

## Sudoku como un problema de satisfacción de restricciones.

### Introducción al sudoku.

El sudoku es un pasatiempo lógico que goza de gran popularidad actualmente. El objetivo del juego es rellenar con los números del 1 al 9 las celdas de una cuadrícula de tamaño 9x9, subdividida a su vez en 9 subcuadrículas 3x3 -llamadas regiones o cajas- de forma que cada fila, columna y región no tenga ningún número repetido.

El puzzle parte de un cierto número de dígitos ya colocados en la cuadrícula, y si un sudoku está bien planteado, tiene una solución única.

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

Fig: 1 Sudoku

### Problemas de satisfacción de restricciones (CSP: Constraint Satisfaction Problem)

Un CSP es un tipo de problema que debemos resolver buscando estados del problema que cumplan una serie de restricciones o criterios. Los CSP en el caso general son problemas NP-completos, es decir, comprobar si una solución de un CSP es correcta o no tiene una complejidad polinomial. En el problema concreto del sudoku la complejidad es  $O(n^2)$

Formalmente un problema de satisfacción de restricciones se define con una *red de restricciones* (X D C) donde:

$X = \{x_1, x_2, \dots, x_n\}$  Es un conjunto de variables

$D = \{d_1, d_2, \dots, d_m\}$  Es una colección de dominios finitos y no vacíos tal que  $\forall x_i \in X \Rightarrow x_i \in d_i$

$C = \{c_1, c_2, \dots, c_k\}$  Es un conjunto de restricciones

Una restricción  $c_i$  es una relación entre un subconjunto de variables de X, y el subconjunto de dominios pertenecientes a D, asociados al subconjunto de variables, especificando la combinación de valores del dominio permitidos para dicho subconjunto de variables. El número de variables a las que afecta una restricción se denomina ámbito. Cuando el ámbito es uno o dos, hablamos de un CSP unario o binario respectivamente, que son los problemas más comunes y con mayor información disponible. Alternativamente cualquier CSP de aridad  $>2$  puede convertirse en un CSP binario añadiendo variables auxiliares.

Las restricciones pueden visualizarse como un grafo de restricciones, en el que cada nodo se corresponde con una variable, y existe un arco  $(x_i, x_j)$  si existe una restricción que involucra a las variables  $x_i$  y  $x_j$ .

## Métodos de resolución del problema

Una de las formas más directas para resolver un CSP es utilizar una búsqueda sistemática en el espacio de estados del problema, buscando una asignación de variables que cumple todas las restricciones. El espacio de estados puede representarse por un árbol de búsqueda (o árbol de expansión) que tiene profundidad acotada, por lo que los métodos de búsqueda del tipo primero en profundidad suelen ser los más adecuados.

### Backtracking

Backtracking es un algoritmo de búsqueda primero en profundidad, que selecciona consecutivamente valores para una única variable en cada paso, y vuelve atrás en el árbol de búsqueda cuando una variable no tiene valores legales que asignarle, simplificando efectivamente el espacio de búsqueda, ya que está desechando todo el subárbol que depende de dicha variable sin valores legales. Una búsqueda por backtracking proporcionará la solución al sudoku, supuesto éste bien planteado.

El pseudo-código para resolver el problema sería:

```
funcion BackTrackingCSP (A: asignaciones de variables, csp: datos del CSP)
: retorna una asignación de variables para el problema
  si esSolucion(A) hacer
    retorna A
  fin si
  x ← seleccionaVariableNoAsignada()
  para cada valor en csp.dominio(x) hacer
    x ← valor
    si cumpleRestricciones(x ∪ A) hacer
      A ← x ∪ A
      resultado ← BackTrackingCSP(A, csp)
      si esInconsistente(resultado) hacer
        A ← A - x
      fin si
    fin si
  fin para
fin funcion
```

Sin embargo, aunque una búsqueda por backtracking nos daría una solución, no es la opción más eficiente. Disponemos de varias heurísticas destinadas a mejorar la efectividad de una búsqueda simple por backtracking:

- **Minimun Remaining Values (MRV)**

Esta heurística nos sirve para seleccionar la variable no asignada que el algoritmo debe procesar (se implementaría en el procedimiento `seleccionaVariableNoAsignada` del pseudocódigo anterior).

Para ello, de entre todas las variables con valor no asignado disponibles, seleccionamos aquella que tiene menos valores disponibles en su dominio, ya que es la que más probabilidades tiene de causar un fallo, y, por tanto, de realizar una poda en el árbol de búsqueda.

- **Heurística de grado**

Esta heurística es complementaria a la anterior; si tenemos varias variables que cumplen MRV, se selecciona aquella variable que está involucrada en mayor número de restricciones con otras variables no asignadas.

- ***Valor menos restrictivo***

Para seleccionar el valor de una variable  $x_{ij}$  de entre todos los permitidos por el dominio actual de esa variable, se escoge antes aquel valor que elimina menos valores del dominio de las variables sin valor asignado con las que  $x_{ij}$  está conectada en el grafo de restricciones. Para un problema CSP en el que haya varias soluciones posibles esta heurística no es necesaria, ya que simplemente tendríamos que probar todas las combinaciones de valores consistentes, pero en el caso del problema del sudoku, con solución única, esta heurística puede ayudar a la poda.

## **Propagación de restricciones.**

Las heurísticas anteriores mejoran la búsqueda utilizando la información de las restricciones que afectan a las variables en el momento de elegir aquella que tiene más probabilidades de reducir el espacio de búsqueda. Sin embargo, si consideramos las restricciones entre variables durante la búsqueda, o incluso antes de que ésta empiece, podemos reducir aún más el espacio de búsqueda. En esencia lo que pretende es reducir los dominios de las variables, siempre de acuerdo con las restricciones, para poder localizar más inconsistencias.

- ***Forward Checking***

Dada una variable  $x_{ij}$ , a la que se le asigna un valor, se elimina dicho valor del dominio de todas las variables con las que  $x_{ij}$  está conectada en el grafo de restricciones. MRV es una heurística particularmente apropiada para Forward Checking ya que se podría entender que FC aumenta la información disponible para la heurística MRV, y por tanto mejora su eficiencia. En conjunto se detectan más inconsistencias, aumentando efectivamente la poda de la búsqueda por backtracking.

- ***Arc consistency (consistencia de arcos)***

Un arco  $(X_i, X_j)$  en el grafo de restricciones es consistente si para todo valor  $x_i$  permitido para  $X_i$  en su dominio actual, hay al menos un valor  $x_j$ , permitido para  $X_j$ , tal que las asignaciones  $X_i = x_i$  y  $X_j = x_j$  no viola ninguna restricción existente entre  $X_i$  y  $X_j$ .

Que un arco  $(X_i, X_j)$  sea consistente no implica que  $(X_j, X_i)$  también lo sea.

Para hacer un arco  $(X_i, X_j)$  consistente, basta con eliminar los valores del dominio de  $X_i$  para los que la condición anterior no es cierta. Esto no elimina soluciones del CSP original.

El proceso para obtener arcos consistentes se debe aplicar reiterativamente entre las variables conectadas hasta que no se produce ningún cambio en los dominios, pero sólo para los arcos que fueran inconsistentes. Para ese proceso tenemos el algoritmo AC-3, que mantiene una lista con los arcos para los cuales se ha de comprobar inconsistencias.

```

funcion AC-3 (csp: datos del CSP)

    queue : lista de arcos, inicialmente todos los arcos del CSP

    mientras noVacia(queue) hacer
        arco ← obtenerPrimero(queue)
        si eliminarValoresInconsistentes(arco) entonces hacer
            para cada  $X_k$  adyacente a  $X_i$  hacer
                add(queue,  $X_k$ )
            fpara
        fsi
    fmientras

ffuncion



---


funcion eliminarValoresInconsistentes( $X_i$   $X_j$  : arco)
    flagDominioEliminado ← falso
    para cada  $x \in \text{dominio}(X_i)$  hacer
        si  $\neg y \in \text{dominio}(X_j)$  es consistente con  $(X_i, X_j)$  hacer
            dominio( $X_i$ ) ← dominio( $X_i$ ) -  $x$ 
            flagDominioEliminado ← verdadero
        fsi
    fpara
    → flagDominioEliminado
ffuncion

```

## Sudoku como un CSP

Formalmente el sudoku como un problema CSP se formularía:

Variables:  $X = \{ x_{ij} / i, j \in \{1, 2, \dots, 9\} \}$

Dominio:  $D = \{1, 2, \dots, 9\} \forall x_{ij} \in X$

Restricciones:

$x_{ij}, x_{ik} \in X$  tal que  $x_{ij} \neq x_{ik}$

$x_{ij}, x_{kj} \in X$  tal que  $x_{ij} \neq x_{kj}$

$x_{ij}, x_{km} \in X$  con  $[ ((i-1) \div 3) + 3 \cdot ((j-1) \div 3) ] = [ ((k-1) \div 3) + 3 \cdot ((m-1) \div 3) ]$   
tal que  $x_{ij} \neq x_{km}$

Cada restricción sólo afecta a dos variables, por lo que tenemos un CSP binario.