Problemas de Satisfacción de Restricciones y Optimización

Inteligencia Artificial

Javier Gutiérrez Rodríguez

Grupo de mañanas

6/5/20

Problema de Satisfacción de Restricciones: Sudoku

El problema elegido es el conocido juego matemático Sudoku, creado por Leonhard Euler en el siglo XVIII y popularizado en japón en la década de los ochenta.

El juego consiste en colocar números en una tabla, normalmente con unas dimensiones de 9x9, a su vez dividida en 9 subtablas de 3x3. De serie se dan unos numero iniciales ya colocados en la tablas y el objetivo es rellenar los espacios vacíos de la tabla sin repetir el mismo numero en la fila, en la columna y dentro de la propia subtabla de 3x3. Podemos ver un ejemplo en la tabla 1.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Tabla 1.

Este popular pasatiempo es un claro ejemplo de un problema de restricciones dado que cumple la definición de los PSR:

- Conjunto de variables: Cada casilla de la tabla es una variable posible.
- Cada variable tiene un dominio no vacío de valores posibles: El dominio de cada casilla son los valores entre 1 y 9.
- Tiene un conjunto de restricciones:
  - R1: Las filas no pueden tener números duplicados.
  - R2: Las columnas no pueden tener números duplicados.
  - R3: Las subtablas no pueden tener números duplicados
- A todas las casillas vacías de la tabla se les debe de asignar un valor que este comprendido en el domino de valores y que cumpla las restricciones.

Este problema se pude solucionar mediante el algoritmo de vuelta atrás. El algoritmo en pseudocodigo es el siguiente:

```
VueltaAtrasSudoku ( tabla x ) {
    Si todas las casillas de la tabla han sido asignadas {
        devuelve verdadero (por lo tanto hemos encontrado una solución)
    }
    Si todas las casillas no están asignadas {
        Para todas las opciones (el dominio de los valores, de 1 a 9) {
            Si no hay conflictos dado el tablero x en la casilla c y con la opcion escogida{
                se le añade al tablero x la opción elegida en la casilla c
                Si VueltaAtrasSudoku(tablero x) se cumple {
                    se devuelve verdadero
                }
                Si no se cumple VueltaAtrasSudoku(tablero x) {
                    se deshacen los cambios en el tablero x
                }
            }
        }
    }
    Ninguna opción funciono, por lo tanto se devuelve falso
}
```

Resolución mediante Ascenso de colinas

Una técnica de optimización que puede resolver un sudoku es la técnica de ascenso de colinas. Esta técnica hace que el algoritmo se mueva en dirección del valor creciente y termina cuando se alcanza el optimo local. En este caso utilizamos la variante de escalada por máxima pendiente, la cual se centra en seleccionar el mejor movimiento (máximo local).

Para escapar de los óptimos locales lo que haremos sera reiniciar la búsqueda en otro punto, ya que puede haber veces que se quede estancado.

El algoritmo en pseudocodigo seria el siguiente:

```
AscensoColinasMaximaPendiente (Sudoku nodo, entero iteraciones){
    Se crea un nodo aleatorio vecino

    Valor valor_nodo = numero de errores en nodo
    Valor valor_vecino = numero de errores en el nodo vecino

    Si el numero de iteraciones del algoritmo es mayor a X, donde x un entero elegido por nosotros
        se devuelve el nodo

    Si el valor_vecino = 0, es decir, un optimo local
        se devuelve el vecino
    Si valor_vecino > valor_nodo
        AscensoColinasMaximaPendiente (nodo) se ejecuta otra vez
    Si no se cumple ninguna de las anteriores
        AscensoColinasMaximaPendiente (vecino)

    iteraciones++
}
```

El algoritmo de resolución seria:

```
Resolución (Sudoku inicial){
        hacer hasta que num_errores sea 0 {

                Sudoku solución = AscensoColinasMaximaPendiente(inicial,iteraciones);

                num_errores = numero de errores de solución
        }

        se devuelve solución
}
```