Lab09: Codificar el problema Sudoku en instancias de SAT

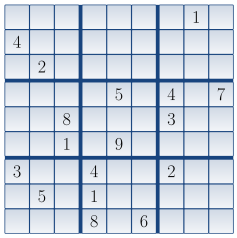
Vamos a ver como los SAT-solvers modernos se pueden usar para resolver problemas difciles.

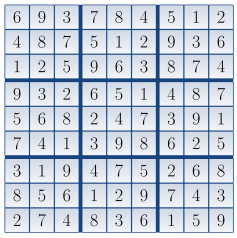
1. **El problema**

   En un sudoku tenemos que completar una matriz M de dimensión $n \times n$, donde $n = d^2$, con $d > 1$. Los elementos para completar M hay que elegirlos de un conjunto de n elementos. M está parcialmente rellena y para completarla hay que seguir estas reglas:

   - cada fila de M contiene los n elementos
   - cada columna de M contiene los n elemntos
   - cada una de las submatrices disjuntas de $d \times d$ de M contiene los n elementos

   Por ejemplo tenemos este sudoku cuyos elementos son los dgitos $\{1, 2, \cdots, 9\}$.





2. **Explicación de la reducción (Sudoku $\leq_p$ SAT)**

   Supongamos un sudoku M de dimensión $n \times n$, con $n = d^2$. Queremos reducir el problema a SAT, esto es, construir una CNF φ, a partir de M, de forma que M tiene solución si y sólo si φ es satisfactible. Además queremos usar una asignación que haga cierta a φ para construir una solución del sudoku.

   Primero la variables de φ: necesitamos n^3 variables: $x_{i,j,k}$ donde $0 \leq i, j, k, \leq n-1$. La idea es que estas variables sean True cuando en la fila i y en la columna j, se encuentre el elemento k.

   Las cláusulas de φ deben asegurar las reglas para completar un sudoku:

   (a) En cada fila están los n elementos:

   $$\bigwedge_{i \in [0..n-1]} \bigwedge_{k \in [1..n]} (x_{i,0,k} \vee x_{i,1,k} \vee \cdots \vee x_{i,n-1,k}))$$

   (b) En cada columna están los n elementos:

   $$\bigwedge_{j \in [0..n-1]} \bigwedge_{k \in [1..n]} (x_{0,j,k} \vee x_{1,j,k} \vee \cdots \vee x_{n-1,j,k}))$$

   (c) Por cada elemento k ya colocado al principio en una fila i y columna j:

   $$x_{i,j,k}$$

(d) En cada submatriz de dimensión $d \times d$ están los n elementos:

$$\bigwedge_{i' \in [0..d-1]} \bigwedge_{j' \in [0..d-1]} \bigwedge_{k \in [1..n]} \Big(\bigvee_{(i,j) \in M_d(i',j')} x_{i,j,k} \Big)$$

donde

$$M_d(u,v) = \{(ud+i, vd+j) \mid i,j \in [0..d-1]\}$$

Además necesitamos asegurar que

(e) en cada posición de la matriz hay como mucho un elemento:

$$\bigwedge_{i \in [0..n-1]} \bigwedge_{j \in [0..n-1]} \bigwedge_{k \in [1..n]} \bigwedge_{k' \in [k+1..n]} \neg(x_{i,j,k} \wedge x_{i,j,k'})$$

La fórmula anterior no es una CNF pero, aplicando las leyes de Demorgan, es equivalente a:

$$\bigwedge_{i \in [0..n-1]} \bigwedge_{j \in [0..n-1]} \bigwedge_{k \in [1..n]} \bigwedge_{k' \in [k+1..n]} (\neg x_{i,j,k} \vee \neg x_{i,j,k'})$$

La CNF φ es la conjunción de todas las cláusulas anteriores.

3. **Tarea a realizar**

   – Debes implementar la función `encode(sudoku,d)` que recibe como entrada un `sudoku` (una lista de listas) de dimensión $n \times n$, con $n = d \times d$. Esta función debe generar la CNF, φ, que se obtiene cuando se reduce el sudoku de entrada según la explicación del apartado anterior. Como las variables de φ son números positivos, dispones de una función biyectiva `var2positive(i,j,k,d)`, que asocia un número positivo a la variable $x_{i,j,k}$. En concreto $var2positive(i,j,k,d) = i \times d^4 + j \times d^2 + k$. Por otro lado, la función `positive2var(m,d)` hace la operación contraria: dado un positivo m, devuelve los tres valores (i,j,k) de la variable asociada a m.

   Cuando hayas generado φ, hay que llamar al solver de Pysat (en este caso al solver Mergesat3) y obtener una asignación cuando φ es satisfactible. Esto lo hace la función `solve_and_decode`, que ya está implementada. Esta función además se encarga de construir la solución del sudoku a partir de la asignación obtenida (para ello tiene que llamar a `positive2var`).

   – Implementa también `validate_solution(solution,d,symbols)`, que cuando recibe la solución de un sudoku de dimensión $n \times n$ con $n = d \times d$, y el juego de símbolos que contiene la solución, decide si es válida o no. Esto es, si la solución está completa y si cumple con las reglas de los sudokus.

4. **Tests**

   Encontrarás comentados seis tests. El primero comprueba `validate_solution`. Los siguientes tests llaman a distintos sudokus de la carpeta `instancias`: un sudoku de 4×4, un sudoku de 9×9, un sudoku de 16×16, un sudoku de 25×25 y una colección de 50 sudokus de 9×9 que irán dejando la solución encontrada en la carpeta `soluciones`.

   Ya verás que los sudokus de 9×9 se resuelven rápido, pero costará el de 16×16 y el de 25×25.