UNIVERSIDAD SIMÓN BOLÍVAR
DPTO. COMPUTACIÓN Y TEC. DE INF.
CI-5437 INTELIGENCIA ARTIFICIAL I

Informe Proyecto 1

Estudiantes
LEONEL GUERRERO
EROS CEDEÑO

Profesor
CARLOS INFANTE

14 de junio de 2023

Contenido General

1. Introducción

La inteligencia artificial esta ganando renombre cada vez mas. Comenzó en 1943 con la publicación del artículo «A Logical Calculus of Ideas Immanent in Nervous Activity» de Warren McCullough y Walter Pitts. En ese trabajo, los científicos presentaron el primer modelo matemático para la creación de una red neuronal. Y desde entonces ha cruzado caminos inhóspitos con avances importantes y épocas de estancamiento que sugerían el final de esta área. Sin embargo cada vez la inteligencia artificial a salido adelante y en nuestro días promete ser una realidad y tan común como lo es hoy la luz eléctrica. Y por este y otros motivos, esta asignatura cobra mayor relevancia

Para este primer proyecto de la asignatura se solicita elaborar un conjunto de algoritmos informados de búsqueda y recorrido en grafos para resolver un conjunto de juegos populares. Los juegos estudiados son: la torre de Hanoi, los sliding puzzles, el cubo de Rubik y el top spin. Cada uno de estos problemas con variaciones. Los Algoritmos implementados son A*, BFS, DFS, IDA* y IDDFS. Se solicita utilizar diversas heurísticas de búsquedas para estimar adecuadamente los costos de los caminos.

En este informe se presentan las consideraciones, resultados y observaciones obtenidas durante la elaboración de este primer proyecto de la asignatura.

2. Implementacion general

Se implementaron diferentes algoritmos para recorrer grafos implicitos, entre los cuales se encuentran BFS, DFS, IDDFS, A* e IDA*, para las implementaciones se utilizo la herramienta de psvn (la cual provee una API en C) para el modelaje e implementacion de los mapas de estados de los problemas. Para la implementacion de los algoritmos se utilizo el lenguaje de programacion C++, los problemas analizados fueron:

- Hanoi Tower: de 4 barras
  - 12 Discos
  - 14 Discos
  - 18 Discos
- Sliding Puzzle
  - 15-Puzzle
  - 24-Puzzle
- Rubik Cube 3x3
- Top Spin: de 4 intercambios
  - 12 numeros
  - 14 numeros
  - 17 numeros

La implementacion de los algoritmos se realizo de manera general para no tener que depender del modelo en especifico de algun problema, para esto se creo un wrapper que se encara de llamar correctamente a cada metodo proveido por la API de PSVN, tambien se creo un archivo que exportara los tipos necesarios para el correcto funcionamiento de los algoritmos, de esta manera se puede utilizar el mismo codigo para todos los problemas. Para la implementacion de los algoritmos se utilizo la libreria de STL de C++ para la implementacion de las estructuras de datos necesarias para la ejecucion de los algoritmos.

Sin embago como la definicion de cada estado es especifico de cada problema, es decir el numero de variables de cada problema varia y este es necesario para la correcta definicion del tipo de estado del problema se implemento un script en python que realizara la compilazacion de cada ejecutable para cada problema, se creo un ejecutable por cada problema para evitar que la definicion del tipo de cada estado conflictuara con la de algun otro.

3. Especificaciones de la maquina de pruebas

Para la ejecucion de los algoritmos se utilizo una maquina con las siguientes especificaciones:

- Procesador: 12th Gen Intel i9-12900H (20) @ 4.900GHz
- Memoria RAM: 16GB
- Sistema Operativo: Ubuntu 22.04

4. Experimentacion

El programa principal, esta diseñado para recibivir varios parametro por entrada estandar, los cuales son:

- Algoritmo a utilizar
- Instacia del problema a resolver
- Tiempo maximo de ejecucion
- Nombre del problema, este solo es requerido para ejecutar A* e IDA*
- Tipo de heuristica a utilizar, este solo es requerido para ejecutar A* e IDA*

Sin embargo a pesar de que el tiempo limite de ejecucion es un parametro que se puede modificar, para la experimentacion se utilizo un tiempo limite de 15 minutos para cada ejecucion, esto debido a que algunos problemas son muy grandes y el tiempo de ejecucion de algunos algoritmos es muy grande, por lo que se decidio utilizar un tiempo limite de 5 minutos para cada ejecucion, para que la experimentacion no se tardara demasiado. No obstante a pesar del tiempo limite el programa esta preparado para el caso en que se quede sin memoria, en este caso el programa termina la ejecucion, muestra un mensaje de error y muestra los resultados obtenidos hasta el momento. Para que el programa pudiera lograr detectar correctamente cuando se quedaba sin memoria se tuvo que cambiar el modo en como se maneja la memoria en en sistema operativo ya que por defecto este realiza un overcommit de la memoria, lo que significa que el sistema operativo asigna mas memoria de la que tiene disponible, esto para evitar que el programa se quede sin memoria, sin embargo esto hace que el programa no pueda detectar cuando se queda sin memoria, por lo que se tuvo que desactivar el overcommit de la memoria para que el programa pudiera detectar cuando se quedaba sin memoria.

Para la experimentacion se estan midiendo 4 parametros:

- Si se encontro una solucion, en los resultados se denora por una f
- Numero de nodos generados, en los resultados se denora por un g
- Numero de nodos expandidos, en los resultados se denora por un e
- Maxima profundidad alcanzada, en los resultados se denora por un d
- Tiempo de ejecucion, en los resultados se denora por un t

De igual forma se estudio sin y con poda de ancestros, el cual se denota por h0 y h1 respectivamente. Por otras parte se estudiaron diferentes dificultades para cada problema a excepcion de n-puzzle, en donde este se denota por la terminacion de las columnas en las tablas, en donde entre mas grande mas dificil es la instacia del problema, para el n-puzzle las instancias no tienen una dificultad en particular.

Para la experimentacion de cada algoritmo se implemento un script en python que automatizara la ejecucion de cada algoritmo para cada problema, se ejecutaron 10 instacias de cada problema (a excepcion del problema de top spin), para cada algoritmo y se tomo el promedio de los tiempos de ejecucion.

5. Sliding Puzzle

5.1. ¿En que consiste Sliding Puzzle?

Un rompecabezas deslizante o rompecabezas de piezas deslizantes es un rompecabezas que reta al usuario a deslizar piezas a lo largo de rutas para llegar a una configuración determinada final. En nuestro caso la configuración objetivo es un ordenamiento correcto de los números escritos en las fichas del tablero.

5.2. Implementacion

La libreria PSVN originalmente creada en la Universidad de Alberta, proporcionada una API en C++ para modelar y consultar adecuadamente Espacios de Estados. Esta fue sumamente util para abstraer los problemas 15-puzzle y 24-puzzle.

Para obtener heuristicas adecuadas se crearon multiples PDBs aditivos y se suman para obtener un estimado del costo de los caminos. Para el 15-puzzle se agruparon las fichas del tablero en dos grupos: uno correspondiente a la mitad superior del tablero y el segundo a mitad inferior. Para el caso del 25-puzzle se dividio el tablero en 6 conjuntos disjuntos y adyacentes y se genero para cada uno de estos su correspondiente PDB. Por otra parte tambien se implemento la heuristica de Manhattan Distance para comparar los resultados obtenidos con las heuristicas aditivas.

5.3. Resultados

5.3.1. 15-puzzle

Con la heuristica de Manhattan Distance se obtuvieron los siguientes resultados:

pruning	algorithm	info	15puzzle100instanceskorf.txt
15-puzzleh0	astar	f	0.4
15-puzzleh0	idastar	f	0
15-puzzleh1	astar	f	0.4
15-puzzleh1	idastar	f	0
15-puzzleh0	astar	g	E6
15-puzzleh0	idastar	g	E7
15-puzzleh1	astar	g	E6
15-puzzleh1	idastar	g	E7
15-puzzleh0	astar	e	E6
15-puzzleh0	idastar	e	E7
15-puzzleh1	astar	e	E6
15-puzzleh1	idastar	e	E7
15-puzzleh0	astar	d	52.4
15-puzzleh0	idastar	d	44
15-puzzleh1	astar	d	52.4
15-puzzleh1	idastar	d	44
15-puzzleh0	astar	t	18.12
15-puzzleh0	idastar	t	27.98
15-puzzleh1	astar	t	18.7
15-puzzleh1	idastar	t	27.44

Con la heuristica aditiva se obtuvieron los siguientes resultados:

pruning	algorithm	info	15puzzle100instanceskorf.txt
h0	bfs	f	0
h0	iddfs	f	0
h0	astar	f	0
h0	idastar	f	0
h1	bfs	f	0
h1	iddfs	f	0
h1	astar	f	0
h1	idastar	f	0
h0	bfs	g	E7
h0	iddfs	g	E7
h0	astar	g	E6
h0	idastar	g	E7
h1	bfs	g	E7
h1	iddfs	g	E7
h1	astar	g	E6
h1	idastar	g	E7
h0	bfs	e	E7
h0	iddfs	e	E7
h0	astar	e	E6
h0	idastar	e	E7
h1	bfs	e	E7
h1	iddfs	e	E7
h1	astar	e	E6
h1	idastar	e	E7
h0	bfs	d	15.6
h0	iddfs	d	15.6
h0	astar	d	48
h0	idastar	d	42.6
h1	bfs	d	15.6
h1	iddfs	d	15.6
h1	astar	d	48
h1	idastar	d	42.6
h0	bfs	t	17.46
h0	iddfs	t	18.65
h0	astar	t	24.21
h0	idastar	t	62.71
h1	bfs	t	18.68
h1	iddfs	t	19.06
h1	astar	t	25.06
h1	idastar	t	59.43

5.3.2. 24-puzzle

Con la heuristica de Manhattan Distance se obtuvieron los siguientes resultados:

pruning	algorithm	info	24puzzle50instanceskorf.txt
24-puzzleh0	astar	f	0
24-puzzleh0	idastar	f	0
24-puzzleh1	astar	f	0
24-puzzleh1	idastar	f	0
24-puzzleh0	astar	g	E7
24-puzzleh0	idastar	g	E7
24-puzzleh1	astar	g	E7
24-puzzleh1	idastar	g	E7
24-puzzleh0	astar	e	E6
24-puzzleh0	idastar	e	E7
24-puzzleh1	astar	e	E6
24-puzzleh1	idastar	e	E7
24-puzzleh0	astar	d	59.8
24-puzzleh0	idastar	d	53.8
24-puzzleh1	astar	d	59.4
24-puzzleh1	idastar	d	53.8
24-puzzleh0	astar	t	24.21
24-puzzleh0	idastar	t	30.79
24-puzzleh1	astar	t	23.2
24-puzzleh1	idastar	t	29.09

Con la heuristica aditiva se obtuvieron los siguientes resultados:

pruning	algorithm	info	24puzzle50instanceskorf.txt
h0	bfs	f	0
h0	iddfs	f	0
h0	astar	f	0
h0	idastar	f	0
h1	bfs	f	0
h1	iddfs	f	0
h1	astar	f	0
h1	idastar	f	0
h0	bfs	g	E7
h0	iddfs	g	E7
h0	astar	g	E6
h0	idastar	g	E7
h1	bfs	g	E7
h1	iddfs	g	E7
h1	astar	g	E6
h1	idastar	g	E7
h0	bfs	e	E7
h0	iddfs	e	E7
h0	astar	e	E6
h0	idastar	e	E7
h1	bfs	e	E7
h1	iddfs	e	E7
h1	astar	e	E6
h1	idastar	e	E7
h0	bfs	d	14.8
h0	iddfs	d	14.4
h0	astar	d	34.2
h0	idastar	d	29
h1	bfs	d	14.8
h1	iddfs	d	14.4
h1	astar	d	34.2
h1	idastar	d	29
h0	bfs	t	17.79
h0	iddfs	t	18.57
h0	astar	t	35.46
h0	idastar	t	76.05
h1	bfs	t	17.93
h1	iddfs	t	18.02
h1	astar	t	33.46
h1	idastar	t	75.95

6. Torres de Hanoi

6.1. ¿En que consiste las Torres de Hanoi?

El juego, en su forma más tradicional, consiste en tres varillas verticales. En una de las varillas se apila un número n-esimo de discos que determinará la complejidad de la solución. Los discos se apilan sobre una varilla en tamaño decreciente. No hay dos discos iguales, y todos ellos están apilados de mayor a menor radio en una de las varillas, quedando las otras dos varillas vacantes. El juego consiste en pasar todos los discos de la varilla ocupada (es decir la que posee la torre) a una de las otras varillas vacantes. Para realizar este objetivo, es necesario seguir tres simples reglas:

1. Sólo se puede mover un disco cada vez.
2. Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
3. Sólo puedes desplazar el disco que se encuentre arriba en cada varilla.

6.2. Implementacion

Para las heuristicas de busquedas se siguio el consejo dado por los autores Ariel Felner, Richard E Korf y Sarit Hanan en su articulo Additive Pattern Database Heuristic y se abstrajo el problema mediante el agrupamiento de los discos en grupos disjuntos y adyacentes.

Para la version del juego con 4 pilares y 12 discos se crearon dos grupos de discos, uno de 8 y otro de 4. Para el de 4 pilares con 14 los grupos fueron de 8 y 6. Y finalmente para la version de 4 pilares con 18 discos se dividio en 3 PDBs de 8-6-4.

6.3. Resultados

6.3.1. 4 pilares y 12 discos

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
24-puzzleh1	bfs	f	0	0	0	0	0
24-puzzleh1	iddfs	f	0	0	0	0	0
24-puzzleh1	astar	f	0	0	0	0	0
24-puzzleh1	idastar	f	0	0	0	0	0
24-puzzleh0	bfs	f	0	0	0	0	0
24-puzzleh0	iddfs	f	0	0	0	0	0
24-puzzleh0	astar	f	0	0	0	0	0
24-puzzleh0	idastar	f	0	0	0	0	0
24-puzzleh1	bfs	g	E7	E7	E7	E7	E7
24-puzzleh1	iddfs	g	E7	E7	E7	E8	E8
24-puzzleh1	astar	g	E7	E7	E7	E7	E7
24-puzzleh1	idastar	g	E7	E7	E7	E7	E7
24-puzzleh0	bfs	g	E7	E7	E7	E7	E7
24-puzzleh0	iddfs	g	E7	E7	E7	E8	E8
24-puzzleh0	astar	g	E7	E7	E7	E7	E7
24-puzzleh0	idastar	g	E7	E7	E7	E7	E7
24-puzzleh1	bfs	e	E6	E6	E6	E6	E6
24-puzzleh1	iddfs	e	E6	E6	E6	E6	E6
24-puzzleh1	astar	e	E5	E5	E5	E5	E5
24-puzzleh1	idastar	e	E6	E6	E6	E6	E6
24-puzzleh0	bfs	e	E6	E6	E6	E6	E6
24-puzzleh0	iddfs	e	E6	E6	E6	E6	E6
24-puzzleh0	astar	e	E5	E5	E5	E5	E5
24-puzzleh0	idastar	e	E6	E6	E6	E6	E6
24-puzzleh1	bfs	d	5	5	5	5	5
24-puzzleh1	iddfs	d	5	5	5	5	5
24-puzzleh1	astar	d	15	14	13	20	18
24-puzzleh1	idastar	d	5	6	6	5	6
24-puzzleh0	bfs	d	5	5	5	5	5
24-puzzleh0	iddfs	d	5	5	5	5	5
24-puzzleh0	astar	d	15	14	13	20	18
24-puzzleh0	idastar	d	5	6	6	5	6
24-puzzleh1	bfs	t	56.47	56.35	67.41	63.14	61.54
24-puzzleh1	iddfs	t	64.85	70.64	58.08	57.43	60.12
24-puzzleh1	astar	t	100.23	110.13	104.9	89.04	92.04
24-puzzleh1	idastar	t	199.38	199.7	204.99	201.86	198.87
24-puzzleh0	bfs	t	57.43	63.42	57.44	66.25	55.91
24-puzzleh0	iddfs	t	64.42	62.33	60.67	66.26	66.27
24-puzzleh0	astar	t	102.2	103.74	102.4	90.65	91.67
24-puzzleh0	idastar	t	198.09	201.31	199.14	201.87	200.02

6.3.2. 4 pilares y 14 discos

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
h0	bfs	f	1	1	1	0	0
h0	iddfs	f	1	1	1	0	0
h0	astar	f	1	1	1	1	0
h0	idastar	f	1	1	1	1	0
h1	bfs	f	1	1	1	0	0
h1	iddfs	f	1	1	1	0	0
h1	astar	f	1	1	1	1	0
h1	idastar	f	1	1	1	1	0
h0	bfs	g	4945	E6	E5	E7	E7
h0	iddfs	g	5965	E6	E5	E7	E7
h0	astar	g	34	69	57	261	E6
h0	idastar	g	14	30	27	96	E7
h1	bfs	g	4945	E6	E5	E7	E7
h1	iddfs	g	5965	E6	E5	E7	E7
h1	astar	g	34	69	57	261	E6
h1	idastar	g	14	30	27	96	E7
h0	bfs	e	848	E5	E5	E7	E7
h0	iddfs	e	1024	E5	E5	E7	E7
h0	astar	e	6	12	10	44	E6
h0	idastar	e	4	7	7	21	E7
h1	bfs	e	848	E5	E5	E7	E7
h1	iddfs	e	1024	E5	E5	E7	E7
h1	astar	e	6	12	10	44	E6
h1	idastar	e	4	7	7	21	E7
h0	bfs	d	5	8	8	11	11
h0	iddfs	d	5	8	8	10	10
h0	astar	d	4	7	7	21	54
h0	idastar	d	4	7	7	21	28
h1	bfs	d	5	8	8	11	11
h1	iddfs	d	5	8	8	10	10
h1	astar	d	4	7	7	21	54
h1	idastar	d	4	7	7	21	28
h0	bfs	t	1	1.04	1.2	71.8	72.35
h0	iddfs	t	1	2.02	1.01	60.33	63
h0	astar	t	1	1	1	1.16	904.6
h0	idastar	t	1	1	1	1	161.09
h1	bfs	t	1	1.07	1.24	71.11	70.33
h1	iddfs	t	1	2.02	1.01	62.04	63.72
h1	astar	t	1	1	1	1.2	902.79
h1	idastar	t	1	1	1	1	159.55

6.3.3. 4 pilares y 18 discos

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
h0	bfs	f	1	1	1	0	0
h0	iddfs	f	1	1	1	0	0
h0	astar	f	1	1	1	1	0
h0	idastar	f	1	1	1	1	0
h1	bfs	f	1	1	1	0	0
h1	iddfs	f	1	1	1	0	0
h1	astar	f	1	1	1	1	0
h1	idastar	f	1	1	1	1	0
h0	bfs	g	436	4945	E6	E7	E7
h0	iddfs	g	530	5965	E6	E7	E7
h0	astar	g	16	34	69	261	E6
h0	idastar	g	7	14	32	97	E7
h1	bfs	g	436	4945	E6	E7	E7
h1	iddfs	g	530	5965	E6	E7	E7
h1	astar	g	16	34	69	261	E6
h1	idastar	g	7	14	32	97	E7
h0	bfs	e	78	848	E5	E7	E7
h0	iddfs	e	97	1024	E6	E7	E7
h0	astar	e	3	6	12	44	E6
h0	idastar	e	3	4	8	21	E7
h1	bfs	e	78	848	E5	E7	E7
h1	iddfs	e	97	1024	E6	E7	E7
h1	astar	e	3	6	12	44	E6
h1	idastar	e	3	4	8	21	E7
h0	bfs	d	4	5	9	10	10
h0	iddfs	d	4	5	9	10	10
h0	astar	d	3	4	8	21	61
h0	idastar	d	3	4	8	21	20
h1	bfs	d	4	5	9	10	10
h1	iddfs	d	4	5	9	10	10
h1	astar	d	3	4	8	21	61
h1	idastar	d	3	4	8	21	20
h0	bfs	t	1.09	1.2	3.05	76.43	78.07
h0	iddfs	t	1	1	4.06	59.03	59.9
h0	astar	t	4	5.01	1	4.35	902.46
h0	idastar	t	1	1	1	1	156.62
h1	bfs	t	1.08	1.23	3.05	66.13	76.11
h1	iddfs	t	1	1	4.06	60.22	59.59
h1	astar	t	1	4	1	4.34	902.52
h1	idastar	t	1	1	1	1	157.71

7. Cubo de Rubik

7.1. ¿En que consiste el cubo de Rubik?

El cubo de rubik es un mecanismo de ejes que permite que cada una de sus caras gire independientemente, mezclando los colores, para solucionar el rompecabezas, cada una de sus caras debe estar compuesto en un solo color.

7.2. Implementación

La libreria PSVN proporciona la definicion del espacio de estados para el cubo rubik, y esta fue utilizada como estructura para recorrer el grafo permutaciones validas del cubo de rubik.

En concordancia con el articulo publicado por Richard E. Korf "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases"la funcion de costo heurística consistió en crear PDBs no aditivas y calcular el valor máximo entre diversas PDBs. Inicialmente como abstracción se considero las 8 esquinas del cubo y adicionalmente 2 PDBs considerando 6 esquinas cada una. Sin embargo al ejecutar el analisis y el mapeo de los PDBs se encontro una restriccion de memoria que dificulto el calculo de las PDBs con estas caracteristicas por lo que se opto por generar una mayor cantidad de PDBs pero con menor cantidad de variables libres de tal manera que de las PDBs se simplifique aun mas. Se crearon 4 PDBs representando las esquinas del cubo y 4 PDBs adicionales que almacenan patrones relacionados a las esquinas.

7.3. Resultados

pruning	algorithm	info	.d=5	.d=15	.d=20	.d=40	.d=160
h0	bfs	f	1	0	0	0	0
h0	iddfs	f	1	0	0	0	0
h0	astar	f	0	0	0	0	0
h0	idastar	f	0	0	0	0	0
h1	bfs	f	1	0	0	0	0
h1	iddfs	f	1	0	0	0	0
h1	astar	f	0	0	0	0	0
h1	idastar	f	0	0	0	0	0
h0	bfs	g	E6	E7	E7	E7	E7
h0	iddfs	g	E6	E7	E7	E7	E7
h0	astar	g	E7	E7	E7	E7	E7
h0	idastar	g	E7	E7	E7	E7	E7
h1	bfs	g	E6	E7	E7	E7	E7
h1	iddfs	g	E6	E7	E7	E7	E7
h1	astar	g	E7	E7	E7	E7	E7
h1	idastar	g	E7	E7	E7	E7	E7
h0	bfs	e	E4	E6	E6	E6	E6
h0	iddfs	e	E4	E6	E6	E6	E6
h0	astar	e	E6	E6	E6	E6	E6
h0	idastar	e	E6	E6	E6	E6	E6
h1	bfs	e	E4	E6	E6	E6	E6
h1	iddfs	e	E4	E6	E6	E6	E6
h1	astar	e	E6	E6	E6	E6	E6
h1	idastar	e	E6	E6	E6	E6	E6
h0	bfs	d	5	7	7	7	7
h0	iddfs	d	5	7	7	7	7
h0	astar	d	12	13	12	13	12
h0	idastar	d	25	33	25	33	25
h1	bfs	d	5	7	7	7	7
h1	iddfs	d	5	7	7	7	7
h1	astar	d	12	13	12	13	12
h1	idastar	d	25	33	25	33	25
h0	bfs	t	1.01	15.77	14.42	14.4	15.08
h0	iddfs	t	1.01	16.11	14.82	14.7	14.91
h0	astar	t	125.65	130.31	123.95	127.18	131.3
h0	idastar	t	143.12	130.61	142.11	141.59	131.99
h1	bfs	t	1.02	14.28	15.2	14.61	15.01
h1	iddfs	t	1.01	14.72	14.82	14.32	15.67
h1	astar	t	125.02	129.97	122.63	122.71	132.86
h1	idastar	t	131.19	140.78	131.82	130.17	131.53

8. Top Spin

8.1. ¿En que consiste Top Spin?

El rompecabezas TopSpin fue diseñado originalmente por Ferdinand Lammertink y patentado en 1988. El rompecabezas en sí consiste en una pista ovalada que contiene N elementos. Los elementos se pueden mover en la pista cambiando todos a la vez en cualquier dirección. La única forma de cambiar el orden de los elementos es girar la parte superior del rompecabezas que resulta en invertir el orden de los 4 elementos en la rueda. El rompecabezas TopSpin es uno de los más intrigantes que se pueden colocar en el mismo estante como el cubo de Rubik al comparar posibles permutaciones. Con su estándar 20! distintas permutaciones casi supera al cubo de Rubik en complejidad pero obviamente se ve menos impresionante y no es una hazaña de ingeniería; nunca alcanzó el mismo nivel de fama. Pero resolverlo es todavía bastante complicado incluso para una computadora. El el espacio de búsqueda para este problema es demasiado grande que la memoria necesaria para encontrar una solución puede superar a la mayoría de las PC estándar. Por otro lado, encontrar una estrategia de resolución eficiente es igualmente desafiante

8.2. Implementacion

Al igual que en la mayoría de los problemas anteriores se recurrió a un PDBs no aditivas y se retorna el valor de la heurística como el máximo entre varios PDBs, pero a diferencia de las heurísticas empleadas en los problemas anteriores, para este no se realizaron proyecciones sobre las variables del dominio, sino que se opto por mapear valores de fichas a un valor constante. Se considero TopSpin 12-4, 14-4 y 17-4. Para los diferentes tamaños y tipos de tableros considerados se mapearon rangos finitos de fichas hacia la ficha cero tal que sean indistinguibles entre si y obtener una abstracción del problema original y con esta obtener estimaciones adecuadas.

8.3. Resultados

Resultados con y sin poda de estados repetidos

8.3.1. Top Spin 12-4

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
h1	bfs	f	1	0.2	0	0.1	0
h1	iddfs	f	1	0.2	0	0.1	0
h1	astar	f	1	1	1	1	1
h1	idastar	f	1	1	1	1	1
h0	bfs	f	1	0.2	0	0.1	0
h0	iddfs	f	1	0.2	0	0.1	0
h0	astar	f	1	1	1	1	1
h0	idastar	f	1	1	1	1	1
h1	bfs	g	E6	E7	E7	E7	E7
h1	iddfs	g	E6	E7	E8	E8	E8
h1	astar	g	501.6	E4	E5	E5	E5
h1	idastar	g	1131.5	E5	E6	E6	E6
h0	bfs	g	E6	E7	E7	E7	E7
h0	iddfs	g	E6	E7	E8	E8	E8
h0	astar	g	501.6	E4	E5	E5	E5
h0	idastar	g	1131.5	E5	E6	E6	E6
h1	bfs	e	E5	E6	E6	E6	E6
h1	iddfs	e	E5	E6	E6	E6	E6
h1	astar	e	41.8	2989.3	E4	E4	E4
h1	idastar	e	96.4	E4	E4	E5	E5
h0	bfs	e	E5	E6	E6	E6	E6
h0	iddfs	e	E5	E6	E6	E6	E6
h0	astar	e	41.8	2989.3	E4	E4	E4
h0	idastar	e	96.4	E4	E4	E5	E5
h1	bfs	d	6	7.4	8	8	8
h1	iddfs	d	6	7.4	8	8	8
h1	astar	d	5.9	8.2	9.8	9.9	10.1
h1	idastar	d	5.8	8.3	9.8	9.9	10.1
h0	bfs	d	6	7.4	8	8	8
h0	iddfs	d	6	7.4	8	8	8
h0	astar	d	5.9	8.2	9.8	9.9	10.1
h0	idastar	d	5.8	8.3	9.8	9.9	10.1
h1	bfs	t	1.01	13.95	17.35	17.62	17.29
h1	iddfs	t	1.01	14.77	18.15	18.67	18.11
h1	astar	t	1	1	1.01	1.01	1.11
h1	idastar	t	1	1.11	1.41	2.31	3.42
h0	bfs	t	1.01	14.25	17.65	17.99	17.66
h0	iddfs	t	1.01	14.96	18.53	18.86	18.45
h0	astar	t	1	1	1.01	1.01	1.11
h0	idastar	t	1	1.11	1.41	2.31	3.52

8.3.2. Top Spin 14-4

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
h1	bfs	f	1	0.1	0	0	0
h1	iddfs	f	1	0.1	0	0	0
h1	astar	f	1	1	1	1	1
h1	idastar	f	1	0.9	0.6	0.3	0.7
h0	bfs	f	1	0.1	0	0	0
h0	iddfs	f	1	0.1	0	0	0
h0	astar	f	1	1	1	1	1
h0	idastar	f	1	0.9	0.6	0.3	0.7
h1	bfs	g	E6	E7	E8	E8	E8
h1	iddfs	g	E6	E8	E8	E8	E8
h1	astar	g	665	E5	E6	E6	E5
h1	idastar	g	932.3	E7	E7	E7	E7
h0	bfs	g	E6	E7	E8	E8	E8
h0	iddfs	g	E6	E8	E8	E8	E8
h0	astar	g	665	E5	E6	E6	E5
h0	idastar	g	932.3	E7	E7	E7	E7
h1	bfs	e	E5	E6	E6	E6	E6
h1	iddfs	e	E5	E6	E6	E6	E6
h1	astar	e	47.5	9071.3	E4	E5	E4
h1	idastar	e	68.4	E6	E6	E6	E6
h0	bfs	e	E5	E6	E6	E6	E6
h0	iddfs	e	E5	E6	E6	E6	E6
h0	astar	e	47.5	9071.3	E4	E5	E4
h0	idastar	e	68.4	E6	E6	E6	E6
h1	bfs	d	6	7	7	7	7
h1	iddfs	d	6	7	7	7	7
h1	astar	d	5.6	9.9	12.1	13.2	11.9
h1	idastar	d	5.9	9.9	11.8	12.6	11.9
h0	bfs	d	6	7	7	7	7
h0	iddfs	d	6	7	7	7	7
h0	astar	d	5.6	9.9	12.1	13.2	11.9
h0	idastar	d	5.9	9.9	11.8	12.6	11.9
h1	bfs	t	1.12	15.84	17.63	17.44	17.33
h1	iddfs	t	1.12	16.81	18.47	18.43	18.43
h1	astar	t	1.01	1.01	2.42	8.75	2.02
h1	idastar	t	1.01	9.44	39.97	55.27	24.52
h0	bfs	t	1.13	16.37	17.67	17.9	17.61
h0	iddfs	t	1.12	16.95	18.42	18.49	18.54
h0	astar	t	1.01	1.01	2.62	8.45	2.02
h0	idastar	t	1.01	9.41	40.11	55.08	24.6

8.3.3. Top Spin 17-4

pruning	algorithm	info	.d=5	.d=10	.d=15	.d=2E+3	.d=E6
h0	bfs	f	1	0	0	0	0
h0	iddfs	f	1	0	0	0	0
h0	astar	f	1	1	0.6	0	0
h0	idastar	f	1	0.9	0	0	0
h1	bfs	f	1	0	0	0	0
h1	iddfs	f	1	0	0	0	0
h1	astar	f	1	1	0.6	0	0
h1	idastar	f	1	0.9	0	0	0
h0	bfs	g	E6	E7	E7	E7	E7
h0	iddfs	g	E6	E7	E7	E7	E7
h0	astar	g	3556.4	E6	E7	E7	E7
h0	idastar	g	7844.3	E7	E7	E7	E7
h1	bfs	g	E6	E7	E7	E7	E7
h1	iddfs	g	E6	E7	E7	E7	E7
h1	astar	g	3556.4	E6	E7	E7	E7
h1	idastar	g	7844.3	E7	E7	E7	E7
h0	bfs	e	E5	E6	E6	E6	E6
h0	iddfs	e	E5	E6	E6	E6	E6
h0	astar	e	209.2	E4	E6	E6	E6
h0	idastar	e	463.1	E6	E6	E6	E6
h1	bfs	e	E5	E6	E6	E6	E6
h1	iddfs	e	E5	E6	E6	E6	E6
h1	astar	e	209.2	E4	E6	E6	E6
h1	idastar	e	463.1	E6	E6	E6	E6
h0	bfs	d	5.8	7	7	7	7
h0	iddfs	d	5.8	7	7	7	7
h0	astar	d	5.6	10.5	13.1	14.5	14.5
h0	idastar	d	5.5	10.4	11.5	12.6	12.6
h1	bfs	d	5.8	7	7	7	7
h1	iddfs	d	5.8	7	7	7	7
h1	astar	d	5.6	10.5	13.1	14.5	14.5
h1	idastar	d	5.5	10.4	11.5	12.6	12.6
h0	bfs	t	2.13	17.25	17.35	16.99	17.48
h0	iddfs	t	2.13	18.16	18.08	17.86	18.41
h0	astar	t	1.01	3.73	37.02	62.71	63.53
h0	idastar	t	1.01	15.16	76.24	77.45	77.08
h1	bfs	t	2.03	17.11	17.06	16.89	17.05
h1	iddfs	t	2.13	17.49	18.01	17.61	17.67
h1	astar	t	1.01	4.03	36.62	62.84	62.25
h1	idastar	t	1.01	15.58	76.51	76.55	76.87

9. Analisis de Resutlados

Se puede apreciar que los 4 algoritmos estudiados se comportan de manera diferente, en donde A* es el algoritmo que en promedio es el que mas logra encontrar la solucion en los diferentes problemas, seguido de IDA* y por ultimo los algoritmos de busqueda no informada. Por otro lado, se puede apreciar que los algoritmos de busqueda no informada no logran encontrar la solucion en los problemas mas grandes, esto se debe a que los algoritmos de busqueda no informada no tienen una heuristica que les permita guiar la busqueda hacia la solucion, por lo que en los problemas mas grandes se pierde mucho tiempo explorando estados que no son utiles para encontrar la solucion. Por otro lado, se puede apreciar que los algoritmos informados logran encontrar la solucion en algunos de los problemas grandes, esto se debe a que los algoritmos informados tienen una heuristica que les permite guiar la busqueda hacia la solucion, por lo que en los problemas mas grandes se pierde menos tiempo explorando estados que no son utiles para encontrar la solucion.

De igual forma se puede apreciar que si los algorimos no informados encuentra la solucion, siempre sucede que los algoritmos informados tambien la encuentran, esto se debe a que los algoritmos informados son una generalizacion de los algoritmos no informados, por lo que si los algoritmos no informados encuentran la solucion, los algoritmos informados tambien la encuentran.

En relacion a la generacion de nodos se puede apreciar que los algoritmos informados generan menos nodos que los algoritmos no informados, esto debido a el calculo de la heuristica, ya que este ayuda a generar menos estados en la busqueda hacia la solucion, esto en los casos donde ambos encuentra la solucion, en casos donde los informados la encuentra y los no informados no, la diferencia en la generecion es mucho mas grande, y en el caso donde ninguno la encuentra la cantidad de generacion es la misma, y esta solo esta limitada por la cantidad de memoria disponible.

Detallando la profundidad maxima alcanzada por los algoritmos, se puede apreciar que el algoritmo que en general logra obtener una mayor profundidad es el A* seguido de IDA*, y por ultimo los algoritmos no informados, esto cundo ninguno de los algoritmos encuentra la solucion, en el caso donde los algoritmos informados encuentran la solucion, se puede apreciar que es gracias tambien a que logran ir a una mayor profundidad en relacion a los no informados, y en el caso donde ambos encuentran la solucion, la profundidad alcanza es muy similar debido que en esa profundidad es donde se encuentra la solucion.

En cuando a la cantidad de memoria utilizada, se puede apreciar que los algoritmos informados utilizan menos memoria que los algoritmos no informados, esto debido a que los algoritmos informados generan menos nodos que los algoritmos no informados, por lo que la cantidad de memoria utilizada es menor, esto se puede ver reflejado indirectamente en el tiempo de ejecucion, ya que los algoritmos informados logran ejecutarse durante mayor tiempo antes de que se acabe la memoria disponible.

En relacion a las heuristicas se puede apreciar que el n-puzzle que la elecion de una buena heuristica puede marcar una diferencia importante, ya que en el caso de la heuristica pdb para el n-puzzle se puede apreciar que no se logro hallar la respuesta para ninguna instancia ejecutada, en cambio para la heuristica manhattan este logro hayar 2 de las 5 instacias

10. Conclusiones

Hemos explorado el desempeño y los resultados obtenidos en durante el recorrido de espacios de estados haciendo uso de la Librería PSVN creada por la universidad de Alberta y analizamos los resultados de la aplicación de diversas heurísticas sobre rompecabezas famosos y con grandes factores de ramificación. Mediante el uso de Pattern Data Bases se pudo crear abstracciones a los distintos problemas presentados y generar funciones de estimación de costo adecuados tales que mejoran los recorridos en busca de los estados objetivos.

Junto a este informe se anexa el código fuente junto con su documentación de uso así como un conjunto de archivos csv con resultados de ejecución en las computadoras de prueba.

En definitiva, la inteligencia artificial es un recurso valioso para la resolucion de problemas. Se observo como buenas heuristicas son capaces de acercarnos mas rapidamente a nuestro objetivo y gastando menos recursos. Las posibles aplicaciones de esta tecnologia estan por doquier, desde enrutamiento de paquetes de datos en redes hasta el monitoreo y planificacion de vuelos para aerolineas. Aunque este es solo la punta del icebearg de las implicaciones de la IA ya podemos observar los beneficios que esta trae consigo.