Análisis de la solución de *Chain Reaction*

Sistemas de Inteligencia Artificial
Instituto Tecnológico de Buenos Aires

Garrigó, Mariano 54393 — Raies, Tomás A. 56099 — Saqués, M. Alejo 56047

Resumen

Se ha implementado un Sistema de Producción para la resolución genérica de problemas de búsqueda, en base a una interfaz previamente consensuada con la Cátedra y el curso.

Dicho sistema provee una serie de mecanismos que afectan la manera en que se insertan y extraen nodos del conjunto de frontera, efectivamente alterando el comportamiento del algoritmo de búsqueda.

Asimismo, en lo que respecta al problema de *Chain Reaction*, se ha implementado una serie de heurísticas a los efectos de acelerar el hallazgo de una solución.

Palabras clave: Sistema de Producción, búsqueda desinformada, *breadth-first search*, *depth-first search*, *iterative deepening depth-first search*, *greedy search*, *A* algorithm*, *Chain reaction*, *Hamiltonian path*.

1. Búsquedas desinformadas

En esta sección se discutirán los resultados obtenidos utilizando búsquedas desinformadas.

1.1. *Depth-first search* y *Breadth-first search*

En las tablas adjuntadas, se puede observar claramente como evoluciona el problema de uso de memoria en BFS en comparación a DFS: para el primer caso, BFS expande una cantidad de nodos dos ordenes de magnitud mayor que DFS. Para el caso de $7X7$, mientras DFS sigue − con dificultades − hallando una solución, BFS provoca un error de falta de *heap* para asignar más memoria.

Sin embargo, DFS muestra sus deficiencias en el caso $8X8$, donde se terminó su ejecución por un *timeout* definido en 10 minutos. Por lo general, DFS provee una mejor *performance* de BFS en casos en los que existe más de una solución. Para el caso de *Chain Reaction*, la cantidad de soluciones está intrínsecamente asociada a la estructura del grafo obtenido a partir del tablero, pudiendo eventualmente ser única.

En casos a analizarse más adelante, se observará que el valor asignado al *timeout* sumamente generoso para el tiempo de ejecución que demandan las implementaciones que utilizan heurísticas.

1.2. *Iterative deepening DFS* (IDDFS)

El objetivo de la profundización iterativa es realizar una búsqueda en anchura sin el costo en memoria de un BFS tradicional. Por ende, lo que se busca es realizar una implementación de DFS limitada por un valor máximo de profundidad.

En el problema de *Chain Reaction*, el objetivo es encontrar un camino hamiltoniano sobre los nodos del tablero, utilizando las reglas que aplican al problema. Es decir, se quiere encontrar un camino que recorra los N vértices una

única vez, por ende, el nodo solución estará en la máxima profundidad posible.

Visto esto, para el caso de *Chain Reaction*, como siempre la solución estará en la máxima profundidad, IDDFS es ineficiente a la hora de encontrar el resultado. Efectivamente, la solución provista por IDDFS es equivalente a la de DFS sin límites de profundidad, más el *overhead* de haber ejecutado un DFS limitado por altura para cada una de las alturas no válidas.

2. Heurísticas

En esta sección, se discutirán las diferentes heurísticas implementadas, demostrando su admisibilidad.

2.1. Heurística sin poda

Esta heurística, a diferencia de las que se exhibirán más adelante, no realiza ningún tipo de poda en base a la situación del tablero que emana del estado a analizar. A continuación sigue la función:

$$h(E) = (Remaining(E)) - \frac{1}{Open(E)} \quad (1)$$

$$Remaining(E) = Total - Occupied(E) \quad (2)$$

Donde $Total$ es la cantidad total de casilleros no vacíos, $Open(E)$ es la cantidad de estados vecinos a E. Si $Open(E) == 0$ y $Remaining(E)! = 0$, se define $h(E) = +\infty$. El sistema de producción debería, luego, obviar insertar al nodo que contenga dicho estado en el conjunto de frontera.

2.1.1. Admisibilidad

Definiendo al costo de recorrer un eje en 1, el costo total de la solución es $N-1$, siendo N el número de casilleros no vacíos. Por ende, a lo sumo $Remaining(E) = N-1$ y $Open(E) = 4$, por lo que como máximo $h(E) = N - 1 - \frac{1}{4} \leq N - 1$, por ende, al nunca sobrestimar el costo hasta la solución, la heurística es admisible.

2.2. Heurística con poda direccional

El objetivo de esta y la siguiente heurística a discutir es eliminar estados que no pueden llevar a una solución dado que con la aplicación previa de alguna regla válida se ha llegado a un estado en el que algún casillero no visitado es inalcanzable. Por ende, dicho estado no puede formar parte de una solución.

La poda se realiza de la siguiente forma: dado el estado actual, para cada una de las direcciones válidas de movimiento (horizontal y vertical, centrándose en la última posición alcanzada), verificar si algún casillero libre es inalcanzable. Si esto se cumple, asignar $h(E) = +\infty$. Caso contrario, retornar el valor dictado por la heurística en 2.1.

De esta forma, se desestiman estados que no pueden llevar a una solución, efectivamente podando el espacio de búsqueda. Sin embargo, este proceso de poda considera que *es más probable* encontrar casilleros inalcanzables en las direcciones de movimiento dictadas por el problema, pero es posible que haya nodos inalcanzables en posiciones no analizadas por esta poda.

2.2.1. Admisibilidad

Esta heurística retorna el mismo valor que 2.1. si no se ha encontrado que algún casillero en las direcciones analizadas es inalcanzable. Si es inalcanzable, el estado analizado no forma parte de ninguna solución. Como la heurística en 2.1. es admisible, esta heurística lo es también.

2.3. Heurística con poda completa

En este caso, a partir del estado actual se analiza todo el tablero en búsqueda de casilleros inalcanzables, efectivamente eliminando todos los caminos que no llevan a una solución. Si no se encuentra un casillero inalcanzable, se retorna el mismo valor que en 2.1.

2.3.1. Admisibilidad

El análisis es idéntico al caso 2.2.

3. Búsquedas informadas

En esta sección se discutirán los resultados obtenidos con las búsquedas informadas (*Greedy* y *A**), utilizando las diferentes heurísticas mencionadas en la sección anterior.