UNIVERSIDAD SIMÓN BOLÍVAR

CI5437

Informe Proyecto #1

Estudiantes:

Pietro Iaia 15-10718

Diego Peña 15-11095

Sartenejas, Junio 2021

1- Metodología

Se utilizaron los algoritmos A* (Con eliminación retardada de duplicados) e IDA* (Con poda parcial) para ejecutar instancias de los siguientes problemas.

- 15 puzzle y 24 puzzle
- Torres de Hanoi de 4 astas con 12, 14 y 18 discos
- Top-spin: 12-4, 14-4 y 17-4
- Cubo rubik 3x3x3

A continuación, para cada problema se enumeran las heurísticas utilizadas para cada uno. Las PDBs son tomadas o inspiradas en el paper de felner at al, 2004, en el caso de los n puzzles y las torres de Hanoi

- 15 Puzzle:
    - Suma de la distancia Manhattan de cada ficha a la posición correcta
    - PDBs aditivas: Cada PDB contenía una solo parte de las fichas, de resto se dejaron espacios en blanco. El valor de la heurística era la suma de los valores de las PDBs utilizadas por la misma. Cabe destacar que las PDBs usadas en una misma heurística debían tener conjuntos disjuntos de fichas para que la heurística aditiva preservara su admisibilidad
        - PDB A: 3 PDBs de 5 fichas cada una
        - PDB B: 2 PDBs de 6 fichas y una de tres
        - PDB C: 1 PDB de 7 fichas, 1 PDB de 6 fichas y otra de 2 fichas
        - La partición de PDBs sugerida por Felner de una 8 y otra de 7 no se pudo implementar ya que requería los recursos disponibles de memoria no fueron suficientes (Ver sección de recursos) para compilar la PDB de 8 fichas.

- 24 Puzzle:
    - PDBs aditivas:
        - 4 PDBs de 5 fichas y 1 de 4 fichas
        - Las PDBs por el paper de Felner, et al. De 4 PDBs de 6 fichas no compiló con los recursos disponibles. Cabe destacar que el paper de

Felner et al, utilizó sparse PDBs para este caso. PSVN (Ver implementación)

- Top Spin 12-4:
    - Max de un conjunto de PDBs:
        - PDB A: 4 PDBs de 3 fichas
        - PDB B: 3 PDBs de 4 fichas
        - PDB C: 2 PDBs de 6 fichas

- Top Spin 14-4:
    - Max de un conjunto de PDBs:
        - PDB A: 4 PDBs de 4 fichas (Se repiten dos fichas)
        - PDB B: 3 PDBs de 5 fichas (Se repite una ficha)
        - PDB C: 2 PDBs de 7 fichas

- Top Spin 17-4:
    - Max de un conjunto de PDBs:
        - 3 PDBs de 6 fichas (Se repite una ficha)
        - Se intentó compilar una PDB de 9 fichas y otra de 8, pero no se pudo con los recursos disponibles

- Torres de Hanoi 12 discos:
    - Max de un conjunto de PDBs:
        - 1 PDB que elimina el disco más pequeño (11v1)
        - 1 PDB que elimina los dos discos más pequeños (10v2)
        - 1 PDB que elimina los tres discos más pequeños (9v3)

- Torres de Hanoi 14 discos:
    - Max de un conjunto de PDBs:
        - 1 PDB que elimina los dos discos más pequeños (12v2)
        - 1 PDB que elimina los tres discos más pequeños (11v3)
        - 1 PDB que elimina los cuatro discos más pequeños (10v4)
    - Max de un conjunto de PDBs:
        - 1 PDB que elimina los dos discos más pequeños (12v2)

        - 1 PDB que elimina los tres discos más pequeños (11v3)
        - 1 PDB que elimina los cuatro discos más pequeños (10v4)
        - 1 PDB que elimina los cinco discos más pequeños (9v5)

- Torres de Hanoi 18 discos:
    - Max de un conjunto de PDBs:
        - 1 PDB que elimina los siete discos más pequeños (11v7)
        - 1 PDB que elimina los ocho discos más pequeños (10v8)
        - 1 PDB que elimina los nueve discos más pequeños (9v9)
        - 1 PDB que elimina los diez discos más pequeños (8v10)

- Rubik 3x3x3:

  Para este caso tuvimos que dividir los PDBs indicados en el enunciado en varias PDBs más pequeñas, el por que de esta decisión se explica más adelante en la sección de Rubik.
    - Max de un conjunto de PDBs:
        - 3 PDBs que eliminan todos los cubitos excepto los 8 de las esquinas y eliminan colores del cubo
        - 3 PDBs que eliminan todos los cubitos excepto los 12 de los bordes y eliminan colores del cubo

Los casos de prueba utilizados fueron suministrados por el profesor de la materia. Incluyen las instancias de Korf para el 15 y 24 puzzle.

2- Recursos

- Google Colab: que incluye 13GB de RAM y modelos no revelados de CPU y GPU
- Laptop con Intel I5 Core con 2.30GHz, 8GB de RAM
- Laptop con Intel Core I7 4ta generación con 2.40GHz, 8GB de RAM

3- Implementación:

El código del proyecto está disponible en https://github.com/djpg98/proyecto-1-ci5437. Se utilizó el lenguaje PSVN para crear una representación del modelo de espacio de estados de cada problema, así como la API del mismo. Estas herramientas fueron diseñadas por el profesor Rob Holte de la universidad de Alberta. El archivo en lenguaje psvn es compilado a C y genera un tipo state_t que contiene la información que las funciones de la API necesitan para determinar cosas como los sucesores de un estado, o determinar si un estado es un estado objetivo. PSVN además provee estrategias para generar abstracciones y PDBs que también fue utilizado.

- **DistSummary y DistSummaryNOP:** DistSummary.c es una implementación del algoritmo de Dijkstra suministrada por PSVN que parte de la meta y empieza a buscar desde allí hasta el estado inicial. Fue usado para generar los árboles de búsqueda. DistSummaryNOP.c es una versión modificada para no realizar ningún tipo de eliminación de duplicados.

- **IDA*:** Se implementó de acuerdo a las especificaciones dadas en clase, particularmente la versión que gasta memoria constante en estados. Esta versión no requiere de nodos para llevar el "bookkeeping" del espacio de estados. Para representar este último fue suficiente con el tipo state_t creado por PSVN basado en los archivos .psvn utilizados para cada problema. Adicionalmente se utilizaron los mecanismos de podado de PSVN "Move pruning", con history_len=1, la cual elimina ciclos de longitud 2, o lo que es equivalente, poda de padres.

- **A*:** Este algoritmo y la estructura Node, que representa los nodos, se implementaron de acuerdo a las especificaciones dadas en clase. Para la implementación de la cola de prioridades, se utilizó el archivo "priority_queue.hpp" encontrado en la carpeta con el material de PSVN, compartida por el profesor. Adicionalmente, se utilizaron los mecanismos de podado de PSVN "Move pruning", con history_len=1, la cual elimina ciclos de longitud 2, o lo que es equivalente, poda de padres.
- Distancia Manhattan: Se utilizó computación incremental para calcular la distancia Manhattan. Para ello se debieron guardar arreglos que almacenaban la diferencia

entre dos estados dado el movimiento de una pieza. Par poder implementar esto se requirió información adicional a la que contenía el estado de PSVN, por lo que se creó una estructura que contenía qué pieza estaba en cada posición y la posición donde estaba el blank para complementar cada estado

Scripts complementarios:

- Se utilizaron scripts de python para generar representaciones de PSVN de los problemas de manera automatizada
- Se utilizó un script de python para ejecutar los algoritmos sobre listas completas de casos que estuvieran incluidas en un archivo. Esto adicionalmente escribe los resultados de cada búsqueda (Tiempo, nodos, costo de la solución) en un archivo csv.

4- Resultados

4.1- 15 Puzzle:

Todas las ejecuciones, salvo A* Manhattan se hicieron en el procesador I5. A* Manhattan se hizo en Google Colab. El identificador de cada PDB (A, B, C) puede ser revisado en la sección de metodología para consultar como se distribuyeron las PDBs en esa heurística. El algoritmo se ejecutó sobre los 100 casos de Korf.

Algoritmos	A*				IDA*			
Heurística	Manhattan	pdb A	pdb B	pdb C	Manhattan	pdb A	pdb B	pdb C
Resuelto %	80	97	94	98	100	100	100	100
Min. error	0.15	0.12	0.14	0.14	0.15	0.12	0.14	0.14
Prom. error	0.3	0.26	0.27	0.26	0.3	0.27	0.27	0.26
Max. Error	0.44	0.42	0.47	0.38	0.47	0.42	0.47	0.38

Tabla 4.1: Porcentaje de casos resueltos y errores en estimación inicial

Algoritmos	A*				IDA*			
Heurística	Manhattan	pdb A	pdb B	pdb C	Manhattan	pdb A	pdb B	pdb C
Min. nodos	54900	26745	18539	12124	236694	57269	45138	50453
Prom. nodes	6271113	3246518	2886376	2466324	364979752	59447523	65829312	38334137

Max. nodes	30787788	24439957	21420012	16611563	6720638063	1672875063	891532242	970978402

Tabla 4.2: Nodos explorados (Cuando se llega a la solución)

Algoritmos	A*				IDA*			
Heurística	Manhattan	pdb A	pdb B	pdb C	Manhattan	pdb A	pdb B	pdb C
Min. T (ms)	72	42	28	18	10	11	9	11
Prom. T (ms)	33519.18	11305.71	10104.24	8513.51	17442.01	11607.09	13703.14	8313.31
Max. T (ms)	184843	98223	90033	79917	285921	325604	186679	204989

Tabla 4.3 Tiempo que tardó en llegar a la solución

Se puede apreciar que los mejores resultados los da, en general IDA*, ya que siempre llega a la solución en menor tiempo. La PDB C es la heurística más efectiva. Las instancias en las que A* no encontró la solución fueron porque se acabó la memoria

4.2- 24 Puzzle:

Se muestran los resultados de la ejecución de 10 de los 50 casos de Korf con IDA*. Valor indica el valor que tomó la heurística en el estado inicial, cota la cota máxima alcanzada de f-valor, solución indica el costo de la solución (-1 si no se encontró), nodos los nodos visitados, nodos última iteración lo que indica su nombre y tiempo ms. Las ejecuciones se detenían a los 15 minutos si no había solución

Valor	cota	solución	nodos	nodos últ. iter.	tiempo (ms)
75	89	-1	928452889	1645632006	900000
66	84	-1	481755450	2091997638	900000
69	89	-1	1275205235	1286451425	900000
74	90	-1	-2063015850	332035570	900000
80	96	-1	1177601685	1399817737	900000
79	93	-1	484436244	2006298603	900000
80	94	-1	299672586	-2088348418	900000
84	100	-1	380692794	2134839408	900000
89	103	-1	625807603	1890656799	900000
86	102	-1	923555171	1575205619	900000

Tabla 4.4: Resultados de IDA* sobre 10 instancias de Korf para 24 Puzzle

Como se puede apreciar, todas las ejecuciones fueron terminadas luego de 15 minutos sin alcanzar la solución. Un vistazo al archivo suministrado de soluciones, confirma que la cota nunca alcanzó el valor de la solución en ninguno de los 10 casos. Adicionalmente, se puede observar que en algunos casos, el número de nodos visitados excedió el valor máximo de un entero. El error en el 7 no se transmitió al total porque los nodos de una iteración sólo se sumaban al total cuando terminaba la iteración

4.3- Top spin 12-4:

Todas las instancias fueron ejecutadas en la computadora con I5. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. Todas las combinaciones de algoritmo y heurística llegaron a la solución

Algoritmos	A*			IDA*		
Heurística	pdb A	pdb B	pdbC	pdbA	pdb B	pdbC
5	67	17.8	6.2	617.6	174.6	36.2
10	17333.2	1185.2	272.6	100262.3	13757.2	1229.3
15	70219.5	12963.4	743.1	719085.4	120212.2	7820.7
2000	137349.6	20012.2	1602.5	1530366	238470	12327.4
2000000	130487.7	28085.1	1383	1587558	249810.6	12468.2
Todos	71091.4	12452.74	801.48	787578	124484.9	6776.36

Tabla 4.5: Nodos explorados (Promedio) con cada algoritmo y cada heurística

Algoritmos	A*			IDA*		
Heurística	pdb A	pdb B	pdbC	pdbA	pdb B	pdbC
5	1.1	0	0	0.5	0	0
10	86.6	6.5	0.9	23.3	5	0.1
15	361.9	58.3	2.8	151.1	28.8	1.7
2000	788.9	96.8	7.1	321	57.2	3.2
2000000	742.7	136.7	5.9	336.5	58.1	3.1
Todos	396.24	59.66	3.34	166.48	29.82	1.62

Tabla 4.6 Tiempo promedio (En ms) que se tarda en resolver con cada algoritmo y heurística

	pdb A	pdb B	pdbC
5	0.4	0.3	0.06
10	0.43	0.31	0.2
15	0.59	0.47	0.33
2000	0.56	0.48	0.23
2000000	0.58	0.44	0.3
Todos	0.51	0.4	0.22

Tabla 4.7: Error promedio al estimar la distancia del estado inicial al estado objetivo con cada heurística

Como se puede observar, el algoritmo más efectivo en términos de tiempo fue IDA* con la PDB C, aunque A* recorrió menos nodos que IDA* usando la misma heurística

4.4- Top Spin 14-4:

Todas las instancias fueron ejecutadas en la computadora con I5. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. Todas las combinaciones de algoritmo y heurística llegaron a la solución

Algoritmos	A*			IDA*		
Heurística	pdb A	pdb B	pdbC	pdbA	pdb B	pdbC
5	15.7	6.3	7.9	191.8	51.4	69.8
10	33719.1	5589.6	315.4	414537.5	61921.5	4102.4
15	523978.9	158108.2	7044	8524135	1594069	61085.4
2000	2449263	381776.9	19636.9	42718879	6109502	228974.6
2000000	680295.1	82259	5209.2	9213536	1370988	48120.2
Todos	737454.3	125548	6442.68	12174256	1827306	68470.48

Tabla 4.8: Nodos explorados (Promedio) con cada algoritmo y cada heurística

Algoritmos	A*			IDA*		
	pdb A	pdb B	pdbC	pdbA	pdb B	pdbC
5	0	0	0	0.1	0	0
10	234.1	37.3	1.5	116.9	18.1	1
15	4513.2	1149.7	47.4	2335.3	445.4	21.4
2000	21691.6	2939.3	145.6	11616.4	1756.9	84
2000000	5669.9	583.3	34.5	2633.5	401	16.8
Todos	6421.76	941.2	45.8	3340.44	534.28	24.64

Tabla 4.9 Tiempo promedio (En ms) que se tarda en resolver con cada algoritmo y heurística

	pdb A	pdb B	pdbC
5	0.24	0.08	0.08
10	0.45	0.33	0.15
15	0.52	0.44	0.31
2000	0.53	0.47	0.33
2000000	0.48	0.43	0.28
Todos	0.44	0.35	0.23

Tabla 4.10: Error promedio al estimar la distancia del estado inicial al estado objetivo con cada heurística

Como se puede observar, el algoritmo más efectivo en términos de tiempo fue IDA* con la PDB C, aunque A* recorrió menos nodos que IDA* usando la misma heurística. Cabe destacar, que los números indican que las instancias con d=2000 fueron en promedio más difíciles que las d=2000000

4.5- Top spin 17-4:

Todas las instancias fueron ejecutadas en la computadora con I5, salvo d=2000 y d=2000000 con A*, esos se corrieron en colab. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. No todas las combinaciones de algoritmo y heurística llegaron a la solución, la primera tabla

refleja que porcentaje de casos fueron resueltos. Los casos que A* no resolvió fue porque se le acabó la memoria

Algoritmos	A*	IDA
	pdb A	pdbA
5	100%	100%
10	100%	100%
15	100%	100%
2000	50%	100%
2000000	80%	100%
Todos	86%	100%

Tabla 4.11: Porcentaje de casos resueltos por cada algoritmo en Top spin 17-4

Algoritmos	A*	IDA
	pdb A	pdbA
5	10.1	133.2
10	1527	18151.2
15	656026.1	8627861
2000	4753215	234589520
2000000	4418109	117773571
Todos	1527595	72201847

Tabla 4.12: Nodos (Promedio) explorados por cada algoritmo

Algoritmos	A*	IDA
	pdb A	pdbA
5	0	0
10	10.5	5.2
15	49488	3106.3
2000	78119.6	86336.1
2000000	73083.5	44358.8
Todos	34191.88	26761.28

Tabla 4.13: Tiempo (Promedio) en ms que tardó cada algoritmo en encontrar la solución

	pdb A
5	0.18
10	0.27
15	0.37
2000	0.47
2000000	0.48
Todos	0.33

Tabla 4.14: Error en la estimación de la distancia del estado inicial al objetivo con la heurística utilizada

Como se puede observar, el algoritmo más efectivo en términos de tiempo fue IDA*, aunque A* recorrió menos nodos que IDA*. Cabe destacar, que los números indican que las instancias con d=2000 fueron en promedio más difíciles que las d=2000000

4.6 - Torres de Hanoi con 12 discos:

Todas las instancias fueron ejecutadas en la computadora con I7. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. No todas las combinaciones de algoritmo y heurística llegaron a la solución, la primera tabla refleja que porcentaje de casos fueron resueltos. Los casos que IDA* no resolvió fue porque se le acabó el tiempo de ejecución (15 min).

Algoritmos	A*	IDA
5	100%	100%
10	100%	100%
15	100%	100%
2000	100%	100%
2000000	100%	0%
Todos	100%	80%

Tabla 4.15: Porcentaje de casos resueltos por cada algoritmo en torres de hanoi 12 discos

	Nodos Totales	Nodos Totales	Nodos Explorados	Nodos Explorados
Algoritmos	A*	IDA	A*	IDA*
5	12.4	4.5	3.1	4.5
10	17	6.4	3.9	6.4
15	20.88	9	4.55	9
2000	170	83	29.5	83
2000000	907028.8	-	151203.5	-
Todos	181449.8	-	30248.9	-

Tabla 4.16: Nodos (Promedio) totales y explorados por cada algoritmo en Torres de Hanoi 12 discos

Algoritmos	A*	IDA
5	0	0
10	0	0
15	0	0
2000	0	0
2000000	2839.4	.-
Todos	2839.4	-

Tabla 4.16: Tiempo (Promedio) en ms que tardó cada algoritmo en encontrar la solución en Torres de Hanoi 12 discos

Valor	cota	solución	nodos	nodos últ. iter.	tiempo (ms)
43	49	-1	324128969	278369390	900000
52	59	-1	252494481	296816580	900000
51	56	-1	321302115	305444333	900000
55	59	-1	297262393	346596148	900000
48	53	-1	137252634	518013970	900000
56	60	-1	192266374	460199783	900000
51	55	-1	192975961	442187655	900000
50	56	-1	158021363	467742432	900000

Tabla 4.17: Resultados de IDA* sobre 8 instancias para Torres de Hanoi 12 discos

4.7- Torres de Hanoi con 14 discos:

Todas las instancias fueron ejecutadas en la computadora con I7. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. No todas las combinaciones de algoritmo y heurística llegaron a la solución, la primera tabla refleja que porcentaje de casos fueron resueltos. Los casos que IDA* no resolvió fue porque se le acabó el tiempo de ejecución (15 min) y los casos que A* no resolvió fue porque se le acabó la memoria. No se crearán columnas separadas para los casos que usaron 3 y 4 PDBs ya que los resultados en las tablas son los mismos, el único cambio será reflejado en una tabla aparte.

Algoritmos	A*	IDA*
5	100%	100%
10	100%	100%
15	100%	100%
2000	100%	100%
2000000	10%	0%
Todos	82%	80%

Tabla 4.18: Porcentaje de casos resueltos por cada algoritmo en torres de hanoi 14 discos

	Nodos Totales	Nodos Totales	Nodos Explorados	Nodos Explorados
Algoritmos	A*	IDA*	A*	IDA*
5	17.37	6.12	4	6.12
10	19.11	8.11	4.22	8.11
15	21.1	8.7	4.6	8.7
2000	151.1	73.5	26.4	73.5
2000000	2075527	-	345973	-
Todos	54672.63	-	9114.55	-

Tabla 4.19: Nodos (Promedio) totales y explorados por cada algoritmo en Torres de Hanoi 14 discos

PDBs	3	4
2000000	8950	7717

Tabla 4.20: Tiempo en ms que diferencia el caso corrido en A* con 3 pdbs y 4 pdbs en Torres de Hanoi 14 discos

Valor	cota	solución	nodos	nodos últ. iter.	tiempo (ms)
65	69	-1	447475684	33459730	900000
64	68	-1	241194622	251489206	900000
68	72	-1	313858017	242409797	900000
37	44	-1	396744342	214454374	900000
63	66	-1	132099686	474128636	900000
64	68	-1	212990995	290458828	900000
66	70	-1	334292479	153503469	900000
59	62	-1	97541056	406950514	900000

Tabla 4.21: Resultados de IDA* sobre 8 instancias usando 3 PDBs para Torres de Hanoi 14 discos

4.8- Torres de Hanoi con 18 discos:

Todas las instancias fueron ejecutadas en la computadora con I7. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. No todas las combinaciones de algoritmo y heurística llegaron a la solución, la primera tabla refleja que porcentaje de casos fueron resueltos. Los casos que IDA* no resolvió fue porque se le acabó el tiempo de ejecución (15 min) y los casos que A* no resolvió fue porque se le acabó la memoria.

Algoritmos	A*	IDA*
5	100%	100%
10	100%	100%
15	100%	100%
2000	100%	100%
2000000	0%	0%
Todos	80%	80%

Tabla 4.22: Porcentaje de casos resueltos por cada algoritmo en torres de hanoi 18 discos

	Nodos Totales	Nodos Totales	Nodos Explorados	Nodos Explorados
Algoritmos	A*	IDA*	A*	IDA*
5	14.22	4.11	3.44	4.11
10	13	4.3	3.2	4.3
15	28.3	11.1	5.6	11.1
2000	158.7	76.2	27.7	76.2
2000000	-	-	-	-
Todos	54.64	24.43	10.23	24.43

Tabla 4.23: Nodos (Promedio) totales y explorados por cada algoritmo en Torres de Hanoi 18 discos

4.9- Rubik 3x3x3:

Todas las instancias fueron ejecutadas en Google Colab. Se corrieron las instancias en los archivos suministrados en el repositorio. El número en la primera columna indica el d usado para generar los casos. En teoría, mientras mayor sea puede generar casos más difíciles. No todas las combinaciones de algoritmo y heurística llegaron a la solución, la primera tabla refleja que porcentaje de casos fueron resueltos. Los casos que IDA* no resolvió fue porque se le acabó el tiempo de ejecución (15 min) y los casos que A* no resolvió fue porque se le acabó la memoria.

Algoritmos	A*	IDA*
5	100%	100%
10	0%	100%
15	0%	100%
20	0%	0%
40	0%	0%
80	0%	0%
160	0%	0%
Todos	14.28%	42.85%

Tabla 4.24: Porcentaje de casos resueltos por cada algoritmo en rubik 3x3x3

	Nodos Totales	Nodos Totales	Nodos Explorados	Nodos Explorados
Algoritmos	A*	IDA*	A*	IDA*
5	73	31.55	4.66	31.55
10	-	1235.8	-	961
15	-	14374898.1	-	10295605.2
Todos	73	4957297.34	4.66	3550549.86

Tabla 4.25: Nodos (Promedio) totales y explorados por cada algoritmo rubik 3x3x3 hasta d=015

Algoritmos	A*	IDA*
5	0	0
10	-	2.40
15	-	27950
Todos	0	9638.79

Tabla 4.26: Tiempo (Promedio) en ms que tardó cada algoritmo en encontrar la solución en rubik 3x3x3

5- Árboles de búsqueda:

Debido a que la mayoría de las tablas generadas son muy grandes, solo colocaremos dos pequeñas en el informe y colocaremos un anexo al repositorio donde podrán encontrar el resto de las tablas en la carpeta ‘Results/Seach_Trees’.

El enunciado del proyecto pide que se reporte en las tablas el número de estados a cada profundidad en el árbol de búsqueda, hasta la profundidad máxima que se alcance en 15 minutos de ejecución, pero hubieron casos en los que, por falta de memoria RAM en nuestras máquinas, la ejecución tuvo que pararse hasta la profundidad en la que estaban al momento de llenarse la memoria.

Profundidad	Nodos	Branching factor
0	18	18
1	324	18
2	5832	18
3	104976	18
4	1889568	18

Tabla 5.1: Tabla con el número de estados y factor de ramificación hasta profundidad 4 del cubo Rubik 3x3x3 sin eliminación de duplicados.

Profundidad	Nodos	Branching factor
0	18	18
1	243	13.5
2	3240	13.3333333
3	43239	13.3453704
4	574908	13.2960522

Tabla 5.2: Tabla con el número de estados y factor de ramificación hasta profundidad 4 del cubo Rubik 3x3x3 con eliminación de duplicados.

El resto de las tablas pueden encontrarse aquí: https://github.com/djpg98/proyecto-1-ci5437/tree/main/results/Search_Trees

6- Análisis de resultados

Las tablas de la sección anterior muestran, en términos generales, una clara ventaja de IDA* sobre A* en términos de tiempo y casos solucionados, aunque en las instancias que soluciona A* tiene una ventaja en número de nodos explorados sobre IDA*. Este último hecho era de esperarse, ya que en cada iteración de IDA*, deben volverse a explorar todos los nodos explorados en las iteraciones anteriores (En la última iteración podría no ser necesario volver a visitar algunos, ya que la solución podría ser encontrada antes). Esto,

sin embargo, no se traduce en ventajas para A*: La implementación de IDA* gasta memoria constante en los estados del problema y memoria lineal en algunos valores enteros que requiere para "bookkeeping", por lo que a pesar de que en ocasiones llega a recorrer 10 veces más estados que A* (Caso top-spin 12-4 con PDB C), su gasto de memoria es muy inferior al del otro algoritmo, que debe guardar nodos asociados a cada estado y mantener una cola de prioridades con los nodos abiertos, la cual puede volverse muy grande a profundidades relativamente bajas si el problema tiene un factor de ramificación significativo, como top-spin cuando no tenía poda**.** Todo este espacio adicional requerido causó que en muchos casos la memoria de la computadora estuviera trabajando a casi 100%, lo cual ralentizó la ejecución del proceso, y contribuyó a que los tiempos de corrida de A* fueran, en promedio, peores que los de IDA*

Otros factores también pudieron influir para que se diera este resultado. Particularmente, las operaciones particulares que realiza cada algoritmo. Las operaciones particulares en IDA* (Cómo llevar el valor de la variable history o devolver el estado al valor que tenía antes de una llamada recursiva) son operaciones que trabajan mayoritariamente con memoria estática o pila, las cuales resultan más fáciles y rápidas de acceder y manipular, que las operaciones sobre el heap que debe realizar A* para asignar, liberar y manipular los nodos.

6.1- 15 puzzle:

Se puede observar que el peor desempeño lo tuvo A* con distancia Manhattan, el cual solo logró resolver 80 de los casos planteados por Korf. Como se comentó previamente en la sección de implementación, para poder aprovechar las ventajas de la distancia Manhattan incremental se requería almacenar información adicional sobre la ficha que se encontraba en cada posición y sobre el blank. Si bien en IDA* esto se tradujo en un costo de memoria y tiempo muy bajo, ya que una variable global y un par de operaciones de tiempo constante eran lo único que se necesitaba, en A* fue necesario almacenar esta información adicional para cada variable, lo cual se sumó a un costo de memoria que ya de por sí solo era alto, aparte del overhead en el que se incurre al reservar el espacio para la estructura de datos necesaria en el heap. No se intentaron utilizar otras técnicas para

calcular la distancia Manhattan en A* debido a que, para IDA* mostraron resultados iniciales muy pobres en comparación con el cálculo incremental

Otro elemento a resaltar sobre el 15 puzzle es que con las PDBs A y B, se puede apreciar casos donde en el tiempo promedio, resulta mejor el de A* que el de IDA*, aunque si se toma en cuenta el número de nodos, es claro que IDA* mantiene la superioridad en cuanto a nodos por segundo. Una posible explicación a estas "excepciones" está en el hecho de que A* no resolvió todos los casos, y el tiempo promedio en las tablas solo toma en cuenta los casos resueltos. Entre los casos no resueltos de A*, encontramos la instancia 88 de Korf, de acuerdo a los id suministrados en los archivos de prueba que de acuerdo a los resultados experimentales del mismo Korf, resulta ser una de las más difíciles en términos de tiempo y nodos explorados, así como otras instancias relativamente complicadas. De hecho, el tiempo máximo de IDA* con la PDB A y PDB B a nivel de casos propuestos, es mucho mayor al tiempo máximo de A* con las mismas heurísticas. Por ejemplo: para la PDB B, el tiempo máximo de IDA* es más del doble que el de su contraparte.

El problema de 15 puzzle presenta otra situación interesante, ya que entre las cuatro heurísticas utilizadas, el error promedio (Sobre uno) en la estimación inicial de la distancia respecto al estado no varía tanto, a pesar de cambios en los tamaños de las PDBs (Solo ocurre una variación significativa entre la estimación de Manhattan respecto a las demás, de 13% o menos). Esta estimación podría ser una explicación de por qué las variaciones en cantidad de nodos explorados y tiempo utilizado no son tan notables como en otros problemas. Por ejemplo, en Top spin 12-4 y 14-4, se puede observar que entre la peor y la mejor heurística en términos de error promedio en la estimación inicial, este indicador se reduce en 56% y 48% en dicha ocasión y tanto el tiempo como los nodos promedio se reducen en un orden 10 veces. Sin embargo, este no parece ser el único factor que influye en la efectividad de la búsqueda, ya que para tener un error inicial prácticamente igual, existen claras diferencias entre los resultados de la PDB A y la PDB C

Es posible que se puedan obtener mejores resultados utilizando PDBs que solo consideren un blank y las fichas no seleccionadas sean reemplazadas por una ficha sin valor, de manera que el número de movimientos requeridos para alcanzar la solución desde

un estado de la abstracción se más parecido al número de movimientos en el problema real. Importante recordar, que de utilizar esta abstracción, el costo de mover una ficha sin valor debe ser 0 para preservar la aditividad de la heurística.

6.2- 24 Puzzle:

No hay mucho particular que comentar sobre este puzzle, más allá de recalcar la necesidad de utilizar PDBs que representen al menos seis piezas, ya que en la mayoría de los casos, con PDBs de menor tamaño la cota de IDA* no alcanzó el valor de la solución indicado por Korf. De hecho, algunos casos, como el 1, se quedaron bastante lejos, ya que la cota máxima reportada fue de 87, y la solución estaba a una profundidad de 95. Sin embargo, los resultados de la compilación en Google Colab, indican que el número de nodos por nivel ya estaba en descenso, lo que significa que faltaba relativamente poco para que terminara. Es posible que computadoras con 16 o 32 GB de RAM puedan compilarlas. Alternativamente, se podrían usar PDBs que usen sparse mapping como sugiere felner et al.

6.3- Top-spin 12-4:

Sin comentarios adicionales respecto a los generales

6.4- Top-Spin 14-4:

En las tablas correspondientes, se puede apreciar que ambos algoritmos cuando d=2000 que con d=2000000. Esto puede deberse a que la semilla utilizada para generar los casos con d=2000000, a pesar de tener una cota máxima mayor. También pudo ocurrir que se etiquetaran mal los archivos.

6.5- Top-Spin 17-4:

Se repite el fenómeno de d=2000 y d=2000000 descrito con Top-Spin 14-4.

6.6- Torres de Hanoi con 12 discos:

Este caso no fue particularmente difícil para las PDBs, ya que la computadora pudo calcularlas sin ninguna dificultad, incluso para la que tomaba en cuenta el problema con 11 discos. Para la elección de cuáles discos eliminar de la abstracción, se tomó en cuenta el paper de felner at al, en donde explican que se eliminan los más pequeños con el fin de usar los discos más grandes como índice en la PDB, y obtener el valor de la heurística correspondiente gracias a esto.

El algoritmo de A* no tuvo problemas para encontrar la solución en todos los casos de prueba seleccionados para este problema, además que lo hizo en un tiempo razonablemente corto debido a la poda de duplicados y a la selección óptima de qué nodos explorar usando la cola de prioridad. Para IDA* los resultados arrojados fueron un poco diferente, ya que para los casos de prueba más difíciles no logró encontrar las soluciones dentro del tiempo límite (15 min) y la cantidad de nodos totales antes de parar su ejecución es gigantesca en comparación con A*, la explicación de este comportamiento es debido a la débil eliminación de duplicados que brinda IDA* y la elevada cantidad de duplicados en el árbol de búsqueda de este problema.

6.7- Torres de Hanoi con 14 discos:

A partir de este caso, para las Torres de Hanoi, empezaron a haber problemas con el cálculo de las PDBs, ya que queríamos aplicar la misma estrategia que con el caso anterior y empezar con solo eliminar el disco más pequeño primero, pero al momento de calcularla nos quedamos sin memoria RAM, por lo que la PDB 13v1 no pudo ser utilizada en el cálculo de la solución para este problema. Se terminó haciendo dos corridas diferentes, cada una con su propio set de PDBs, para evaluar si al agregar una PDB más al problema podríamos encontrar más rápido la solución. El primer set de PDBs es comprendido por las PDBs 12v2, 11v3 y 10v4; el segundo set agrega una nueva PDB a este grupo, que sería la PDB 9v5.

El hecho de agregar una PDB más al solucionador del problema no generó mejoras notables al momento de la ejecución, ya que no existe diferencia en el número de nodos explorados con el set de 3 PDBs. El caso en donde podemos notar la diferencia, es en los resultados de encontrar la solución de un caso de prueba d=2000000 usando A* (cuyas tablas se encuentran en el apartado de los resultados), aquí podemos comparar las dos versiones y notar que la que encuentra más rápido la solución es la versión con un set de 4 PDBs, esto es debido a que el cálculo de los valores de la heurística se acerca más al valor verdadero que si lo calculamos con solo 3 PDBs. De esto podemos concluir que la mejor elección, para optimizar el uso de memoria RAM, es utilizar el set de 3 PDBs para calcular el valor de la heurística en un momento dado. En los resultados mostramos las tablas de ambos para evidenciar que casi no existen cambios entre ellos.

En cuanto a A*, es a partir de este momento que casi ningún caso de prueba difícil (d=2000000) pudo finalizar la ejecución, debido a que nos quedamos sin memoria RAM en nuestras máquinas en tan solo unos minutos. De resto, A* hace un buen trabajo en encontrar las soluciones, igual que con el caso de los 12 discos. Para IDA* No existen cambios en comparación con el caso anterior, se pudo encontrar la solución para casos de prueba fáciles, pero para los más difíciles el algoritmo casi nunca la encuentra. En comparación con los nodos totales y explorados, en este caso A* genera más nodos que IDA*, pero recorre o explora menos nodos que IDA* para encontrar la solución, esto gracias a que la selección de nodos a explorar en A* se hace por medio de una cola de prioridades.

6.8- Torres de Hanoi con 18 discos:

Al igual que sucedió con el caso de 14 discos, no pudimos calcular las PDBs que otorgan mejor precisión (Las que tienen más discos) debido a que en ninguna de nuestras máquinas la memoria RAM soportaba la cantidad de estados generados al compilar las PDBs. Para esto, se decidió diseñar un solo set de PDBs, empezando por un PDB poco preciso, 11v7, hasta la PDB 8v10.

No contamos con tablas de resultados para A* en d=2000000 debido a que con todos los casos de prueba, las máquinas se quedaban sin memoria RAM al momento por la generación de tantos nodos. Se intentó usar una menor cantidad de PDBs para liberar espacio en memoria, pero no nos ayudó de nada ya que igualmente la llenaba completamente. De nuevo, IDA* no presenta cambios en comparación con los casos anteriores, sigue consiguiendo con facilidad los casos fáciles, pero para los difíciles nunca logra llegar al estado final o solución dentro del tiempo límite, por lo que para estos casos de prueba tampoco tenemos tabla de resultados para IDA*, ya que no nos pareció necesaria ni informativa. Se repite el mismo fenómeno de los nodos totales y explorados entre IDA* y A*.

6.9- Rubik 3x3x3:

Para este caso, fue de gran importancia estudiar y entender cómo PSVN calcula las PDBs, debido a que estas para este problema ocupan bastante espacio en la memoria RAM, en comparación con los otros problemas abordados en el proyecto. Las PDBs propuestas por el enunciado para Rubik 3x3x3, o Rubik3, fueron: Eliminar todos los cubitos exceptuando los 8 que se encuentran en las esquinas, y eliminar todos los cubitos exceptuando los 12 bordes, los cuales serán particionados en grupos de 6 para generar dos PDBs de estos. Estas 3 PDBs requieren un espacio en memoria muy alto debido a la cantidad de estados por profundidad que Rubik3 tiene, la razón de esto es simple y es que para cualquier estado se tendrán 18 posibles movimientos válidos para pasar a otro. El hecho de que los estados posean tantas opciones de movimientos trae como consecuencia tener un factor de ramificación extremadamente alto, que será justamente 18 si no se realiza poda de duplicados y 13.5 si se realiza.

Con el fin de afrontar el problema del peso de las PDBs propuestas, decidimos diseñar otras que fueran menos pesadas y más rápidas de calcular. En cuanto a la PDB con los 8 cubitos esquina, decidimos diseñar 3 PDBs que representaran una abstracción del cubo con solo sus 8 cubitos esquina, pero eliminando 4 colores y dejando solo 2. Decidimos en hacer 3 PDBs de esta forma ya que en el primero dejábamos los 2 primeros colores, en el segundo los 2 medios y en el tercero los últimos 2 colores, esto con el fin de

poder tomar en cuenta todos los colores del cubo. Para las 2 PDBs de 6 cubitos bordes, decidimos diseñar 3 PDBs, parecidas a las anteriores, que representaran una abstracción del cubo con solo sus 12 cubitos bordes y eliminando 4 colores de la misma forma explicada. Intentamos quitar la menor cantidad de colores posibles de las abstracciones, pero en ninguna de nuestras máquinas, ni haciendo uso de Google Colab, se lograban compilar si dejábamos 3 o más colores.

Entre los dos algoritmos informados utilizados para encontrar la solución de manera óptima, A* fue el que peor desempeño tuvo entre los dos, dado que solo pudimos encontrar solución para los casos de prueba más sencillos (Los d=005), los siguientes se quedaban sin memoria RAM rápidamente en nuestras máquinas. Tratamos de aligerar la carga y utilizar solamente 4 PDBs en vez de 6, pero igualmente la corrida ocupaba toda la RAM disponible y tuvimos que abortar la ejecución. IDA* por otro lado completa casos de prueba más difíciles y nunca se queda sin memoria debido a que no necesita crear objetos de tipo nodo y mantiene su estado actual en una variable. Hasta los casos d=015 IDA* reporta la solución del problema, luego de eso para los casos d=020 en adelante nunca llega a la solución dentro de los 15 minutos que colocamos como límite.

Comparando los resultados de ambos algoritmos, podemos ver que IDA* toma en cuenta o "genera" menos nodos para encontrar la solución que A*, pero este explora menos nodos que IDA*, por lo que la cola de prioridad usada en A* para escoger el próximo nodo a explorar le da una ventaja en este aspecto. Es posible que para computadoras con mayor capacidad de memoria RAM sea preferible el uso de A*, pero no podemos concluir nada con respecto al tiempo de ejecución ya que los dos algoritmos tomaron la misma cantidad de segundos en encontrar la solución en d=005, solo que en teoría A* debería ser más rápido en encontrar la solución, a costa de necesitar gran parte de la memoria.

7- Bibliografía:

- Felner, A., Korf, R. E., & Hanan, S. (2004). Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, *22*, 279-318.
- Korf, R. E. (1997, July). Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI/IAAI* (pp. 700-705).