SudokuHex

Búsqueda heurística

José Ignacio Carmona Villegas

ETSIIT, Universidad de Granada, Granada, España
joseicv@correo.ugr.es

Abstracto. Obtener la solución de un sudoku (puzle combinatorio de permutaciones de números de una cifra en base decimal) no es algo trivial. Al incrementar el conjunto de números a 16 (números de una cifra en base hexadecimal), el número de posibles combinaciones se disparan, lo cual obliga a utilizar heurísticas para reducir el espacio de búsqueda de soluciones. El uso de técnicas de búsqueda sistemática como backtracking en combinación con heurísticas convierte este problema en tratable, y permite garantizar la optimalidad sin sacrificar la completitud.

Palabras clave. Sudoku, Backtracking, Satisfacción de Restricciones, Propagación de Restricciones, Heurística.

1 Introducción

1.1 Definición del problema

Crear un agente en Java (jre7) que resuelva cualquier Sudoku Hexadecimal (SudokuHex) dado. Aunque no se requiere, pues siempre se proporcionarán SudokuHex con solución, sería conveniente que indicase que no se ha encontrado solución. Como límite de tratabilidad se establece la resolución de un grupo de 60 puzles desconocidos en menos de 180 segundos (resolución de 0.5 puzles por segundo de media).

1.2 Definición de un SudokuHex

Un SudokuHex es un puzle combinatorio de permutaciones de números de una cifra en base hexadecimal. El puzle consiste en un tablero de 256 casillas, organizado de forma cuadrada, con 16 filas y 16 columnas. Además, se distinguen 16 cajas de 16 casillas, organizadas como un cuadrado de 4x4. En la *figura 1* se observa con más claridad el puzle descrito.

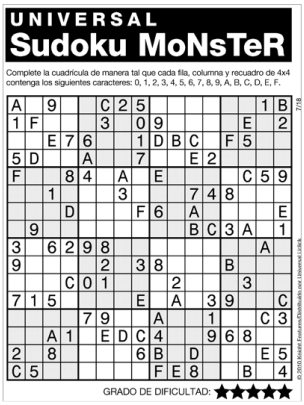

Figura 1. SudokuHex sin resolver.

1.3 Restricciones de un SudokuHex

Las restricciones impuestas para considerar el puzle resuelto son:

- Todos los valores de las casillas pertenecientes a una misma fila deben ser distintos.
- Todos los valores de las casillas pertenecientes a una misma columna deben ser distintos.
- Todos los valores de las casillas pertenecientes a una misma caja deben ser distintos.
- Todas las casillas deben tener asignado un valor.

Una vez estas restricciones hayan sido satisfechas, se considerará que el SudokuHex ha sido resuelto. Es importante considerar que un puzle de este tipo no tiene por qué tener solución.

2 Especificación de estrategias y heurísticas utilizadas

Estas estrategias se han usado sobre un esqueleto basado en Backtracking, que es una búsqueda exahustiva sin información primero en profundidad, de forma que se pueda garantizar la completitud del resultado final.

2.1 Propagación de restricciones (Constraint Propagation)

Utilizando las restricciones definidas en el apartado 1.3, se puede reducir el espacio de búsqueda mediante la técnica de propagación de restricciones. Esto consiste en tener en cuenta las restricciones del problema al realizar cualquier acción. En este problema, la operación básica es la eliminación, pues la asignación se puede hacer en función de la eliminación, así que sólo en la eliminación se propagarán las restricciones.

Al propagar las restricciones, pasamos de un nodo padre, a un nodo sucesor que está mucho más profundo en el árbol de búsqueda que el nodo sucesor que Backtracking tomaría por sí sólo. Esto es así, porque al tomar una decisión, backtracking no tiene en cuenta sus consecuencias (con respecto a las restricciones) y más tarde tomará decisiones que podrían haber sido obviadas debido a las restricciones del problema.

2.2 Variable más restringida (MRV o Most Restrained Variable)

Esta heurística consiste en escoger la variable más restringida, esto es, al escoger qué cuadrado considerar primero (orden de generación de sucesores de un nodo en backtracking), se toma el cuadrado que menos valores posibles puede tomar. La motivación de esta heurística es minimizar el factor de ramaje.

2.3 Comprobación de consistencia (Consistency Check)

Esta heurística consiste en comprobar la consistencia de una rama, concretamente, consiste en comprobar que el número de variables (cuadrados) no sea mayor al número de valores (valores hexadecimales posibles) que esas variables pueden tomar. Si hay más variables que posibles valores para esas variables, se puede asegurar que tarde o temprano se llegará a un estado inconsistente, pues no se podrá asignar un valor a una o más variables. El objetivo de esta heurística es podar, tan pronto como sea posible, ramas que conducen a estados inconsistentes.

2.4 Valor que menos restringe (LCV o Least Constraining Value)

Esta heurística consiste en, una vez fijada la variable, escoger primero (orden de generación de sucesores de un nodo en backtracking) de entre sus valores posibles a aquellos que menos restringen a otras variables. Haciendo esto se evalúan primero las ramas que llevarán a situaciones menos restringidas, y permitirán mayor libertad a la hora de generar posibles soluciones válidas. Las ramas correspondientes a situaciones sobrerrestringidas quedan pospuestas, no se podan.

3 Especificación y comparativa de versiones

3.1 Especificación de versiones

Version	*EDs*	*Orden de sucesores*	*Poda*
v1.0	HashMaps y TreeSet	MRV	-
v1.1	HashMaps y TreeSet	MRV	Comprobación de Consistencia
v2.0	Arrays	MRV	-
v2.1	Arrays	MRV	Comprobación de Consistencia
v2.2	Arrays	-	Comprobación de Consistencia
v2.3	Arrays	MRV + LCV	-
v2.4	Arrays	MRV + LCV	Comprobación de Consistencia

(*) Todas las versiones implementan Backtracking y Propagación de Restricciones.

3.1 Comparativa de versiones

Las primeras versiones (v1.0 y v1.1) se realizaron usando estructuras de datos que resultaron ser muy ineficientes. A partir de la v2.0, se cambian las estructuras de datos (por otras menos legibles, sin embargo, más eficientes) y los tiempos se redujeron notablemente. En la *figura 2* se puede apreciar el tiempo medio de resolución de un sudoku para la batería completa de 181 sudokus.

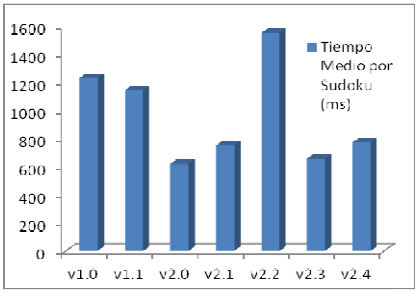

Figura 2. Tiempo medio por versión.

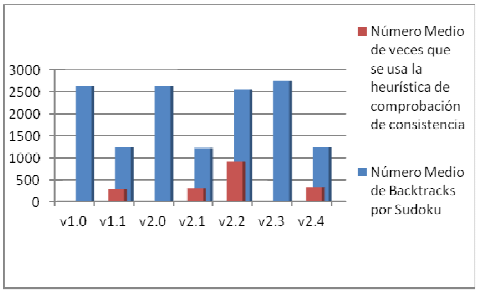

Figura 3. Comparativa Backtracs vs Uso heuristica.

Con las nuevas estructuras de datos, el tiempo de resolución medio por sudoku se reduce notablemente. El siguiente paso consiste en la combinación y prueba de distintas heurísticas para mejorar aún más el agente. Debido a lo mucho que oscilan las versiones 2.0 y 2.1 (la 2.2 fue una prueba cuyos resultados dejaban mucho que desear), y tras analizar los sudokus complejos que causaban este efecto, se incluye la heurística LCV en las versiones posteriores (2.3 y 2.4). Es interesante notar (*figura 3*) que la heurística de comprobación de consistencia reduce a la mitad la cantidad de backtracks que se realizan.

Finalmente, para poder comparar adecuadamente (pues la batería objetivo está formada por sudokus complejos) se realizan nuevas pruebas contra el subconjunto de 120 sudokus difíciles. Debido a que hay mucha dispersión en los tiempos, se escoge un estadístico más robusto, la mediana. Así, se desestiman las versiones 2.1 y 2.4.

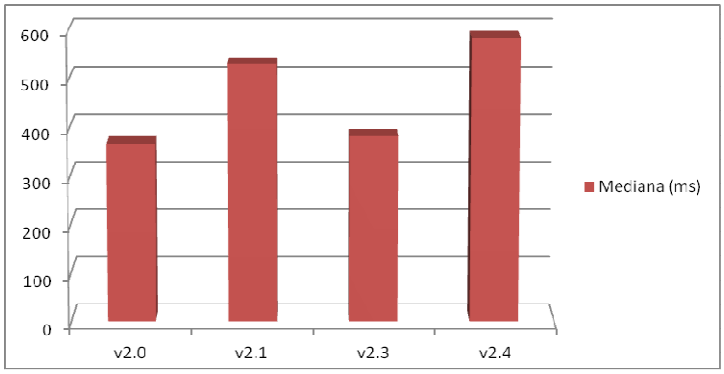

Figura 4. Tiempo mediano por versión.

	v2.0	***v2.1***	***v2.3***	***v2.4***
Media (ms)	827	1011	852	964
Mediana (ms)	367	531	383	586
Mejor tiempo (ms)	15	32	16	15
Peor tiempo (ms)	7500	9579	15797	7297
Número de sudokus resueltos en menos de 100 ms	17	14	16	11
Número de sudokus resueltos en menos de 200 ms	37	25	37	20
Número de sudokus resueltos en menos de 500 ms	75	57	71	54
Número de sudokus resueltos en más de 1000 ms	27	34	27	35
Número de sudokus resueltos en más de 1500 ms	16	25	16	20
Número de sudokus resueltos en más de 2000 ms	12	18	**7**	16
Número de sudokus resueltos en más de 4000 ms	6	4	5	4

Al estudiar la tabla anterior, se aprecia que el número de sudokus resueltos en más de 2000 ms es significativamente menor que con el resto, aún a pesar de que el peor tiempo es mucho mayor. En general, la versión 2.3 es la que mejor se comporta con los sudokus complejos, aún cuando incrementa los tiempos de resolución de los medios/fáciles y tiene un pico tan grande.

4 Conclusión

Debido a las razones ya expuestas en el apartado anterior, se escoge como versión ***final*** la ***versión 2.3***. Es además la que mejor se comporta con la batería objetivo, gracias a que Fernando Berzal tuvo la amabilidad de comprobarlo.

Es importante notar que las heurísticas se obtienen de la experiencia, y por ejemplo tras analizar los resultados en las diferentes versiones, MRV se comporta muy bien con casi todos los tipos de puzles, LCV simplifica mucho el proceso con los más complejos, pero complica el resto. La heurística de Comprobación de Consistencia, reduce mucho las cantidades de backtracks, lo cual no se comporta muy bien con este problema, debido a su reducido tamaño, pero con un sudoku de mayor orden seguramente sería la heurística más dominante.

Es posible mejorar la implementación pasando el proceso recursivo a iterativo (eliminando así la sobrecarga que impone colocar las llamadas recursivas en la pila), pero no se ha hecho, debido al tipo de heurísticas usadas (de generación y orden de evaluación sucesores, y de poda) que influyen en el iterador que tendría el árbol de búsqueda correspondiente al problema. Sería

además interesante comprobar la eficacia de otras técnicas menos generales y más específicas al problema en cuestión, como pares desnudos o técnicas aritméticas (en hexadecimal) en combinación con estas, así como de forma independiente.

5 Referencias

1. Stuart Russell y Peter Norvig. *Artificial Intelligence: A modern Approach*. Pearson, 3º edición, 2010.
2. Peter Norvig. *Solving every Sudoku puzzle*. http://norvig.com/sudoku.html
3. Dottan Asselmann. *Sudoku as a CSP*. http://www.codeproject.com/Articles/34403/Sudoku-as-a-CSP