Sistemas de Inteligencia Artificial
TP2 : Algoritmos de Búsqueda

Grupo 9

Marcos Abelenda, 55086
Santiago Manganaro Bello, 56239
Matias Perazzo, 55024
Agustín Ignacio Vázquez, 55354

Introducción

En la vida cotidiana se generan nuevos problemas diariamente que desafían las limitaciones del hombre y su capacidad intelectual u otros ya existentes de los cuales no se conoce su solución. En este trabajo vamos a generarlas para un problema conocido a partir de distintos criterios de búsqueda o algoritmos y comparar los resultados obtenidos. Al mismo tiempo se analizará distintas optimizaciones y cómo estas influyeron en la performance del motor de búsqueda para un algoritmo dado.

Problema

El juego a tratar es GridLock, el cual cuenta con un tablero y una disposición de fichas y el objetivo es mover la principal a la salida que estará bloqueada por otras fichas del tablero.

Instancias de GridLock

- Se pueden observar en el anexo.

Clasificación de casos

Al momento de clasificar los casos mencionados según dificultad surgieron distintos criterios posibles. ¿Se debería realizar según que tan complicado resulta el problema para el ser humano o que tan complejo es de resolver para la máquina? En caso de elegir el segundo, ¿que métricas nos indican dicha dificultad? Podría ser la altura de la solución, la cantidad de estados generados o el tiempo de ejecución. De las últimas mencionadas se les otorgó más peso a las dos últimas. Una vez elegidas las métricas, ¿el análisis se debería efectuar sobre cada algoritmo posible de búsqueda ponderando si es necesario o solo sobre los más óptimos? Se consideraron todos los algoritmos posibles para comprender la dificultad de cada caso.

Según los criterios mencionados se clasificó el caso “A” como el más fácil y el “B” como el más difícil.

Función de costo

Al tratarse del juego GridLock esta función representaría el desplazamiento de una pieza en una dirección. En vez de tratar desplazamientos de 1 solo casillero se consideraron de N casilleros, es decir, se puede mover una pieza de a 1 casillero o hasta colisionar con otra pieza o el borde del tablero.

Heurísticas

GridLockBasicHeuristic

Cuenta la distancia de la pieza principal a la salida. Como pueden haber piezas en el medio que no son consideradas por esta heurística siempre va a subestimar la cantidad de movimientos requeridos para ganar el juego ya que habría que mover estas piezas para poder llegar a la meta. Considerando lo último mencionado se puede deducir que es admisible.

GridLockMediumHeuristic

Calcula la distancia de la pieza principal a la salida y le suma la cantidad de piezas que la obstruyen para llegar al objetivo. Dado que no considera los movimientos que hay que realizar para liberarle el camino a la pieza principal siempre va a subestimar el costo final. Por lo tanto es admisible.

GridLockAdvancedHeuristic

Calcula la distancia de la pieza principal a la salida y le suma la cantidad de piezas que la obstruyen para llegar al objetivo. Además considera los movimientos que habría que hacer para liberar las piezas que obstruyen a la principal. Como las últimas son piezas verticales se podría ser optimista y considerar el desplazamiento de menor movimientos o pesimista y considerar el de mayor movimientos. Se probaron ambos casos y se obtuvieron mejores métricas siendo pesimista. Considerando esto último esta heurística no es admisible ya que al quedar una única ficha entre la pieza principal y la meta se haría la sugerencia de moverla hacia el lado de mayor movimientos.

GridLockProHeuristic

Utiliza la heurística anterior pero además considera si hay piezas que bloquean a las que están obstruyendo a la principal. Dado que utiliza la estrategia mencionada previamente no es admisible.

Métricas

A continuación se mencionan las métricas que van a ser mostradas por cada ejecución del proyecto.

- Estados generados
- Altura de la solución
- Número de nodos frontera
- Número de nodos expandidos
- Tiempo de ejecución
- Costo de la solución
- Repeticiones omitidas
- Movimientos elegidos para ganar
    - Muestra la cantidad de movimientos realizados
    - Muestra el costo de cada movimiento
    - Muestra el costo acumulado
    - Muestra el estado generado por ese movimiento

Observaciones

Las siguientes observaciones se obtuvieron a partir del archivo “Cuadro comparativo de algoritmos de búsqueda” que está adjunto en el trabajo.

- DFS vs BFS
    - El segundo pudo encontrar el camino de costo óptimo mientras que el otro no pudo.
    - Si bien ambos generan una cantidad de estados similares el primero expandió muchos menos nodos manteniendo una mayor cantidad de los mismos en la frontera.
    - Como BFS expande más nodos cuenta con una mayor cantidad de repeticiones omitidas.
    - BFS utiliza mayor memoria.

- Profundización Iterativa vs DFS vs BFS
    - El segundo cuenta con una mayor cantidad de nodos frontera.
    - El último cuenta con una gran proporción de nodos expandidos
    - El primero mantiene una proporción más similar de nodos expandidos y nodos frontera.
    - Mantiene las ventajas de DFS y BFS. Utiliza el primero cuando es posible obteniendo en el peor caso el segundo mencionado.

- Greedy Search vs Algoritmos desinformados
    - Disminución notable en la cantidad de estados generados
    - Mientras más precisa es la heurística se reducen aún más los estados generados y se obtiene una solución de menor costo.
    - La solución obtenida no necesariamente es la de mejor costo
    - Se obtiene una solución de forma rápida

- A* vs Greedy Search
    - El primero siempre obtiene la solución óptima. También tiene que analizar una mayor cantidad de estados para hallarla. A mejor heurística menor cantidad de estados generados.

- Heurísticas
    - De las cuatro planteadas se puede observar como influye la complejidad de cada una en la cantidad de estados generados y en el costo de la solución.
    - Optimistas vs Pesimistas
        - En el caso de las dos últimas heurísticas se planteó la posibilidad de utilizar un criterio pesimista u optimista. Se probaron ambos y se pudo observar que con el pesimista se obtuvieron mejores resultados.

- Clasificación de casos
    - En términos generales se puede observar una mayor cantidad de estados generados y tiempo de ejecución para el caso “B”, luego para el “C” y por último para el “A” (orden de dificultad respectivamente).

- Pre generación de reglas
    - Se compararon los tiempos de ejecución para cada algoritmo generando las reglas de antemano o generandolas para cada estado. En términos generales se pudo observar un incremento de un 30% a un 100% del tiempo de ejecución cuando se optaba por pre generar todas las reglas.

Optimizaciones

- Función de costo
    - Al poder desplazar la ficha N pasos se obtuvieron tiempos de ejecución de entre un 10% y un 20% menores y un árbol de solución de menor altura.

- Generación de reglas a partir de un estado
    - Al hacer esto en vez de generar todas las reglas posibles se redujeron tiempos de ejecución desde el 15% hasta el 50%.

- Profundización iterativa
    - En vez de generar el árbol desde cero cuando se expande la altura se realiza DFS a partir del árbol anterior.

- Matriz de bits para colisiones
    - En vez de recorrer todas las piezas para detectar colisiones se mantiene un tablero de bits indicando si la posición deseada está ocupada o no. El mismo se mantiene actualizado.

Conclusión

Al momento de hallar la solución para un problema existen distintos algoritmos que se pueden utilizar. El algoritmo a utilizar depende de las métricas priorizadas y del conocimiento o no de una estrategia de búsqueda para el problema en cuestión. Las heurísticas empleadas en los algoritmos informados influyen de forma notable sobre la solución hallada.

- Si se desea obtener una solución rápida en base a una estrategia entonces el algoritmo a utilizar es Greedy Search.

- Si se quiere tener la solución óptima usando una estrategia entonces la búsqueda será con A*.

- Si se desea una solución sin usar muchos recursos entonces se empleará DFS.

- Si el objetivo es obtener un resultado aceptable sin importar los recursos consumidos entonces se debe elegir BFS.

- Si se quiere una solución aceptable usando menos memoria que BFS entonces el candidato es Profundización iterativa.

- Una buena heurística influye notablemente en la búsqueda de un algoritmo informado