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. El tablero contiene además algunas variables necesarias para incrementar la performance de ciertas heurísticas explicadas adelante. Entre dichas variables se encuentra las coordenadas del último edificio puesto.

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 $(3, 1)$
- ...

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 para estrategias de búsqueda no informadas) en encontrar la solución. Además este conjunto de rueglas tiene el problema de ser mayormente conmutativas. Esto crea muchos estados redundantes.

20 de Marzo, 2012.

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 según el recorrido utilizado, con $x \in [1, n]$.

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 según el recorrido usado.
- $\vdots$
- Poner un edificio de altura n en el próximo lugar disponible según el recorrido usado.

Esto otorga un factor de ramificación igual a n siendo este muy inferior al del caso anterior. El próximo casillero a llenar está indicado por el recorrido. Más adelante se enumerarán los recorridos y sus respectivas propiedades, ventajas y desventajas.

Para la primer etapa se decidió implementar ambos conjuntos de reglas y los resultados fueron comparados en las tablas de la sección **Anexo A**. Para la segunda etapa del trabajo se decidió utilizar únicamente el segundo conjunto de reglas, utilizando distintos recorridos y comparándolos (reference needed).

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

Por cómo se ha planteado el modelado del problema, tiene poco sentido hablar de una función heurística que estime el costo de un cierto nodo a la solución. Esto se debe a que la profundidad de la solución es sabida de antemano. Sin embargo, es válido plantear que si un nodo no tiene solución, es debido a que ninguna asignación de valores de las variables o casilleros aún incompletos satisface las restricciones del problema. Por lo tanto, se decidió asignar a la función heurística un costo unitario por cada casillero aún no asignado si el tablero aún parece tener solución, e *infinito* si no se puede alcanzar la solución. Los nodos marcados con un valor de h *infinito* permanecerán en la frontera y nunca serán explotados a menos que el problema no tenga solución.

Se centró el estudio de los tableros sin soluciones en base a las restricciones dadas por los números fuera del tablero, numeradas a continuación.

A. Propiedad 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. Propiedad 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.

Estas dos heurísticas previas se combinaron con las restricciones de *naturaleza Sudoku* del problema, es decir la restricción *alldif* que existe en cada fila y columna -*todos los valores deben ser distintos*- para intentar maximizar el hallazgo de nodos irresolubles.

C. Valor heurístico de un estado

Si un estado no conducirá de ninguna manera a una solución, asignar el valor *infinito* a ese estado.

Durante el desarrollo del trabajo se analizaron distintos valores heurísticos que se le puede asignar a aquellos estados que no han sido identificados como irresolubles. A continuación se enumerarán las opciones:

C.1 Costo simple

Consiste en asignar simplemente la cantidad de casilleros aún no completados como valor de h. Es fácil ver que esta asignación favorece el explotado de nodos con mayor profundidad, lo cual dio mejores resultados que sus contrapartes. Esta función heurística es trivialmente admisible dado que:

- Si un estado es soluble y es considerado soluble, h es exactamente la longitud del camino más corto a una solución.
- Si un estado es irresoluble y no se lo ha identificado como tal, h es trivialmente una subestimación de la longitud del camino más corto a la solucion.

- Si un estado es irresoluble y se lo ha identificado como tal, h es teóricamente la longitud del camino a la solución, infinito, pero en la práctica es una subestimación debido a limitaciones del tamaño de almacenamiento.

C.2 Análisis de la restrictividad de un estado

Evaluar la *restrictividad* de un determinado estado según la **selección de valores** (y no la de variables, que será discutida más adelante). Este es un índice que se crea calculando la cantidad de posibles números que se pueden colocar en cada casillero aún incompleto, intentando ser normalizado según la cantidad de casilleros ya completos, dado que tableros más completos son trivialmente más restrictivos que tableros más vacíos. Por un lado, se investigó la penalización de tableros poco restrictivos para favorecer a estados con menor ramificación y sucesores. Por otro lado se investigó la penalización de tableros más restrictivos, con el pretexto de que mucha restrictividad implica poca probabilidad de hallar una solución, sobre todo si aún hay muchos casilleros para completar.

Para penalizar dichos tableros, sea cual sea el criterio, se incrementó el valor de h por arriba de la cantidad de casilleros sin llenar. Esto hace que la heurística sea trivialmente no admisible. Sin embargo, la admisibilidad de una heurística garantiza en el algoritmo $A*$ la optimalidad. Dado que en este problema la solución tiene siempre la misma profundiad, no existe tal cosa como una solución más óptima que otra. En vista de esto se decidió implementar dicha heuristica y comparar los resultados.

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 (mayor restrictividad). Nótese que se apunta a mayor restrictividad en cuanto a la **selección de variables o casilleros**. Cabe destacar que esto no entra en conflicto las heurísticas recién planteadas. Por ejemplo, se podría penalizar con heurísticas tableros muy restrictivos, y aún aplicar este recorrido. En esencia lo que se hace en cada paso es buscar el casillero con menor cantidad de posibles asignaciones de valores, generar un sucesor con cada uno de estos, pero explotar primero el que hace que el resto de los casilleros tenga la menor restrictividad posible. Esto se denomina *variable selection fail-first, value selection fail-last*. Para mayores referencias se puede consultar el libro de Russel y Norvig.

IX. Resultados

A. Algoritmos

A través de las distintas pruebas realizadas se observó que para las búsquedas no informadas, el algoritmo con mejor desempeño cualquiera sea la dimensión del tablero, fue el DFS. Ver tablas en la sección **Anexo A**.

Para las búsquedas informadas, ambos algoritmos tuvieron resultados similares para tableros de menor dimensión. Para tableros de mayor dimensión, el algoritmo $A*$ tuvo mejores resultados. (reference! Explicacion?)

B. Comparación de heurísticas

La implementación de la última heurística, basada en restrictividad, modificó la performance del algoritmo, pero irregularmente. Como muestran los resultados (reference! y corregir lo siguiente...) premiar o penalizar estados con mayor restrictividad alteró la performance positiva y negativamente, con magnitudes variantes, dependiendo del tablero.

Por otro lado, la identificación de estados irresolubles redujo considerablemente la cantidad de nodos generados, así como también el tiempo de ejecución, de una manera más regular y contundente. (reference!)

C. Comparación de recorridos

Los distintos recorridos tuvieron distintos resultados. Los recorridos estáticos varían considerablemente su desempeño en diferentes tableros. El recorrido *MVR* fue el que obtuvo los mejores resultados e incluso fue capaz de resolver en tiempos aceptables tableros de mayores dimensiones. Sin embargo, cabe destacar que en ciertos tableros a pesar de que el recorrido *MVR* generó menos estados que, por ejemplo, el *secuencial*, el tiempo de ejecución fue mayor, algo previsible debido al costo de tener que calcular en cada estado cuál es el casillero con mayor restrictividad.

En vista del desmepeño superior del *MVR*, se concluyó que es, en promedio, más eficiente llenar los casilleros en orden de mayor a menor restrictividad. Por lo tanto se conllevó a plantear que la irregularidad de los otros recorridos se debe a la irregularidad en la restrictividad de los tableros. Un tablero cuya primer fila puede ser determinada a través de las restricciones (por ejemplo, tener una n como restricción) hará que el recorrido secuencial complete los casilleros más restrictivos primero, mientras que los espirales no.

El recorrido espiral comenzando por el centro hacia afuera obtuvo regularmente los peores resultados. Este fenómeno no se relacionó con la restrictividad, que es irregular, sino con la naturaleza en la que llena el tablero. Se puede apreciar gráficamente que el espiral es el que más se demora en completar las filas o columnas, por lo cual muchas de las podas por alturas no se pueden efectuar hasta profundidades muy grandes, lo cual lleva a un árbol de estados más ancho y con mayor cantidad de retroceso o *backtracking*.

X. Conclusión

ES notable destacar la cantidad tiempo requerido y uso de recursos para resolver pequeños problemas con métodos no informados, sobre todo si existen reglas conmutativas. En particular para la resolución de problemas de satisafcción de restricciones, 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*.

Para intentar llegar a una solución generando menos nodos y en una menor cantidad de tiempo utilizando búsquedas informadas, resultó esencial identificar propiedades de las restricciones del problema para descartar la mayor cantidad de valores posibles, intentando de identificar a los tableros irresolubles como tales lo antes (menor profundidad) posible.

Resultó crucial elegir un recorrido o criterio de selección de variables o casilleros para evitar el problema de la conmutatividad de reglas. Los recorridos estáticos varían su eficieincia en distintos tableros, y no cuentan con la regularidad del *MVR*, pero tienen la ventaja de no tener que calcular en cada estado cuál es la próxima variable a completar. Sería interesante quizás analizar una correlación entre las restricciones del tablero y la densidad de restrictividad del tablero, para entonces poder elegir recorrido estático más adecuado en función de estas últimas.

El recorrido óptimo resultó ser el de *minimum remaining values*, que intenta hacer *fallar* lo antes posible al tablero (es decir, identificar si el estado es irresoluble).

En cuanto al manejo de restrictividad en las funciones heurísticas, no se identificó una clara correlación entre favorecer estados de mayor o menor restricción de las variables no asignadas con la cantidad de nodos generados o el tiempo de ejecución. Por lo que, en vista de los resultados, se identificó que se debe apuntar a mayor restrictividad en la selección de variables, pero no hubo una clara conclusión sobre la selección de valores. (references, algo de greedy vs a* faltaria!)

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

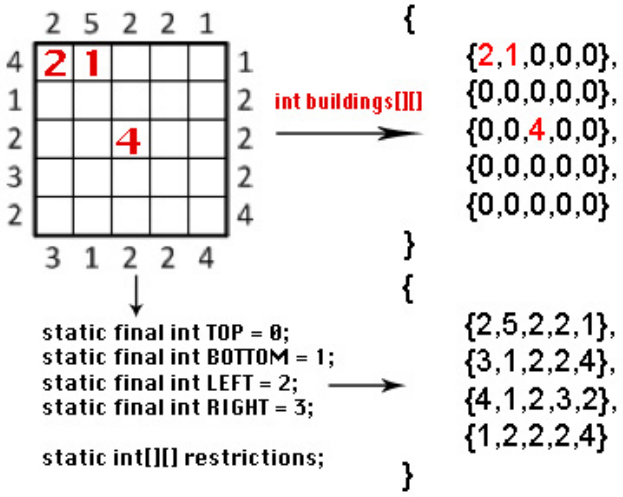

Figura 1: Modelado del problema