Métodos de búsqueda no informados e informados

Gonzalo V. Castiglione, Alan E. Karpovsky, Martín Sturla
Estudiantes Instituto Tecnológico de Buenos Aires (ITBA)

20 de Marzo de 2012

Entrega Final - Informe

***Resumen*—El presente informe busca analizar y comparar distintas estrategias de búsqueda sobre un problema en particular (Skyscraper puzzle) haciendo uso de un motor de inferencias, como así también evaluar las mejoras obtenidas mediante la aplicación de heurísticas.**

***Palabras clave*—DFS, BFS, General Problem Solver, depth-first, breadth-first, search strategy, iterative DFS, A Star**

I. Introducción

El juego *Edificios* también conocido como *Skyscraper puzzle* es una variante del conocido *Sudoku* y consiste de una grilla cuadrada con números en su borde que representan las pistas sobre cuántos edificios se visualizan en esa dirección. El tablero, visto desde arriba, representa un espacio cubierto de edificios. Cada casillero debe ser completado con un dígito que va entre 1 y N, siendo N el tamaño de la grilla; haciendo que cada fila y cada columna contengan sólo una vez a cada dígito (como sucede en el Sudoku).

II. Estados del problema

A. Estado inicial

El estado inicial del problema es un tablero que contiene sólo las pistas en sus bordes (no contiene ningún edificio en la grilla). Para que el problema tenga solución éste tablero debe ser válido; es decir que no cualquier combinación de pistas sobre la visibilidad de edificios en esa dirección conducirán a un problema resoluble.

B. Estado final

El estado final del problema es un tablero con todos los casilleros completos (lleno de edificios) que cumpla con las reglas del juego citadas anteriormente.

III. Modelado del problema

El tablero del juego se modeló mediante la creación de la clase Board; la misma contiene una variable privada de tipo matriz de enteros que representa a la grilla propiamente dicha. Si un casillero de dicha matriz tiene un 0, significa que está vacío; si en cambio tiene un número $k \in [1,n]$ significa que hay situado un edificio con altura k.

Las pistas o restricciones del tablero (números en los bordes del mismo) se modelaron con un arreglo de cuatro vectores de enteros (TOP, BOTTOM, LEFT, RIGHT) que simplemente contienen la restricción en esa dirección. Cabe destacar que en los estados únicamente se guarda el tablero; guardar las restricciones sería redundante. Ver **Figura 1** en la sección **Anexo B**.

IV. Reglas

Para la resolución de este problema se han pensado dos conjuntos de reglas distintos. El primero, enunciado a continuación, fue descartado por su baja performance. Seguido de este se explicará el conjunto de reglas elegido y sus beneficios.

Dado un tablero de tamaño $n \times n$ podemos describir las reglas del problema en forma general como sigue:

Poner un edificio de altura x en la posición (i,j) del tablero, con $x \in [1,n]$.

Por lo que, que dado un tablero de $n \times n$, el total de reglas está dado por $n \times n \times n = n^3$ ya que por cada fila y por cada columna, se tienen n edificios distintos para colocar.

Ejemplos de reglas:

- Poner un edificio de altura 1 en la posición $(0,0)$
- Poner un edificio de altura 4 en la posición $(3,1)$
- $\dots$

Las reglas así definidas producen un factor de ramificación de n^3 lo que hace que, para tableros grandes (mayores a 5×5) se tarde un tiempo muy considerable (en el orden de las horas) en encontrar la solución. Para subsanar este inconveniente se puede plantear el problema enunciando con un conjunto de reglas distinto, descripto a continuación:

Poner un edificio de tamaño x en el próximo lugar disponible de izquierda a derecha y de arriba hacia abajo, con $x \in [1,n]$.

20 de Marzo, 2012.

En un tablero de $n \times n$ el conjunto de reglas quedaría definido de la siguiente forma:

- Poner un edificio de altura 1 en el próximo lugar disponible de izquierda a derecha y de arriba hacia abajo.
- ⋮
- Poner un edificio de altura n en el próximo lugar disponible de izquierda a derecha y de arriba hacia abajo.

Esto otorga un factor de ramificación igual a n siendo este muy inferior al del caso anterior.

Se decidió implementar ambos conjuntos de reglas y los resultados fueron comparados en las tablas de la sección **Anexo A**.

V. Costos

Debido a la naturaleza del probema, la aplciacion de todas las reglas tiene el mismo costo y éste es unitario. Es decir que la transición de un estado X a un estado Y, en nuestro caso, siempre cuesta 1.

VI. Algoritmos de búsqueda implementados

Hemos implementado seis algoritmos de búsqueda no informada diferentes. Los mismos son: DFS, BFS, IDFS, HIDFS, Greedy search y AStar *(A*)*.

VII. Heurísticas

La idea detrás de las heurísticas, en un problema de satisfacción de restricciones como el discutido, no es hallar el camino óptimo a la solución ya que previamente se conoce su profundidad y costo, sino descartar caminos que no conducirán al algoritmo hacia la solución. Esto se logra, para un determinado nodo, asignándole el valor *infinito* a la función $h(n)$ de los sucesores irresolubles de éste.

Al comenzar el estudio del problema, se pensaron heurísticas sólo basadas en las restricciones o pistas del tablero. Las mismas son:

A. Heurística 1

Si una restricción indica el número 1, esa fila o columna deberá comenzar con la altura maxima (n). Además si la restricción opuesta es un 2, se coloca $n-1$ al lado de este.

B. Heurística 2

Dado un tablero de dimensión n, si una restricción indica algún número distinto de 1 , esa fila o columna trivialmente no podrá comenzar con (n).

Si se considera el caso general, si una restricción indica algún número a mayor o igual que 1 , esa fila o columna deberá tener un edificio con la altura máxima (n) a, al menos, a casilleros de distancia. Si generalizamos este resultado, podemos obtener el siguiente resultado útil:

Dado una fila o columna con restricción n, para cada edificio de altura k tenemos $dist(k) >= a + k - n - 1$*, siendo n la dimensión del tablero y a la restricción a validar.*

En otras palabras, el edificio de altura máxima debe estar a n casilleros de distancia, el de altura $n-1$ a distancia $n-1$ o mayor, y así sucesivamente. Esto implica que los edificios deben estar ordenados de menor a mayor si se da esta situación.

Una vez estudiado el problema con un poco mas de detalle, se decidió explotar la *naturaleza Sudoku* del problema y así definir heurísticas como sigue:

C. Heurística 3

Si un estado conducirá de manera seguro a un tablero no resoluble, asignar el valor *infinito* a ese estado.

D. Heurística 4

Si un estado conducirá de manera seguro a un tablero no resoluble, asignar el valor *infinito* a ese estado.

E. Heurística 4

Para un determinado estado, contar la cantidad de casilleros libres de su tablero. Mientras menos casilleros libres tenga, mejor será puntuado dicho estado.

F. Heurística 5

Evaluar la *certeza* de un determinado estado. Esto se hace evaluando la cantidad de posiciones seguras y posiciones casi-seguras que tiene el tablero, es decir, la cantidad de lugares donde de antemano ya se sabe que allí sólo podrá ir un sólo número o, en su defecto, podrán ir dos números distintos, generando alternativas posibles para ese casillero.

De esta manera, utilizando las reglas similares a las del *Sudoku* en las que un mismo número no puede repetirse en las filas y columnas, se logra puntuar mejor a los tableros que tienen más lugares seguros ante tableros que tienen más posibilidades para sus casilleros.

Aunque resulte poco intuitivo, la implementación de esta última heurística redujo considerablemente la performance del algoritmo. Debido a esto se decidió optar por utilizar la heurística de asignarle *infinito* a los tableros irresolubles, en combinación con la de calcular el número de casilleros libres.

VIII. Recorridos

Se han implementado tres tipos de recorridos distintos sobre el tablero, en busca de comparar la performance obtenida con cada uno de ellos.

- **Recorrido secuencial:** Se avanza de izquierda a derecha, desde arriba hacia abajo.

- **Recorrido en espiral:** Se avanza desde el centro en forma de espiral o desde una esquina hacia al centro, de forma análoga.
- **Recorrido MVR:** Se avanza hacia el casillero con menor cantidad de valores legales.

IX. Resultados

A través de las distintas pruebas realizadas se observó que el algoritmo con mejor desempeño cualquiera sea la dimensión del tablero, es el DFS. Ver tablas en la sección **Anexo A**. A su vez, entendiendo que para este trabajo los métodos debían ser desinformados y no podíamos aplicar heurísticas, resultó interesante comparar los tiempos obtenidos en el punto anterior con los tiempos obtenidos dándole un órden particular a las reglas antes de que el *General Problem Solver* comience a resolver. Básicamente lo que se hizo fue preprocesar las pistas o restricciones del tablero y ordenar el conjunto de reglas por única vez. Los tiempos obtenidos preprocesando las reglas fueron superiores a los anteriores pero aún muy inferiores al obtenido por el conjunto reducido de reglas.

Observando las tablas se puede apreciar que el algoritmo con mayor performance fue el DFS.

X. Conclusión

ES notable destacar la cantidad tiempo requerido y uso de recursos para resolver pequeños problemas con métodos no informados. Luego de ver los resultados obtenidos para pequeños tableros, se necesitó hacer uso de un pre-ordenamiento de las reglas ya que la resolucion del problema tomaba un tiempo en el orden de las horas para tableros tan simples como ser 4×4.

Resultó muy interesante también la forma de resolver problemas mediante la aplicación del mismo conjunto reglas en forma repetida y variando el orden en cual se agregan los nuevos estados al conjunto de no explorados.

No sorprende que el DFS haya sido el algoritmo con mejor desempeño, dado que al tener un árbol con profundidad acotada, el DFS evita los inconvenientes más usuales como pasarse de la profundidad de la solución óptima o entrar en ciclos del grafo (la cota de profundidad le es muy útil). El IDFS tiene una performace menor dado que como el problema tiene profundidad acotada de n, el IDFS no es más que un DFS con el overhead extra de tener que iterar por las $n-1$ profundidades anteriores. El BFS resultó ser menos óptimo dado que son demasiados los estados que debe explorar para llegar a la solución en problemas exponenciales como es el de los *Skycrapers*.

Referencias

[1] Stuart Russell and Peter Norvig (December 2009), *Artificial Intelligence: A Modern Approach. (Third edition)*, Prentice Hall; USA ©.

Anexo A: Tablas

*Nota: **N/A** significa que el algoritmo no encontró la respuesta en un tiempo razonable. Algunos tableros han sido corridos durante horas y otros durante varios minutos.*

A. DFS

	Conj. reglas 1 (estándar)	Conj. reglas 2 (reducido)
Tablero 1 (3×3)		
Tiempo	245ms	3ms
Nodos frontera	82	4
Nodos expandidos	993	10
Estados generados	1076	15
Profundidad de la solución	9	9
Tablero 2 (4×4)		
Tiempo	*N/A*	9ms
Nodos frontera	290	3
Nodos expandidos	91236	93
Estados generados	91527	97
Profundidad de la solución	11 (cantidad edificios puestos al corte)	16
Tablero 3 (5×5)		
Tiempo	*N/A*	331ms
Nodos frontera	935	7
Nodos expandidos	65729	901
Estados generados	66665	909
Profundidad de la solución	17 (cantidad edificios puestos al corte)	25

TABLE I
Comparación de tiempos para DFS con distintos sets de reglas

B. BFS

	Conj. reglas 1 (estándar)	Conj. reglas 2 (reducido)
Tablero 1 (3×3)		
Tiempo	3 sec 793 ms	11ms
Nodos frontera	0	0
Nodos expandidos	7132	16
Estados generados	7133	17
Profundidad de la solución	9	9
Tablero 2 (4×4)		
Tiempo	*N/A*	58ms
Nodos frontera	52836	1
Nodos expandidos	4132	111
Estados generados	57522	113
Profundidad de la solución	3 (cantidad edificios puestos al corte)	16
Tablero 3 (5×5)		
Tiempo	*N/A*	519ms
Nodos frontera	152016	0
Nodos expandidos	3488	1889
Estados generados	155505	1890
Profundidad de la solución	2 (cantidad edificios puestos al corte)	25

TABLE II
COMPARACIÓN DE TIEMPOS PARA BFS CON DISTINTOS SETS DE REGLAS

C. IDFS

	Conj. reglas 1 (estándar)	Conj. reglas 2 (reducido)
Tablero 1 (3×3)		
Tiempo	802ms	21ms
Nodos frontera	82	4
Nodos expandidos	2583	57
Estados generados	2666	62
Profundidad de la solución	9	9
Tablero 2 (4×4)		
Tiempo	*N/A*	68ms
Nodos frontera	297	3
Nodos expandidos	28979	833
Estados generados	29777	837
Profundidad de la solución	10 (cantidad edificios puestos al corte)	16
Tablero 3 (5×5)		
Tiempo	*N/A*	1sec 643ms
Nodos frontera	954	7
Nodos expandidos	25571	6528
Estados generados	26526	6536
Profundidad de la solución	19 (cantidad edificios puestos al corte)	25

TABLE III
COMPARACIÓN DE TIEMPOS PARA IDFS CON DISTINTOS SETS DE REGLAS

Anexo B: Imágenes

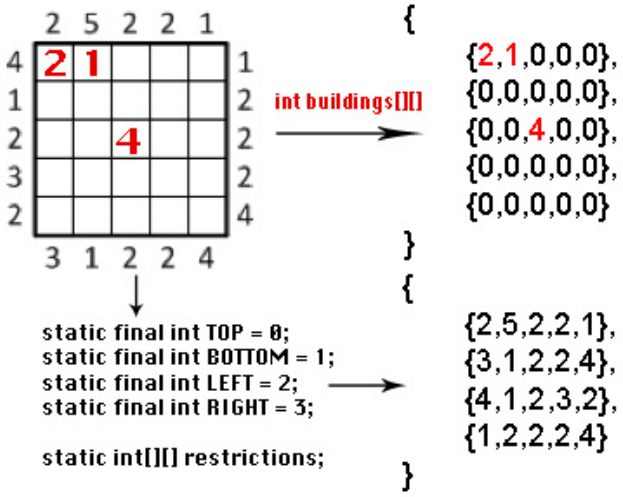

Figura 1: Modelado del problema