Informe Electivo de Heurística
Tarea 2 - Klotsky

Javier Mahana
Joaquín Herrera

09 Junio 2023

Abstract

En este documento se aborda la implementación del juego de mesa *Klotsky* en C++. El tablero de juego y sus piezas fueron representadas con bitsets, debido a su rapidez de ejecución y su eficiencia en el uso de memoria. Luego se implementaron los algoritmos BFS, A* y IDA* para intentar resolver el juego. Finalmente se hizo una comparativa de estadísticas entre los tiempos que demoraron los distintos algoritmos utilizados para encontrar cuál es óptimo en *Klotsky*.

1 Introducción

Klotsky es un juego de puzles deslizantes de un jugador, en el que piezas de distintos tamaños se colocan en una tablero de 5 filas y 4 columnas(Fig.1). Al comienzo del juego hay solo dos casillas libres, el objetivo es lograr que la pieza de 2x2 llegue a la parte central inferior del tablero.

Reglas

- Las piezas deben ser deslizadas.
- El juego termina cuando la pieza de 2x2 llega al espacio final de 2x2.

Piezas

- 4 piezas de 1x1.
- 4 piezas de 2x1.
- 2 piezas de 1x2.
- 1 pieza de 2x2.

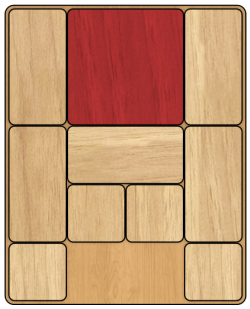

Figure 1: Tablero inicial de *Klotsky*

2 Implementación

Se utilizó C++ para la implementación de Klotsky. Debido a que es un juego de puzle deslizantes que usa casillas se decidió usar una abstracción de bitsets, de tamaño 25. Se crearon bitsets para cada pieza, para la posición final del juego y un *boardmask* para mostrar los espacios válidos del bitset. Con la abstracción del juego lista se dio paso a su implementación, añadiendo funciones que permitieran realizar movidas, revisar movidas legales y revisar la condición de victoria. Por la simplicidad del juego se crearon solo tres clases:

- **Klotsky**: Contiene los bitboards, las funciones de movimientos de juego y los algoritmos de búsqueda.
- **Node Klotsky**: Se usa para guardar nodos(estados del tablero) guardando su costo inicial y su valor heurístico.
- **Main**: Instancia la clase Klotsy, busca la solución con los distintos algoritmos e imprime datos sobre el desempeño de cada uno.

3 Algoritmos de búsqueda

- **Breadth First Search**: Éste algoritmo se implementó usando pseudocódigo de forma directa en nuestra abstracción.
- **A*:** Éste algoritmo se uso con la heurística calculada que es explicada más adelante, fuera de eso el algoritmo se implementó usando pseudocódigo de forma directa en nuestra abstracción.

- **IDA***: Éste algoritmo se uso con la heurística calculada junto a textitDepth Limit Search. Para facilitar la búsqueda los nodos vecinos se ordenaron por la distancia **h** y un *hashmap* para omitir nodos ya encontrados.

4 Heurística

La heurística implementada, genera un calculo basado en tres criterios:

- Manhattan (**Admisible**): Calcula la distancia entre la pieza 2x2 y la meta (casilla de 2x2).
- Piezas que Bloquean (**Admisible**): Retorna la cantidad de piezas que bloquean la distancia de la pieza 2x2 hasta la meta.
- Piezas en la parte superior (**No Admisible**): Nos dimos cuenta que en las soluciones las piezas 2x1 se ubican en la parte superior del tablero, por eso esta heurística que calcula la distancia de estas piezas la parte superior. La arbitrariedad de éste criterio lo hace no admisible.

5 Resultados

El algoritmo IDA* no fue ejecutado hasta que encontrara el resultado debido a que el tiempo que se habría demorado en encontrar un resultado habría estado en la magnitud de horas/días, ya que por cada iteración el tiempo de búsqueda se incrementaba casi el doble. Y dado lo visto con los algoritmos BFS y A*, para que IDA* encuentre la solución necesitaría unas 80 iteraciones aproximadamente.

5.1 Sin Heurística

Algoritmo de búsqueda	Tamaño Solución	Nodos Visitados	Tiempo (segundos)
Breadth First search	90	10.531.657	281,65

5.2 Con Heurística Admisible

Algoritmo de búsqueda	Tamaño Solución	Nodos Visitados	Tiempo (segundos)
A*	90	8.936.584	297,45
IDA*	Nula ; iteración 13	1	417,86
IDA* (tabla transposición)	Nula ; iteración 25	14014	149,76

5.3 Con Heurística No Admisible

Algoritmo de búsqueda	Tamaño Solución	Nodos Visitados	Tiempo (segundos)
A*	90	6.978.053	235,357
IDA*	Nula ; iteración 13	1	271,759
IDA* (tabla transposición)	Nula ; iteración 21	7884	32,70

6 Análisis

Por los datos obtenidos se logra concluir que A* es el algoritmo óptimo para Klotsky, además la heurística no admisible dio buenos resultados, le sigue BFS que revisa más nodos, pero es más rápida. Por otro lado, IDA* es extremadamente lenta en éste caso y no apta para la configuración de tablero de Klotsky utilizada.