Comparativa entre los algoritmos para resolver Sudoku

Introducción

El problema de decisión de Sudoku ya es conocido como un problema NP-Completo [Yato03] y que está muy relacionado con el problema de la reconstrucción de cuadrados latinos que también es un problema NP-Completo [Colbourn84]. Eso demuestra que es difícil encontrar un algoritmo que resuelva de manera eficiente y se han encontrado distintas propiedades que ayuda a mejorar como heurística a los algoritmos de búsqueda que tienen la complejidad exponencial.

Descripción del Problema

El problema clásico de Sudoku consiste en completar una matriz, generalmente de tamaño 9 x 9, con números del 1 al 9, respetando las tres restricciones del problema:

- Cada fila debe contener todos los números del 1 al 9.
- Cada columna debe contener todos los números del 1 al 9.
- Cada caja de tamaño 3 x 3 debe contener todos los números del 1 al 9.

Las cajas están ubicadas de manera ordenada de tal forma que no se solapen entre sí y que ocupen toda la matriz. Como cada fila, columna y caja deben contener todos los números del 1 al 9, no pueden repetirse ningún número dentro de ellas.

A cada elemento de la matriz son llamados celdas e inicialmente puede contener algunos números que no pueden ser cambiados ni reubicados. Las celdas que no contienen los números pueden ser asignados cualquier número respetando las reglas mencionadas anteriormente. Cuando se llegue a completar todas las celdas de la matriz con números, se dice que el problema está resuelto.

Una generalización propuesta de este problema está definido de la siguiente manera:
Definimos el rank R como un número que determina el tamaño del problema. El problema de Sudoku de rank R consiste en completar una matriz de R^2 x R^2 con los números del 1 al R^2 con las tres restricciones definidas como:

- Cada fila debe contener todos los números del 1 al R^2.
- Cada columna debe contener todos los números del 1 al R^2.
- Cada caja de tamaño R x R debe contener todos los números del 1 al R^2.

Así, el problema clásico de Sudoku de tamaño 9 x 9 es el problema de Sudoku generalizado de rank 3.

Algoritmos Implementados

Algoritmo de Backtracking

El algoritmo de backtracking se puede resumir en el siguiente pseudocódigo:

```
backtracking(){
      si(la_matriz_está_completa){
            terminar algoritmo
      }sino si(la_matriz_es_válida){
            celda := encontrar_celda_vacía
            desde(número := 1 al n){
                  celda := número
                  backtracking()
                  celda := vacío
            }
      }
}
```

En la implementación, para calcular la encontrar_celda_vacía, se recorre la matriz por fila y para cada fila se verifica si la celda se encuentra vacía. Así, elige la primera celda vacía encontrada.
Para verificar el valor de la_matriz_es_válida, se verifican cada fila, columna y cajas, si se repiten los números dentro de las mismas.
Para verificar si la matriz se encuentra completa o no, se realiza un recorrido de toda la matriz para buscar alguna celda vacía.
Tanto estas búsquedas y verificaciones se realizan en tiempo de orden cuadrático.

Algoritmo de Búsqueda por Heurística de la variable más restringida

Este algoritmo es muy similar al algoritmo de backtracking, sólo que la manera de elegir la celda para encontrar_celda_vacía es distinta. No elige a la primera celda vacía encontrada, sino que primero se calcula el número de restricciones para cada celda vacía y luego se elige la celda con la mayor restricción.

Algoritmo de Las Vegas

Este algoritmo también es similar al algoritmo de backtracking, pero se diferencia en la elección de la celda a rellenar y el número a rellenar.

Para calcular la celda para encontrar_celda_vacía, se elige aleatoriamente una celda y verifica si la celda se encuentra vacía. Sino, se elige otra hasta encontrar una celda vacía.

Para elegir el número a rellenar, se elige un número aleatorio. Para evitar que se siga eligiendo el mismo número aleatorio, se crea una lista de números del 1 al n, la lista es mezclada y se eligen los números de la lista de manera ordenada.

Los procesos aleatorios se tomaron las decisiones con la distribución uniforme de probabilidades.

Resultados Experimentales

Para cada algoritmo se buscó entender su comportamiento según la variación del Rank y de la Cantidad de Celdas Vacías. El rank se varió desde 2 hasta 7, inclusive. La cantidad de celdas vacías se calculó en función a un factor Taza de Vacíos, donde la cantidad es igual a $(rank)^6 \times (Taza\ de\ Vacios)$ es decir a mayor Tasa mayor cantidad de celdas vacías.
La Taza de Vacíos inicia con 0,1 y se incrementa de a 0,1. El valor máximo está acotado por un tiempo de ejecución de 30s.
Los resultados son el Tiempo esperado (nanosegundos) y la Cantidad de nodos esperado, los cuales son el promedio de 5 muestras.

Búsqueda por Heurística.

Rank	Tasa de Vacios	Tiempo Esperado	Cantidad de Nodos Esperado
2	0,1	1594967,8	6
2	0,2	485803,4	22
2	0,3	408662	45
2	0,4	2036995,2	76
2	0,5	694017,2	117
2	0,6	307200,6	165
2	0,7	1456216	221
2	0,8	795223,2	284
2	0,9	696406,6	355
3	0,1	1698307	412
3	0,2	3289776,4	481
3	0,3	1536599,8	590
3	0,4	2192558,4	739
3	0,5	5601375,2	928
3	0,6	14939673,6	1168,8

4	0,1	8356622,2	1360
4	0,2	14966127,2	1568
4	0,3	23406290,6	1903
4	0,4	88016107,6	2366
5	0,1	70649208,8	2761
5	0,2	50628609,2	3265
5	0,3	56554933,8	4081
5	0,4	179415260,8	5210
6	0,1	82129206,8	6102
6	0,2	298909779,8	7142
6	0,3	445527373,4	8829
7	0,1	226012546	10330
7	0,2	407516513,8	12255
7	0,3	1079881352	15380

Busqueda por BackTracking

Rank	Tasa de Vacios	Tiempo Esperado	Cantidad de Nodos Esperado
2	0,1	837462,8	3,4
2	0,2	155563,2	8
2	0,3	156757,8	11,6
2	0,4	193536,6	18
2	0,5	135936,4	22
2	0,6	136192,2	24,8
2	0,7	172971	40,2
2	0,8	175787	35
2	0,9	179371,2	43,2
3	0,1	387157,8	37,6
3	0,2	716886,6	77,8
3	0,3	908204,2	130,6
3	0,4	1973166,4	276,2
3	0,5	3129264	508,6
3	0,6	34501691	5820,6
3	0,7	539475267,6	95472,8
3	0,8	366695623	65675,2
4	0,1	945239,2	216,6

4	0,2	2439342,8	536
4	0,3	9015993,8	2179,2
4	0,4	46147663	13048
5	0,1	7189857,4	845,6
5	0,2	20374818,8	2274,8
5	0,3	179620573	20126,2
6	0,1	44536822,8	3009,6
6	0,2	280422537,4	17671,4
7	0,1	236109544	10401,2
7	0,2	1854816094	71399

Búsqueda por Las Vegas

Rank	Tasa de Vacios	Tiempo Esperado	Cantidad de Nodos Esperado
2	0,1	1009495,2	3,4
2	0,2	167936	8,2
2	0,3	122624	11,4
2	0,4	155306,8	17,8
2	0,5	201728,2	22
2	0,6	189440	26
2	0,7	162048,2	32,4
2	0,8	225707,2	45,8
2	0,9	180821,4	36
3	0,1	380587,4	43,8
3	0,2	673366,6	75,8
3	0,3	1378391,8	184
3	0,4	1743619,2	227
3	0,5	5411934,6	811,4
3	0,6	55303944,8	9897,8
3	0,7	966381510,8	180750,2
4	0,1	880300,2	215,2
4	0,2	2986756,6	577,4
4	0,3	8482318,6	2087,2
4	0,4	91729223,2	20180,2
5	0,1	8694628	841,8
5	0,2	27559471,2	2634
5	0,3	161421843,4	21670

6	0,1	41548871,2	2541,2
6	0,2	196636922,4	14376,2
7	0,1	157529100,8	6397,6
7	0,2	1860136892	74002,8

Conclusiones

Como era de esperarse, el tiempo y la cantidad de nodos expandidos se incrementa según la cantidad de celdas vacías en todos los algoritmos. Además, el BT expande la mayor cantidad de nodos.
Los valores máximos de tasa de vacíos son menores en BT debido a que el algoritmo suele tardar más cuando más celdas están vacías. Sin embargo, para ranks pequeños BT presenta un rendimiento superior tanto en tiempo como en nodos expandidos, esto se debe a la baja cardinalidad del dominio de las variables.
Los resultados de Las Vegas y BT son esencialmente los mismos, tanto la máxima tasa de vacíos como el rendimiento en tiempo y nodos. Esto se debe a que LV es una versión no sesgada de BT. Existen instancias construidas manualmente que provocan el peor rendimiento en BT, sin embargo esto no ocurre con LV con una alta probabilidad.

Todo el proyecto se puede encontrar en: https://github.com/pillowpilot/ia_tp2

Referencias

Yato, Takayuki, and Takahiro Seta. "Complexity and completeness of finding another solution and its application to puzzles." *IEICE transactions on fundamentals of electronics, communications and computer sciences* 86.5 (2003): 1052-1060.

Colbourn, C. J., M. J. Colbourn, and D. R. Stinson. "The computational complexity of recognizing critical sets." *Graph Theory Singapore 1983*. Springer, Berlin, Heidelberg, 1984. 248-253.