GAME SUDOKU

ANTÓN MAESTRE GÓMEZ - SISTEMAS INTELIGENTES

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

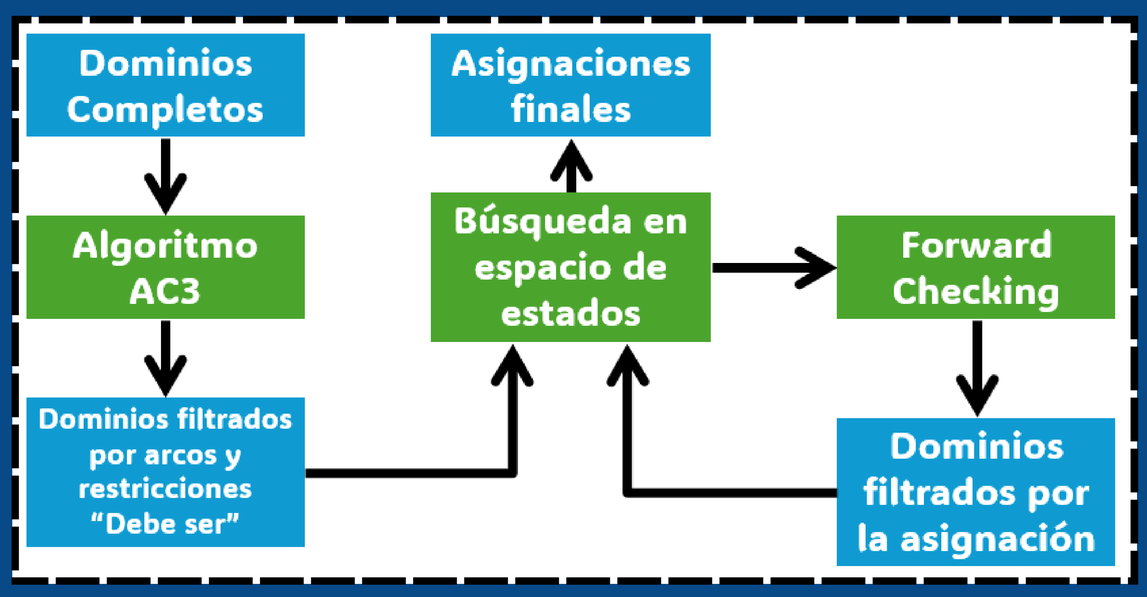
INTRODUCCIÓN

Utilizando un agente basado en una máquina de satisfacción de restricciones, se pide resolver cualquier sudoku proporcionado en modo texto.

META

El objetivo del sudoku es rellenar todas las casillas de forma que no se repitan números del 1 al 9 en cada:

- Fila
- Columna
- Cuadrado 3x3



ANÁLISIS

En una primera aproximación, programé el algoritmo ya descrito. Pese a que resolvía los sudokus correctamente, los tiempos de ejecución se disparaban. Esto se debía a la gran cantidad de estados que debía comprobar el algoritmo para ciertos sudokus.

Realizando una segunda aproximación, decidí implementar la técnica de "forward checking". De esta forma, cada vez que se realiza una asignación en la búsqueda, se retira este valor de los dominios de las variables que compartan la restricción AllDisjoint con la variable afectada. Al realizar backtracking, será necesario revertir estos cambios en los dominios (forward checking reverse).

Con esta segunda aproximación, el algoritmo conseguía solucionar los sudokus en mucho menos tiempo que en un principio, ya que tiene que generar y comprobar muchos menos estados.

METODOLOGÍA

En primer lugar, se lee el sudoku correspondiente y se almacena cada casilla como "Variable" (inicializa el dominio con todos los valores del 1 al 9) en un array 9x9 denominado "Tablero". También almacenamos cada variable como atributo de una asignación(su valor se inicializa a -1). Estas instancias las guardamos en otro array. Finalmente, aquellas casillas que ya tienen un valor en el sudoku, se introducen como restricciones de tipo "DebeSer" en el array de restricciones.

Antes de aplicar la búsqueda en el espacio de estados, pasamos los dominios de las variables por un algoritmo AC3. Una vez el algoritmo define todos los arcos a tener en cuenta (de filas, columnas y 3x3) , reduce los dominios de las variables que presentan restricciones "DebeSer" (únicamente deja en el dominio el valor que debe tener la variable). Posteriormente, va comprobando si cada arco es consistente de manera que si el arco donde la variable distinguida "a" no es consistente, elimina de su dominio los valores que la hacen inconsistente y añade a los arcos aquellos donde "a" es no distinguida. De esta forma, el AC3 devolverá el conjunto de dominios actualizado tal que todos los arcos del problema sean consistentes.

La búsqueda en el espacio estará compuesta de estados que guardarán en cada momento las asignaciones que se han llevado a cabo hasta ese estado. El estado inicial añadirá al conjunto de restricciones los AllDisjoint (variables que deben tener asignaciones con valores diferentes) correspondientes a filas, columnas y 3x3.

Mientras que la lista de abiertos no esté vacía, el algoritmo de búsqueda va generando y añadiendo los sucesores válidos, asignando valores de los dominios (ya filtrados) a las casillas y comprobando que se cumplan las restricciones. Si el estado actual no ha generado sucesores (no es un camino válido) el algoritmo realiza backtracking, retirando las asignaciones que haya realizado para llegar a ese estado (hasta tener las del siguiente estado en la lista de abiertos).

Una vez se consigue un estado que contenga todas las asignaciones, se considera que se ha llegado a la meta y devuelve la lista de asignaciones correspondientes.

CONCLUSIÓN

Realizando una prueba sobre 10 sudokus distintos, percibimos que la diferencia de los tiempos de ejecución es abismal. La adición del forward checking provoca que el algoritmo resuelva los sudokus en milesimas de segundo, volviendo mucho más eficiente al agente (color amarillo).

Así pues, la segunda aproximación será la más óptima si nuestro objetivo es la resolución en el menor tiempo posible.