TM) i7-12650H 2.30 GHz, 32,0 GB (31,7 GB usable) de RAM y se ejecutaron en una WSL2 de Ubuntu 22.04.3 LTS.

Computadora 2: AMD Ryzen 7 5800H 3.20 GHz, 40,0 GB (39,4 GB usable) de RAM y se ejecutaron en una WSL2 de Ubuntu 22.04.3 LTS.

3. Representación

3.1. N-Puzzle

Para representar los estados se definió un dominio tile de tamaño n+1 de acuerdo a cada puzzle, este define los diferentes valores que puede tomar una pieza. Los tiles pueden tomar valores entre el rango [1,n), siendo n la cantidad total de casillas numeradas en el puzzle, así como el valor de b para representar la casilla en blanco.

Posteriormente, se definieron los movimientos de la casilla blanco b hacia la izquierda, derecha, arriba o abajo, según el caso. El conjunto de los estados objetivos sólo posee un estado, y es aquel donde en la posición 0 del tablero se encuentra la casilla en blanco, es decir, el valor de b y en las n posiciones siguientes los n números de forma ascendente.

3.2. Cubo de Rubik

Los estados están representados por un dominio denominado colour, el cual puede poseer 6 valores: RED, GREEN, BLUE, YELLOW, ORANGE y WHITE, los colores de un cubo de rubik. El estado es el conjunto de las 48 piezas movibles del cubo.

Los nombres de las 6 caras del cubo son: Upper, Down, Left, Right, Front, Back. Ahora bien, en las piezas movibles de un cubo de rubik existen los lados y las esquinas. Las esquinas tienen tres colores mientras que los lados tienen dos. Así, el valor de cada variable es una combinación de 2 o 3 letras que representan la posición actual de esa casilla y la posición (o las dos posiciones) adyacentes a la misma.

Así pues, las ocho casillas corresponden a las piezas de lado o esquina de una de las caras del cubo hasta completar las seis caras, 48 piezas. Los movimientos de cada pieza será la rotación de una de las caras en un giro a cada una de las dos caras adyacentes. El estado objetivo será aquel cuyos lados y esquinas de cada cara sean del mismo color que la ficha central de la misma.

3.3. Torres de Hanoi

Para representar los estados se usaron los dominios de 48, 56 y 72 variables. Cada variable es un binario, es decir, puede tomar el valor 0 o 1.

Supongamos que cada variable se encuentra en la posición v_{ij}, donde i corresponde al número de un disco y j corresponde a la asta en donde se encuentra el disco. Por ejemplo, si $v_{ij} = 1$, entonces el disco i se encuentra en la asta j. Si $v_{ij} = 0$, entonces el disco i no se encuentra en la asta j.

Cada disco i se puede mover desde una asta hacia otra distinta y se definió como estado objetivo aquel en donde todos los discos están en la primera asta.

3.4. Topspin

Los estados están representados por 12, 14 y 17 variables. Cada variable puede tomar valores en un rango de [0,n), donde n es la cantidad de variables del topspin.

Los movimientos son todas las posibles rotaciones de cada segmento compuesto por 4 variables consecutivas. Estas rotaciones incluyen los bordes del vector que se completan tomando el inicio del mismo.

Para cada variación del topspin, se definieron n estados objetivos. Los estados objetivos son las rotaciones ordenadas de las variables de estado.

4. Abstracciones utilizadas

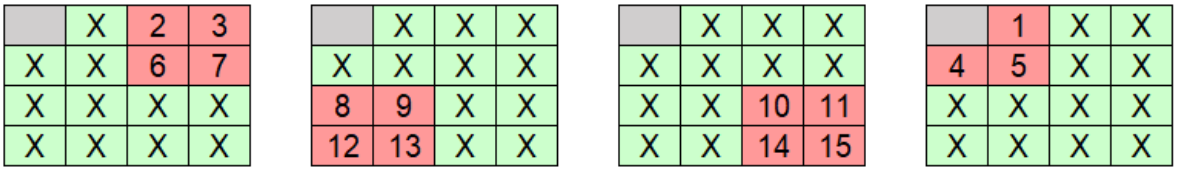
4.1. N-Puzzle

Se crearon abstracciones por el método de proyección. En cada una, se mappeó el problema para que quedaran un número determinado de tiles, más el blanco, de forma disjunta entre las abstracciones. De esta forma, cuando se juntan todas las abstracciones, las N casillas están nuevamente llenas y ninguna se sobrepone a la otra.

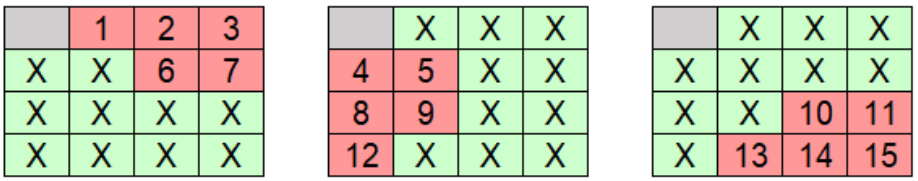
4.1.1. 15-Puzzle

Para el 15-Puzzle, se decidió realizar tres PDBs aditivas:

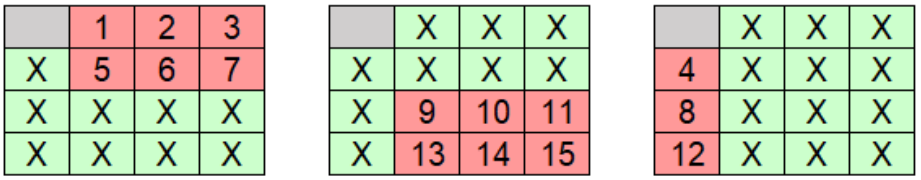
- 4-4-4-3: Con 4 abstracciones, una con tres casillas y las demás abstracciones con cuatro casillas. Así como se muestra a continuación.



- 5-5-5: Con 3 abstracciones, cada una considerando 5 casillas. A continuación, se muestra una figura explicativa de dichas abstracciones.



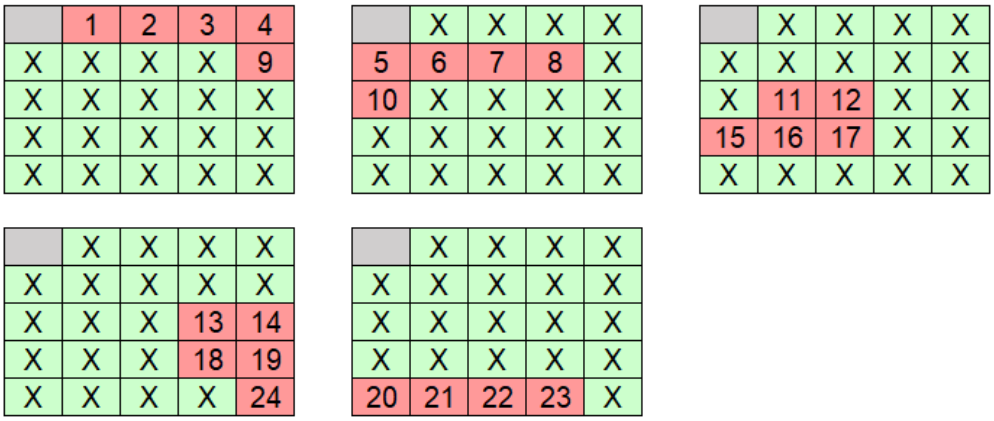
- 6-6-3: Con 3 abstracciones, dos con 6 casillas y una con 3. Como se muestra en las figuras a continuación.



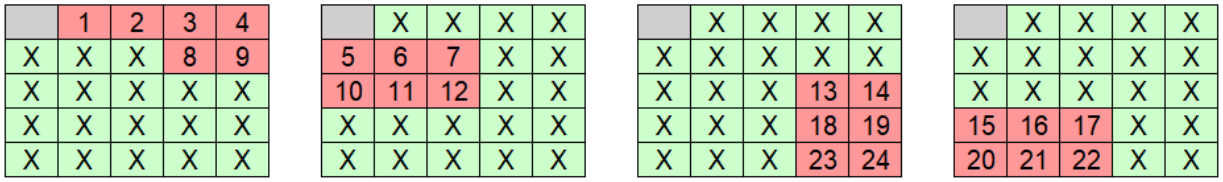
4.1.2. 24-Puzzle

Para el 24-Puzzle, se decidió realizar dos PDBs aditivas:

- 5-5-5-5-4: Este caso son 5 abstracciones, una con 4 casillas y las demás con 5 casillas. A continuación, se muestran las figuras representativas de dicha abstracción.



- 6-6-6-6: Con 4 abstracciones, cada una de ellas considerando 6 casillas. Así como se muestra a continuación.



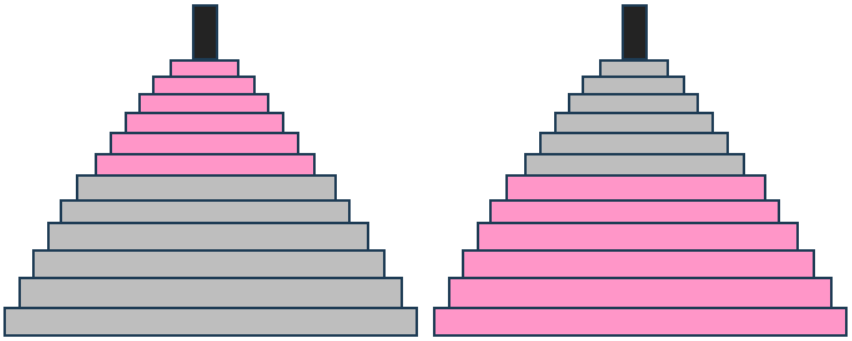
4.2. Cubo de Rubik

Se generó una PDB donde se realizó una abstracción para mantener las piezas de las esquinas y dos abstracciones para los lados.

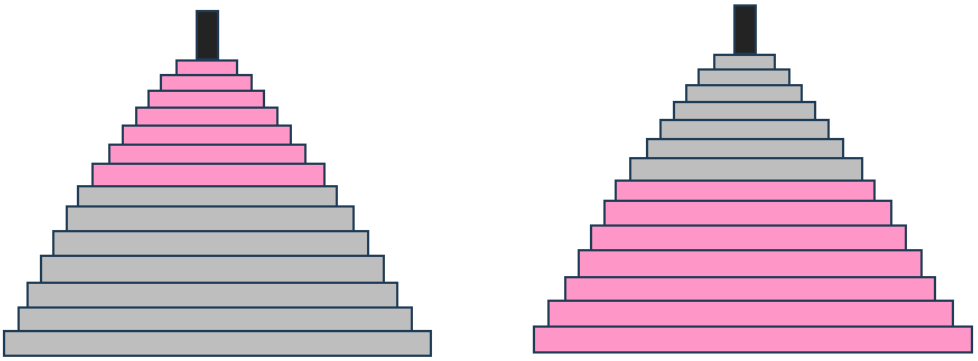
4.3. Torres de Hanoi

Para todos los casos de las torres de Hanoi se crearon dos abstracciones. Una de estas proyecta la primera mitad de discos y la otra proyecta la segunda mitad de discos.

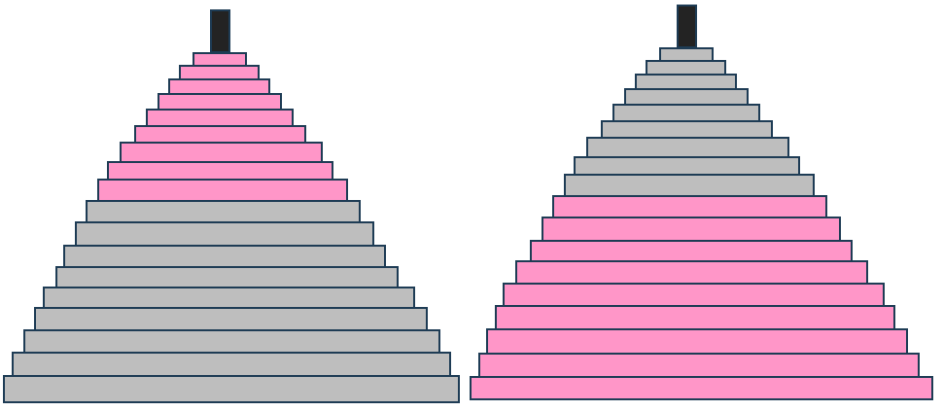
Para el caso 4-12:



Para el caso 4-14:



Para el caso 4-18:



4.4. Topspin

Se generó una PDB utilizando tres abstracciones distintas por cada modalidad del rompecabezas. Se mappearon diferentes variables de estados para simplificar el problema, dejando solamente estados impares/pares e igualando diferentes grupos de variables a una sola variable de estado.

5. Resultados

5.1. Árbol de búsqueda

En este apartado, se muestran los resultados obtenidos al ejecutar el algoritmo de búsqueda Breadth First Search (BFS) para cada problema durante un tiempo (hasta que la memoria RAM se agotara). En particular, se muestra la cantidad de nodos generados y el factor de ramificación en cada profundidad del árbol.

5.1.1. Not prunning

- **15-Puzzle:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	2.000000
1	2	3.000000
2	6	3.000000
3	18	3.222222
4	58	3.206897
5	186	3.236559
6	602	3.232558
7	1946	3.236382
8	6298	3.235630
9	20378	3.236137
10	65946	3.236011
11	213402	3.236080
12	690586	3.236060

13	2234778	3.236070
14	7231898	3.236067
15	23402906	3.236068

- **24-Puzzle:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	2.000000
1	2	3.000000
2	6	3.000000
3	18	3.333333
4	60	3.233333
5	194	3.463917
6	672	3.312500
7	2226	3.504043
8	7800	3.336667
9	26026	3.515254
10	91488	3.344198
11	305954	3.518176
12	1076400	3.346765
13	3602458	3.518837
14	12676464	3.347763
15	42437794	3.518922

- **Cubo de Rubik:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	18.000000
1	18	18.000000

2	324	18.000000
3	5832	18.000000
4	104976	18.000000

- **Torres de Hanoi 4-12:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	3.000000
1	3	5.000000
2	15	5.000000
3	75	5.240000
4	393	5.366412
5	2109	5.446657
6	11487	5.517106
7	63375	5.566154
8	352755	5.608258
9	1978341	5.641695

Para los casos de torre de Hanoi de 4-14 y 4-18 no se logró compilar el BFS, se esperaron 4 horas y aún así no terminó de compilar el archivo.

- **Topspin 12-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	12.000000
1	12	12.000000
2	144	12.000000
3	1728	12.000000
4	20736	12.000000
5	248832	12.000000

6	2985984	12.000000

- **Topspin 14-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	14.000000
1	14	14.000000
2	196	14.000000
3	2744	14.000000
4	38416	14.000000
5	537824	14.000000

- **Topspin 17-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	17.000000
1	17	17.000000
2	289	17.000000
3	4913	17.000000
4	83521	17.000000
5	1419857	17.000000

5.1.2 Prunning

- **15-Puzzle:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	2.000000
1	2	2.000000
2	4	2.500000

3	10	2.400000
4	24	2.250000
5	54	2.000000
6	108	2.018518
7	218	2.165138
8	472	2.182203
9	1030	2.141747
10	2206	2.118767
11	4674	2.114677
12	9884	2.127074
13	21024	2.136891
14	44926	2.132507
15	95805	2.127739
16	203848	2.126751
17	433534	2.129238
18	923097	2.130866
19	1966996	2.131303
20	4192264	2.130082
21	8929865	2.130860
22	19028296	2.130777
23	40545065	2.131565
24	86424455	2.131490

- **24-Puzzle:**

Profundidad	**Nro. de nodos**	**Factor de ramificación**
0	1	2.000000
1	2	2.000000

- **Cubo de Rubik:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	18.000000
1	18	13.500000
2	243	13.333333
3	3240	13.350000
4	43254	13.348314

- **Torres de Hanoi 4-12:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	3.000000
1	3	3.000000
2	9	3.111111
3	28	3.000000
4	84	3.369048
5	283	3.289753
6	931	3.412460
7	3177	3.511489
8	11156	3.485030
9	38879	3.551660
10	138085	3.566991
11	492548	3.591301
12	1768888	3.602649
13	6372683	3.615880

Para los casos de torre de Hanoi de 4-14 y 4-18 no se logró compilar el BFS, se esperaron 4 horas y aún así no terminó de compilar el archivo.

- **Topspin 12-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	12.000000
1	12	8.500000
2	102	7.686275
3	784	7.501276
4	5881	7.478830
5	43983	7.461565

6	328182	7.444058
7	2443006	7.420242
8	18127695	7.399623

- **Topspin 14-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	14.000000
1	14	9.500000
2	133	8.315789
3	1106	7.965642
4	8810	7.905108
5	69644	7.843705
6	546267	7.825293
7	4274699	7.804489

- **Topspin 17-4:**

Profundidad	Nro. de nodos	Factor de ramificación
0	1	17.000000
1	17	11.000000
2	187	9.272727
3	1734	8.651095
4	15001	8.489834
5	127356	8.345951
6	1062907	8.296053

5.2. Algoritmos de búsqueda informada

En este apartado, se muestran los resultados obtenidos al ejecutar los algoritmos A* e IDA* para cada problema durante un tiempo.

- **15-Puzzle:**

Para este caso se escogió el archivo de prueba de 100 instancias proporcionado en el proyecto. Los resultados obtenidos al ejecutar A* y IDA* con las distintas heurísticas implementadas se muestran en la siguiente tabla:

➢ Distancia Manhattan: En el caso de A*, solo logró resolver 17 instancias de las 100 del archivo de prueba hasta que se agotó la memoria del computador. Por ese motivo, se escogieron las instancias 001 y la 016. El archivo con los resultados completos se encuentra en la carpeta del repositorio en la ruta /src/N-Puzzle/15-Puzzle/Results

Instancia	**Distancia**	**Nodos Exp. (A*)**	**Tiempo A***	**Nodos Exp. (IDA*)**	**Tiempo IDA***
001	53	3417668	0.330715 ms	4450114	0.253045
016	62	134358062	18.2548 ms	122029518	6.8722

➢ PDBs aditivos:

A continuación, se muestran los resultados de dos instancias (la primera y la última generada) de las 100. El archivo con los resultados completos se encuentra en la carpeta del repositorio en la ruta /src/N-Puzzle/15-Puzzle/Results

Caso 4-4-4-3:

Instancia	**Distancia**	**Nodos Exp. (A*)**	**Tiempo A***	**Nodos Exp. (IDA*)**	**Tiempo IDA***
001	53	267648	0.057062 ms	300265	0.267621
086	52	15995776	4.21359 ms	26327376	13.5283

Caso 5-5-5:

Instancia	Distancia	Nodos Exp. (A*)	Tiempo A*	Nodos Exp. (IDA*)	Tiempo IDA*
001	53	322202	0.060648 ms	366522	0.074651
099	54	10359882	2.69461 ms	15085422	2.5476

Caso 6-6-3:

Instancia	Distancia	Nodos Exp. (A*)	Tiempo A*	Nodos Exp. (IDA*)	Tiempo IDA*
001	53	299352	0.061366 ms	313826	0.058466
099	54	2829086	0.739992 ms	3774422	0.93486

- **24-Puzzle:**

Para el 24-Puzzle no se lograron compilar los PDBs y por lo tanto, no se pudo implementar los algoritmos IDA* y A* a este puzzle debido a que se agota la memoria del computador, se intentó en ambas computadoras.

- **Torres de Hanoi**

Para las Torres de Hanoi no se lograron compilar los PDBs y por lo tanto, no se pudo implementar los algoritmos IDA* y A* a este puzzle debido a que se agota la memoria del computador, se intentó en ambas computadoras.

- **Topspin**

En las siguientes tablas se muestran los resultados obtenidos al ejecutar A* y IDA* con PDBs aditivos para el problema Top Spin, utilizando los casos de prueba del benchmark proporcionado.

Los resultados mostrados corresponden solo a dos instancias (la segunda y la penúltima). El archivo con los resultados completos se encuentra en la carpeta del

repositorio en la ruta /src/TopSpin/TopSpinN-4/Results, donde N puede ser 12, 14 o 17.

➢ 12-4

Dificultad	Instancia	Distancia	Nodos Exp. (A*)	Tiempo A*	Nodos Exp. (IDA*)	Tiempo IDA*
Fácil	002	2	6	2.2e-05 ms	2	8e-06
Fácil	098	9	24	3.5e-05 ms	15	2.3e-05
Medio	002	9	36	8.8e-05 ms	15	3.5e-05
Medio	098	8	20	5.3e-05 ms	10	1e-05
Difícil	02	9	12	3.2e-05 ms	56	0.000123
Difícil	09	9	34	5.1e-05 ms	32	5e-05
Muy difícil	02	9	124	0.000237 ms	247	0.005252
Muy difícil	09	9	128	0.000154 ms	167	0.00256

➢ 14-4

Dificultad	Instancia	Distancia	Nodos Exp. (A*)	Tiempo A*	Nodos Exp. (IDA*)	Tiempo IDA*
Fácil	002	5	12	4.3e-05 ms	5	1.2e-05
Fácil	098	10	12312	0.007618 ms	4199	0.005747
Medio	002	12	162744	0.114637 ms	14629	0.022004
Medio	098	11	51566	0.028229	3179	0.006858

				ms		
Difícil	02	11	58030	0.035955 ms	3695	0.005801
Difícil	09	11	809392	0.540314 ms	7494	0.010039

➢ 17-4: Para el algoritmo A*, en los casos de prueba de nivel medio solo logró resolver una instancia y en los casos de prueba de nivel difícil no logró resolverlos ya que se agota la memoria de la computadora.

Dificultad	**Instancia**	**Distancia**	**Nodos Exp. (A*)**	**Tiempo A***	**Nodos Exp. (IDA*)**	**Tiempo IDA***
Fácil	002	3	8	4.5e-05 ms	3	8e-06
Fácil	082	15	168113574	151.03 ms	2142795	3.92854
Medio	002	16	-	-	21114173	39.5016
Medio	098	15	-	-	4316680	8.1935
Difícil	02	15	-	-	3862256	7.19516
Difícil	09	15	-	-	10617219	19.6867

- **Cubo de rubik**

Para el caso del Cubo de Rubik 3x3x3 se generaron varios casos de pruebas de profundidad 5, 10 y 15. Los resultados obtenidos se muestran en la siguiente tabla:

Profundidad	**Distancia**	**Nodos Exp. (A*)**	**Tiempo A***	**Nodos Exp. (IDA*)**	**Tiempo IDA***
5	4	10	0.000139 ms	4	0.000154
5	3	8	5.1e-05 ms	3	7.4e-05
5	4	10	9.2e-05 ms	4	9.3e-05
5	3	8	0.000124	3	0.000112

			ms		
5	5	12	9.4e-05 ms	5	9.1e-05
5	5	12	0.000455 ms	5	9.8e-05
5	3	8	4.2e-05 ms	3	9.7e-05
5	5	12	9.2e-05 ms	7	0.000252
5	3	8	4.7e-05 ms	3	6.3e-05
5	5	12	8.7e-05 ms	5	0.000119
10	8	28	0.00021 ms	19	0.001192
10	8	824	0.002248 ms	210	0.005751
10	8	152	0.000617 ms	34	0.001529
10	10	7310	0.028374 ms	2038	0.049037
10	4	10	5e-05 ms	4	5.9e-05
10	9	600	0.002347 ms	167	0.003707
10	7	3960	0.005376 ms	201	0.003341
10	7	20	9.4e-05 ms	7	0.000103
10	7	24	0.000116 ms	8	0.000219
10	8	42	0.000201 ms	19	0.000416
15	13	21086198	67.1442 ms	16253639	110.379
15	10	7768	0.024665 ms	3843	0.027168
15	12	2090680	7.63437 ms	102258	0.703413
15	13	18355778	66.0518 ms	3020818	20.9548
15	13	28170814	97.8316 ms	2608060	18.0668

15	12	418830	1.48831 ms	348991	2.35595
15	12	2345558	8.58899 ms	607376	4.17024
15	11	44912	0.130834 ms	139902	0.953161
15	11	44756	0.175261 ms	26904	0.194726
15	14	-	-	27505860	187.647

5.3. Análisis de resultados

En cuanto al BFS, al observar las tablas, se puede observar como la eliminación de duplicados reduce significativamente el número de estados generados en cada profundidad. Por ejemplo, para el caso de 15-puzzle sin pruning, el número de estados generados con profundidad 15 es de 23402906, mientras que con pruning es de 95805.

En el caso del Top-Spin y el Cubo de Rubik 3x3x3, el factor de ramificación, sin prunning, permanece constante en todos los niveles. Esto ocurre porque estos juegos no presentan estados inválidos que disminuyan la cantidad de movimientos posibles, a diferencia de lo que podría suceder en el 15-puzzle o las Torres de Hanoi.

Cuando se examina la cantidad de estados producidos a diferentes profundidades en varios puzzles, con o sin poda, se nota que tanto el Top-Spin como el Cubo de Rubik muestran un incremento rápido en la cantidad de estados generados (alrededor de 105 estados a una profundidad de 4). Por otro lado, las Torres de Hanoi, el 15-puzzle y el 24-puzzle muestran un crecimiento más gradual (máximo 103 estados a profundidad 4). Esto se atribuye a que los primeros puzzles permiten más movimientos posibles que los últimos, lo que resulta en un mayor factor de ramificación.

En cuanto a los algoritmos de búsqueda informada, para el 15 puzzle, se puede observar que para el caso con la heurística de distancia Manhattan implementada el algoritmo IDA* tiene mejores tiempos de ejecución que el A*. Para el caso de los PDBs, el caso que obtuvo mejores tiempos con IDA* fue el 6-6-3.

Para todos los casos de TopSpin, igualmente el algoritmo que tuvo mejores tiempos fue el IDA*. Sin embargo, en el caso de 12-4, para las pruebas fácil y medio el algoritmo A* expandió menos nodos que con el algoritmo IDA*, y en los casos difícil y muy difícil el IDA* expandió más nodos en comparación de A*.

En el caso de 14-4 y 17-4, aunque este último solo se pudo resolver los casos de prueba fáciles, se puede observar que con A* se expandieron muchos más nodos que con IDA*.

Ahora, en cuanto al cubo de rubik 3x3x3 los tiempos de ejecución de ambos algoritmos fueron variando, es decir, en algunos casos el A* obtuvo mejor tiempo que el IDA*, pero por muy poco. Sin embargo, a mayor distancia, A* no logra terminar la ejecución dado que se agota la memoria del computador, como es el caso de la prueba de profundidad 15 que sólo solucionó 9 instancias, mientras que el IDA* logró solucionar 10 instancias.

En ciertas situaciones, IDA* puede tener una ventaja debido a la posibilidad de encontrar la solución óptima expandiendo menos nodos que A*. Esto suele ser más común en problemas de menor dificultad, donde A* necesita completar la revisión de todos los nodos con un valor específico antes de avanzar al siguiente.

La representación del problema y la forma en que se almacenan y codifican las PDBs son fundamentales para la eficiencia y efectividad de los algoritmos. Como se ha comentado anteriormente, en varios casos, el tamaño de las PDBs eran tan grandes que requería de mucha memoria RAM, lo que hace que el problema sea inviable en computadoras con recursos limitados.

6. Conclusiones

En el transcurso de este proyecto, se observó el comportamiento de algoritmos de búsqueda informada y no informada.

Para la búsqueda no informada, se empleó el algoritmo de búsqueda en anchura (BFS) para la generación de árboles de búsqueda. Se observó que la implementación de la técnica de poda (pruning) resultó en una mejora significativa en los tiempos de ejecución en comparación con la versión sin poda. Sin embargo, es crucial señalar que la implementación del BFS, en la mayoría de los casos, agotaba la memoria del sistema antes de alcanzar el tiempo estipulado. Esto subraya una limitación importante del BFS en términos de eficiencia de la memoria. En contraste, según la teoría vista, el algoritmo de búsqueda en profundidad iterativa (IDDFS) podría ser una alternativa más eficiente, ya que no presenta el mismo consumo de memoria. Por lo tanto, para futuras implementaciones y para optimizar el rendimiento, se recomienda el uso del algoritmo IDDFS.

En cuanto a la búsqueda informada, se observó el comportamiento de dos algoritmos: A* e IDA*. Ambos algoritmos utilizan heurísticas para guiar la búsqueda y tienen sus propias fortalezas y debilidades.

A* es eficaz en la exploración de nodos más profundamente con una heurística consistente, lo que puede permitirle converger más rápidamente a la solución. Sin embargo, A* puede consumir más memoria ya que mantiene todos los nodos que han sido abiertos pero no explorados. En algunos casos, observamos que A* no arrojó todos los resultados como IDA*, lo que podría deberse a la heurística utilizada o a limitaciones de memoria. En este caso, la segunda opción es la más probable, ya que en algunos casos se pudo observar que el algoritmo IDA* expandió menos nodos, lo cual implica que requirió menos memoria y pudo completar más instancias de los problemas, mientras que A* solía resultar en "*killed*" más rápidamente..

Por otro lado, IDA* limita la profundidad de exploración, lo que puede ser útil para evitar la expansión de demasiados nodos en problemas con un espacio de estados muy grande. Sin embargo, IDA* puede requerir más tiempo ya que repite la búsqueda a profundidades crecientes.

Entonces, la elección entre A* e IDA* depende en gran medida del problema específico que se esté resolviendo y de los recursos disponibles. Ambos algoritmos tienen sus fortalezas y debilidades, y la elección entre uno u otro debe basarse en una cuidadosa consideración de las características del problema y de los recursos disponibles.

Con respecto a las Torres de Hanoi y el 24-Puzzle, poseen PDBs tan grandes que nunca terminan de compilar, o en su defecto, generan representaciones muy grandes para nuestro disco, lo cual trae como consecuencia que no se compilen los códigos de programas que contienen los algoritmos de búsquedas, por esta razón, algunos experimentos no se pudieron realizar. Esto implica, que aunque A* e IDA* son algoritmos que pueden llegar a ser eficientes en las resoluciones de muchos problemas, aún existen algunos que estos no pueden resolver en máquinas comunes.

Además, cabe resaltar que probablemente no se utilizó toda la memoria RAM disponible por las máquinas, ya que no se modificó la cantidad de memoria RAM permitida para los procesos ejecutados (WSL y algoritmos) por defecto. Si se hubiera modificado, quizás, A* hubiera podido ejecutar más instancias de los problemas, aunque esto no quita el hecho de que IDA* utiliza menos recursos que A*.

Finalmente, aunque hubo experimentos que no se pudieron realizar debido a su complejidad, la mayoría de experimentos fueron realizados de manera exitosa y nos demuestran que dependiendo de los recursos y del problema, se utiliza uno u otro, por ejemplo, si hay restricciones de memoria, es preferible usar IDA*, de lo contrario se podría utilizar A* ya que por lo general es más óptimo.

.