PRÁCTICAS

DE

SISTEMAS

INTELIGENTES

PRÁCTICA 1:

SUDOKU

DOCUMENTACIÓN

ALEJANDRO REYES ALBILLAR

45931406 - S

ara65@alu.ua.es

Curso 2015 / 2016

Grado en Ingeniería Informática

Universitat d'Alacant
Universidad de Alicante

1. INTRUDUCCIÓN

La práctica propuesta consiste en varias partes. La primera de estas partes nos dice que debemos solucionar un sudoku utilizando el algoritmo de backtracking o algoritmo de vuelta atrás. Dicho algoritmo consiste en probar una solución posible, comprobar si es factible y, en el caso de serlo, continuar hasta terminar de rellenar las casillas.

En este nos pide rellenar un sudoku consistente en una cuadrícula de 9x9 celdas, agrupadas en 9 submatriz de 3x3. Además de esto, las reglas del juego especifican que se deben rellenar los huecos vacíos de forma que los números, del 1 al 9, no se repitan en la misma fila, columna, o submatriz.

Teniendo todo esto en mente únicamente nos falta comenzar a programar, para lo cual se nos proporciona un código base con una interfaz y un conjunto de clases que nos permitirán llevar a cabo una serie de pruebas más cómodamente. A continuación se muestra una captura (Imagen 1) de la interfaz básica que se nos proporciona.

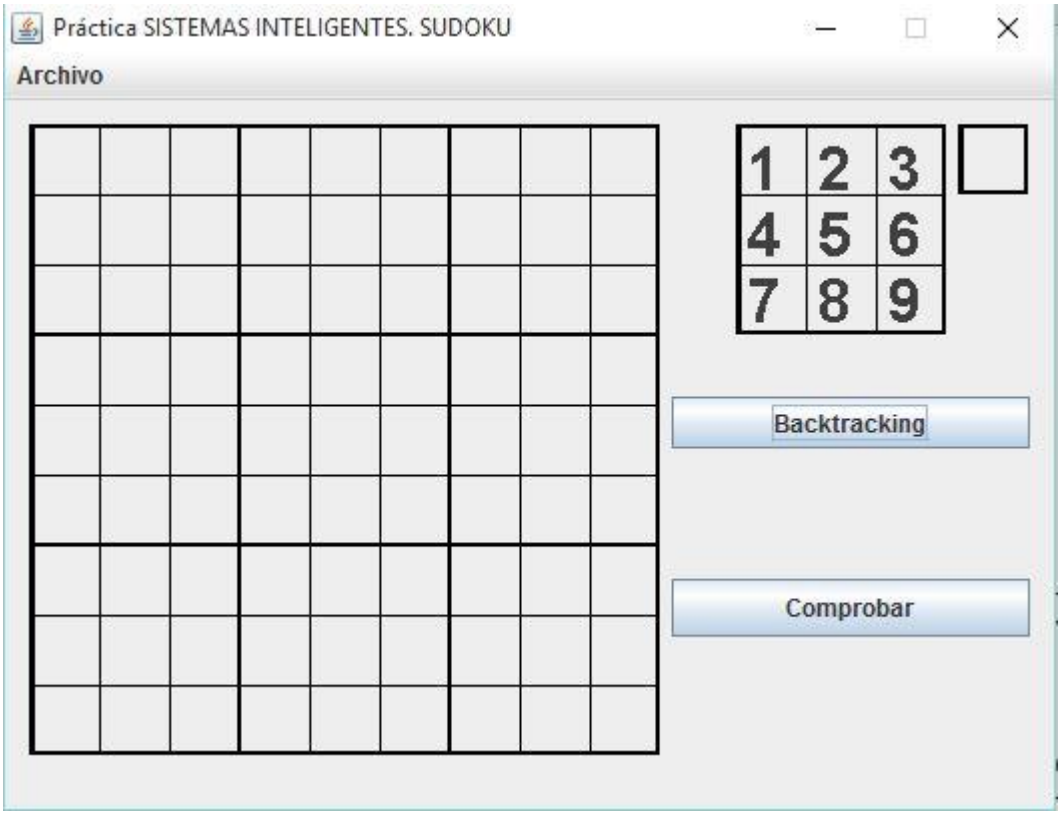


Imagen 1: Interfaz del Sudoku proporcionada por el profesor.

La interfaz cuenta actualmente con 2 botones principales que ejecutan el algoritmo de backtracking y una comprobación de que el sudoku introducido es correcto. Además de esto, dentro de la pestaña archivo se nos permite cargar sudokus desde archivo y borrar el tablero completo para dejarlo tal y como se ve en la imagen.

Se nos han proporcionado ficheros, con el formato a continuación mostrado (Figura 1), que nos permite cargar en la interfaz algunos sudokus, de modo que también podemos incluir sudokus hechos por los usuarios.

```
3 1 2 4 0 0 0 0 8
7 4 0 0 9 3 6 0 0
0 6 0 1 0 8 0 0 0
1 8 0 0 6 7 0 5 4
5 7 4 3 0 2 0 6 0
6 2 9 5 0 1 8 0 0
8 0 1 0 2 0 0 3 0
0 3 6 8 1 0 5 0 0
2 0 0 6 0 4 1 0 9
```

Figura 1. Formato de los ficheros de texto con los sudokus.

2. EL PROBLEMA DEL SUDOKU COMO UN CSP

El sudoku, como hemos mencionado anteriormente consta de una matriz de posiciones i y j en la que cada una de ellas tiene un valor contenido en el dominio 1-9.

V= Conjunto de variables

D= Dominio de las variables

ρ = Conjunto de restricciones

S = Submatriz a la que pertenece una variable

$V = \{V_{0,0}, \dots, V_{0,j}, \dots, V_{i,0}, \dots, V_{i,j}\}$ $V=\{V_{i,j}\}_{i=0,\dots,8 \;\; j=0,\dots,8}$

$D_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}, \forall k, 1 \leq k \leq 81$

$\rho_k(V_i, V_j) = \{<v_i, v_j> \mid v_i \in D_i, v_j \in D_j, v_i \neq v_j\}, \forall k, 1 \leq k \leq 81$

$c(e_i) = <V_{ij}, V_{im}>, <V_{ij},V_{nj}>, \{<V_{ij}, V_{nm}> \mid V_{ij} \in S_x, V_{nm} \in S_x\} \forall x, 1 \leq x \leq 9$

Esto se traduce de la siguiente manera:

Cualquier variable i que se encuentre en la misma fila, columna o submatriz que una variable j deberá cumplir la restricción siguiente:

- El valor de la variable i no puede ser nunca igual a la variable j, por lo que su dominio queda reducido.

3. IMPLEMENTACIÓN

A. BACKTRACKING

En mi caso, y teniendo en cuenta que se debe implementar otro algoritmo basado en dominios, he optado por crear una solución basada en recursividad, que utilice un Array de 81 casillas que poseen:

- Posición en el eje X de la matriz.
- Posición en el eje Y de la matriz.
- Valor de la casilla
- Dominio de la casilla.

Dicho Array, tras ser inicializado con los valores iniciales dados por el tablero, se envían a un módulo que utilizará el dominio asignado, originalmente de 1 a 9 exceptuando las casillas que ya tienen un valor asignado antes de iniciar, para introducir un valor posible en el tablero, comprobar si dicho valor es correcto y probar la siguiente casilla mediante una llamada recursiva al módulo.

Tras unas cuantas llamadas recursivas al módulo en cuestión, se llega a la última casilla, la cual, al ser comprobada y definida como correcta, devolverá true al módulo que lo llamó de manera recursiva hasta regresar a la llamada principal, que imprimirá la matriz solucionada sobre la interfaz.

Además del algoritmo se ha implementado un algoritmo de seguridad que comprueba que el fichero cargado en la interfaz es correcto y no contiene casillas repetidas en la misma fila, columna o submatriz.

A continuación se muestra el extracto de código correspondiente al algoritmo de backtracking (Figura 3) y al módulo inicializador que recibe únicamente el tablero (Figura 2):

```
public boolean ejecutarBC(Tablero tablero) {
   //El booleano ok guardará el valor que devolverá la ejecución del algoritmo
   boolean ok = true;
   if (!buscadoAC3) {
      //Rellena el array de casillas
      init(tablero);
   }

   if (!compruebaCargado(tablero, casillas)) {
      return false;
   }

   try {

      int vacias = 0;
```

```
            for (Casilla casilla : casillas) {
                if (casilla.getValor() == 0) {
                    vacias++;
                }
            }
            int it = 0;
            Casilla[] cambiar = new Casilla[vacias];
            for (Casilla casilla : casillas) {
                if (casilla.getValor() == 0) {
                    cambiar[it] = casilla;
                    it++;
                }
            }

            //ejecución del algoritmo de bactracking
            long init;
            long fin;
            long total;
            init = System.nanoTime();
            ok = back(tablero, cambiar, 0);
            fin = System.nanoTime();
            total = fin - init;
            System.out.println("El tiempo tardado en ejecutar Backtracking ha sido: " + total + " ns.");
            buscadoAC3 = false;

        } catch (Exception ex) {
            System.err.println("El algoritmo de Backtracking ha tenido un problema: " + ex);
        }

        return ok;
    }
```

Figura 2: Módulo principal llamado desde la interfaz.

```
public boolean back(Tablero tablero, Casilla[] casillas, int pos) {
        boolean result = false;

        if (pos == casillas.length) {
            return true;
        }
        for (int i = 1; i <= 9; i++) {
            if (casillas[pos].getValor() == 0) {
                int[] aux = casillas[pos].getDom();
                int[] dom = new int[10];
                dom[0] = 0;
                for (int j = 1; j < 10; j++) {//haciendo copia por valor
                    dom[j] = aux[j];
                }
                tablero.setCasilla(i, casillas[pos].getFil(), casillas[pos].getCol());
                casillas[pos].setValor(i);
                dom[i] = 0;
                if (correcto(tablero, casillas[pos].getFil(), casillas[pos].getCol())) {
                    result = back(tablero, casillas, pos + 1);
```

```
            if (result == true) {
               return true;
            }
         }
         dom[i] = 1;
         tablero.setCasilla(0, casillas[pos].getFil(), casillas[pos].getCol());
         casillas[pos].setValor(0);
         casillas[pos].setDom(dom);
      } else {
         result = back(tablero, casillas, pos + 1);
      }
   }
   return result;
}
```

Figura 3: Módulo que ejecuta el algoritmo de backtracking.

A continuación se muestran las imágenes de uno de los sudokus proporcionados por el profesorado de la asignatura (Imagen 2) y la solución obtenida por el algoritmo de backtracking (Imagen 3):

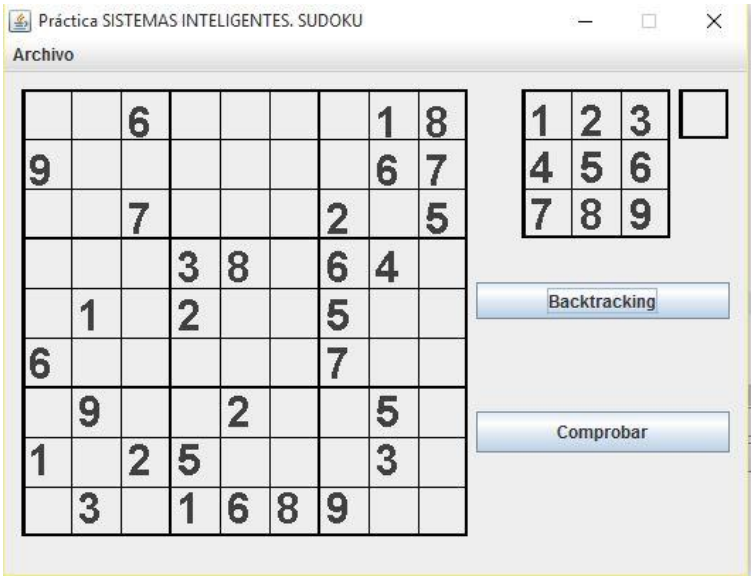


Imagen 2: Sudoku proporcionado por el profesorado.

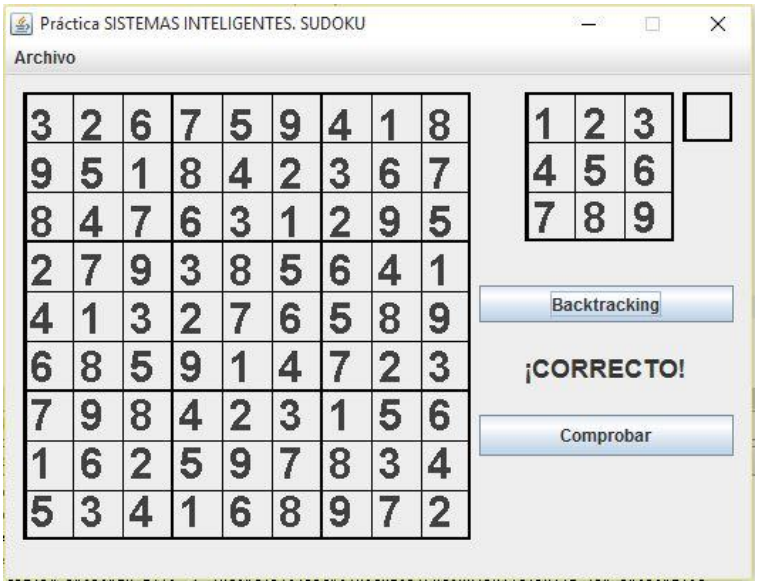


Imagen 3: Sudoku solucionado mediante algoritmo de backtracking.

Además de ejecutar el algoritmo, se muestra por la salida de consola de NetBeans, o en el terminal en caso de ejecutarlo desde allí, el tiempo en nanosegundos que tarda en ejecutarse el algoritmo (Imagen 4).

```
run:
El tiempo tardado en ejecutar Backtracking ha sido: 18667065 ns.
```

Imagen 4: Tiempo de ejecución del algoritmo de Backtracking

B. AC3

Para la ejecución de este algoritmo se nos ha proporcionado una interfaz modificada que posee un nuevo botón al que podremos llamar para ejecutar el algoritmo AC3 (Imagen 5):

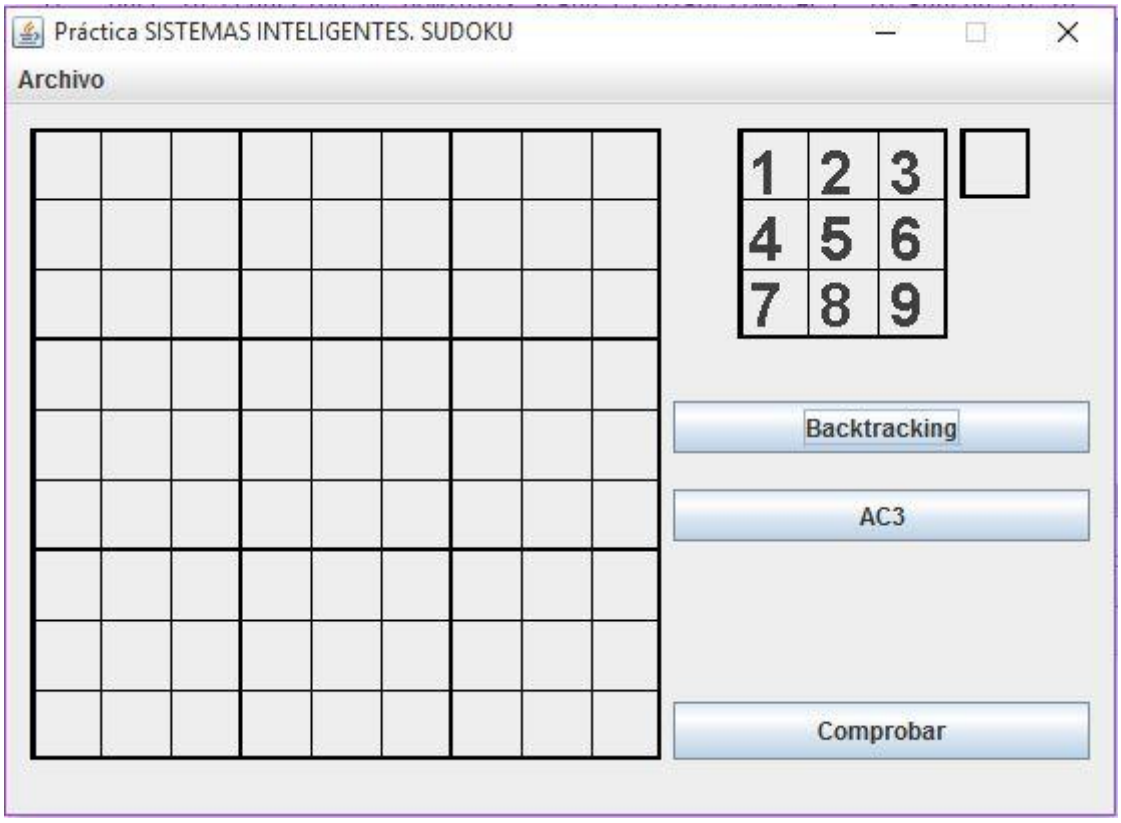


Imagen 5: Nueva interfaz proporcionada por el profesorado

El algoritmo AC3 es un algoritmo utilizado para reducir dominios. Este algoritmo mira, para cada una de las 81 casillas del tablero, si tiene o no un valor asignado. En el caso de tener un valor asignado, el dominio de la casilla habrá sido inicializado con únicamente dicho valor. En caso contrario, el algoritmo mirará las casillas con valores asignados en la fila, columna y submatriz correspondiente a la casilla actual y eliminará del dominio inicial de la casilla los valores que no cumplan las restricciones.

Este algoritmo no resuelve el problema, únicamente ejecuta la reducción de dominios para que después otro algoritmo, como el de Backtracking, sea ejecutado con los dominios reducidos.

En el caso de que cualquier casilla del tablero posea un dominio vacío, el algoritmo de backtracking, al ejecutar el módulo *compruebaCargado(Tablero tablero, Casilla[] casillas)*, que comprueba tanto que el tablero introducido no posea incongruencias como que ningún dominio sea vacío, devolverá falso al detectar dicho dominio vacío sin empezar a calcular la solución. Además de esto, se mostrará un mensaje especificando la causa de que no se haya ejecutado el algoritmo.

Al ejecutar el módulo principal, y tras realizar la reducción de dominios, se imprimirá el dominio de cada una de las casillas existentes por consola y el tiempo utilizado para ejecutar el algoritmo (Imagen 6):

```
Dominio de la casilla(5,3): [4,9]
Dominio de la casilla(5,4): [1,4,5,9]
Dominio de la casilla(5,5): [1,4,5,9]
Dominio de la casilla(5,6): [7]
Dominio de la casilla(5,7): [2,8,9]
Dominio de la casilla(5,8): [1,2,3,9]
Dominio de la casilla(6,0): [4,7,8]
Dominio de la casilla(6,1): [9]
Dominio de la casilla(6,2): [4,8]
Dominio de la casilla(6,3): [4,7]
Dominio de la casilla(6,4): [2]
Dominio de la casilla(6,5): [3,4,7]
Dominio de la casilla(6,6): [1,4,8]
Dominio de la casilla(6,7): [5]
Dominio de la casilla(6,8): [1,4,6]
Dominio de la casilla(7,0): [1]
Dominio de la casilla(7,1): [4,6,7,8]
Dominio de la casilla(7,2): [2]
Dominio de la casilla(7,3): [5]
Dominio de la casilla(7,4): [4,7,9]
Dominio de la casilla(7,5): [4,7,9]
Dominio de la casilla(7,6): [4,8]
Dominio de la casilla(7,7): [3]
Dominio de la casilla(7,8): [4,6]
Dominio de la casilla(8,0): [4,5,7]
Dominio de la casilla(8,1): [3]
Dominio de la casilla(8,2): [4,5]
Dominio de la casilla(8,3): [1]
Dominio de la casilla(8,4): [6]
Dominio de la casilla(8,5): [8]
Dominio de la casilla(8,6): [9]
Dominio de la casilla(8,7): [2,7]
Dominio de la casilla(8,8): [2,4]
El tiempo tardado en ejecutar AC3 ha sido: 11515320 ns.
```

Imagen 6: Dominios de las casillas y tiempo de ejecución de AC3 para el caso del sudoku anteriormente mostrado.

A continuación se muestran el módulo principal del algoritmo AC3 (Figura 4) y el método que realiza la reducción de dominios para cada casilla (Figura 5):

```
public boolean ejecutarAC(Tablero tablero) {

   //Rellena el array de casillas
   init(tablero);

   if (!compruebaCargado(tablero, casillas)) {
      return false;
   }
   try {
      long init;
      long fin;
      long total;
      init = System.nanoTime();
      //ejecución de la reduccion de dominios por AC3
      casillas = switchBackAC3(tablero, casillas, 0);
      fin = System.nanoTime();
```

```
            total = fin - init;
            System.out.println("El tiempo tardado en ejecutar AC3 ha sido: " + total + " ns.");

            int[] initial;
            int count;
            for (Casilla casilla : casillas) {
               count = 0;
               initial = casilla.getDom();
               System.out.print("Dominio de la casilla(" + casilla.getFil() + "," + casilla.getCol() + "): [");
               for (int j = 1; j < 10; j++) {
                  if (initial[j] != 0) {
                     if (count != 0) {
                        System.out.print(",");
                     }
                     System.out.print(j);
                     count++;
                  }
               }
               System.out.print("]");
               System.out.println();
            }
            buscadoAC3 = true;
         } catch (Exception ex) {
            System.err.println("El algoritmo AC3 ha tenido un problema: " + ex);
         }

         return true;
      }
```

Figura 4: Módulo principal del algoritmo AC3 llamado desde la interfaz

```
   public Casilla[] switchBackAC3(Tablero tablero, Casilla[] casillas, int pos) {

      if (pos == casillas.length) {
         return casillas;
      }
      int[] dom = casillas[pos].getDom();

      if (casillas[pos].getValor() == 0) {
         //Vector inicial vacío a rellenar que almacena los posibles valores que puede tomar la casilla
inicial.
         //La posición del vector será el valor, como el 0 no es una solución no se utilizará y se pondrá a 0
         int number;
         //Comprueba la fila
         for (int i = 0; i < 9; i++) {
            number = tablero.getCasilla(casillas[pos].getFil(), i);
            if (number != 0) {
               dom[number] = 0;
            }
         }
         //Comprueba la columna
         for (int i = 0; i < 9; i++) {
            number = tablero.getCasilla(i, casillas[pos].getCol());
            if (number != 0) {
```

```
            dom[number] = 0;
          }
        }

        //Comprueba la submatriz
        int indexfil;
        int indexcol;
        //Seleccion de submatriz
        //filas
        if (casillas[pos].getFil() < 3) {
          indexfil = 0;
        } else if (casillas[pos].getFil() < 6) {
          indexfil = 3;
        } else {
          indexfil = 6;
        }
        //columnas
        if (casillas[pos].getCol() < 3) {
          indexcol = 0;
        } else if (casillas[pos].getCol() < 6) {
          indexcol = 3;
        } else {
          indexcol = 6;
        }

        //Comprobación
        for (int i = 0; i < 3; i++) {
          for (int j = 0; j < 3; j++) {
            number = tablero.getCasilla(indexfil + i, indexcol + j);
            if (number != 0) {
              dom[number] = 0;
            }
          }
        }
      }

      casillas[pos].setDom(dom);

      return switchBackAC3(tablero, casillas, pos + 1);
    }
```

Figura 5: Módulo que realiza la reducción de dominios para cada casilla.

4. EXPERIMENTACIÓN

A continuación se detallarán los sudokus utilizados para comprobar el correcto funcionamiento de los algoritmos, así como una comparativa de ejecución entre Backtracking y Backtracking tras haber ejecutado AC3.
Los ejemplos utilizados para la comparación y comprobación de la ejecución son los siguientes:

3 1 2 4 0 0 0 0 8 7 4 0 0 9 3 6 0 0 0 6 0 1 0 8 0 0 0 1 8 0 0 6 7 0 5 4 5 7 4 3 0 2 0 6 0 6 2 9 5 0 1 8 0 0 8 0 1 0 2 0 0 3 0 0 3 6 8 1 0 5 0 0 2 0 0 6 0 4 1 0 9 sudoku1.txt	0 9 0 0 7 6 0 3 0 2 1 3 0 0 4 7 0 0 0 0 0 0 0 1 0 0 0 0 6 0 0 0 7 0 1 0 0 2 0 0 0 0 9 4 7 0 4 0 0 0 0 2 6 0 8 0 0 0 6 0 5 2 1 0 0 6 0 0 0 0 7 0 1 0 0 0 4 3 0 9 0 sudoku2.txt	0 0 6 0 0 0 0 1 8 9 0 0 0 0 0 0 6 7 0 0 7 0 0 0 2 0 5 0 0 0 3 8 0 6 4 0 0 1 0 2 0 0 5 0 0 6 0 0 0 0 0 7 0 0 0 9 0 0 2 0 0 5 0 1 0 2 5 0 0 0 3 0 0 3 0 1 6 8 9 0 0 sudoku3.txt
0 4 0 0 0 5 3 7 0 0 0 0 0 6 0 9 0 2 7 0 0 0 3 0 0 4 0 8 0 0 5 2 0 0 0 9 0 0 7 0 0 0 0 0 4 3 5 0 0 0 0 0 0 0 0 0 3 6 8 0 0 0 7 0 0 0 3 0 0 0 6 0 0 0 9 0 4 0 0 5 0 sudoku4.txt	0 1 0 0 0 3 0 0 0 0 0 4 0 0 0 3 0 0 0 9 0 7 0 2 0 5 0 0 0 9 5 0 7 4 0 0 5 0 0 0 0 0 0 0 1 0 0 2 6 0 4 7 0 0 0 2 0 9 0 5 0 1 0 0 0 3 0 0 0 2 0 0 7 0 0 0 4 0 0 0 8 facil1.txt	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 0 voiddomain.txt
1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Iniciomalo1.txt	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Iniciomalo2.txt	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Iniciomalo3.txt

Exceptuando “voiddomain.txt”, “iniciomalo1.txt”, “iniciomalo2.txt” e “iniciomalo3.txt”, el resto de sudokus han sido proporcionados por el profesorado.

El sudoku “facil1.txt” ejecuta el algoritmo de backtracking de manera secuencial. De modo que al probar todas las soluciones posibles para la primera casilla y no encontrar solución, devuelve falso, ya que no ha podido resolverse el algoritmo.

Aquí mostramos la ejecución del algoritmo bactracking con el archivo “facil1.txt” (Imagen 7):

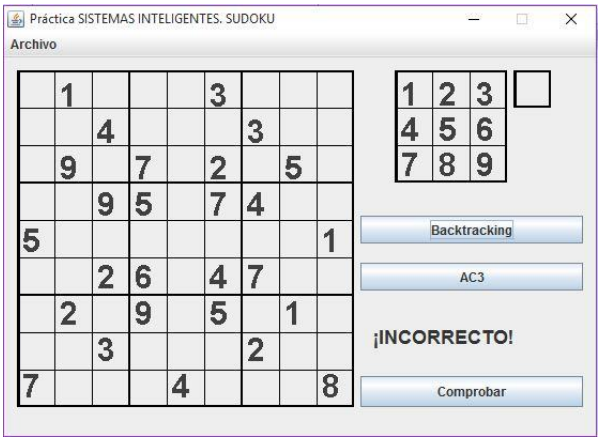


Imagen 7: Ejecución del sudoku “facil1.txt”

Podemos comprobar que, al no encontrar solución, el sudoku muestra incorrecto a la ejecución del programa, ya que no se ha podido solucionar ejecutando backtracking.

El sudoku “voiddomain.txt” posee un dominio vacío en su última casilla, por lo que, al ejecutar el algoritmo de backtracking comenzará a probar posibles soluciones al problema. Sin embargo, debido a que existen restricciones, además de un dominio vacío, nunca podrá llegar a la solución.

Si ejecutamos el algoritmo de backtracking con este sudoku, debido a la gran cantidad de combinaciones que debe realizar, el tiempo de ejecución del programa puede extenderse a días hasta que encuentre que no tiene solución. Sin embargo, si ejecutamos la reducción de dominios de AC3, el algoritmo de backtracking terminará antes de empezar a calcular debido a que, como hemos explicado anteriormente, se detectará el dominio vacío y no se ejecutará el algoritmo de backtracking.

A continuación mostramos la ejecución del algoritmo de backtracking tras haber ejecutado AC3, mostrando el dominio de las variables (Imagen 8) y el resultado mostrado en la interfaz (Imagen 9):

```
Dominio de la casilla(5,5): [1,2,3,4,5,7,8,9]
Dominio de la casilla(5,6): [1,2,3,4,5,6,8,9]
Dominio de la casilla(5,7): [1,2,3,4,5,6,7,9]
Dominio de la casilla(5,8): [1,2,3,4,5,6,7,8]
Dominio de la casilla(6,0): [4,5,6,7,8,9]
Dominio de la casilla(6,1): [4,5,6,7,8,9]
Dominio de la casilla(6,2): [4,5,6,7,8,9]
Dominio de la casilla(6,3): [1,2,3,7,8,9]
Dominio de la casilla(6,4): [1,2,3,7,8,9]
Dominio de la casilla(6,5): [1,2,3,7,8,9]
Dominio de la casilla(6,6): [1,2,3,4,5,6]
Dominio de la casilla(6,7): [1,2,3,4,5,6]
Dominio de la casilla(6,8): [1,2,3,4,5,6]
Dominio de la casilla(7,0): [4,5,6,7,8]
Dominio de la casilla(7,1): [4,5,6,7,8]
Dominio de la casilla(7,2): [4,5,6,7,8]
Dominio de la casilla(7,3): [1,2,3,7,8]
Dominio de la casilla(7,4): [1,2,3,7,8]
Dominio de la casilla(7,5): [1,2,3,7,8]
Dominio de la casilla(7,6): [1,2,3,4,5,6]
Dominio de la casilla(7,7): [1,2,3,4,5,6]
Dominio de la casilla(7,8): [9]
Dominio de la casilla(8,0): [1]
Dominio de la casilla(8,1): [2]
Dominio de la casilla(8,2): [3]
Dominio de la casilla(8,3): [4]
Dominio de la casilla(8,4): [5]
Dominio de la casilla(8,5): [6]
Dominio de la casilla(8,6): [7]
Dominio de la casilla(8,7): [8]
Dominio de la casilla(8,8): []
El tiempo tardado en ejecutar AC3 ha sido: 19612584 ns.
Existe un dominio vacío, por lo que no hay solución.
El tiempo tardado en ejecutar Backtracking ha sido: 215527 ns.
```

Imagen 8: Dominio de las variables de “voiddomain.txt”

Imagen 9: Ejecución de Backtracking tras ejecutar AC3 en “voiddomain.txt”

En la imagen 8 podemos observar que el último dominio mostrado es vacío, por tanto no existe solución posible. Además, al ejecutar el Backtracking podemos ver el mensaje de error que nos dice que no se ha podido ejecutar el algoritmo debido a que existe un dominio vacío.

Los tres sudokus “iniciomalo1.txt”, “iniciomalo2.txt”e “iniciomalo3.txt”, son sudokus creados con el simple fin de comprobar que cuando se introduce un sudoku con incongruencias, es decir, un sudoku que no cumple las condiciones para la resolución del mismo, no se ejecuta el algoritmo de backtracking, mostrando una salida que explica el por qué no se puede ejecutar (Imagen 10):

```
El tablero inicial posee un valor repetido en una fila, columna o submatiz, por lo que no hay solución.
El tiempo tardado en ejecutar Backtracking ha sido: 73553 ns.
```

Imagen 10: Salida por consola del error que impide la ejecución del backtracking.

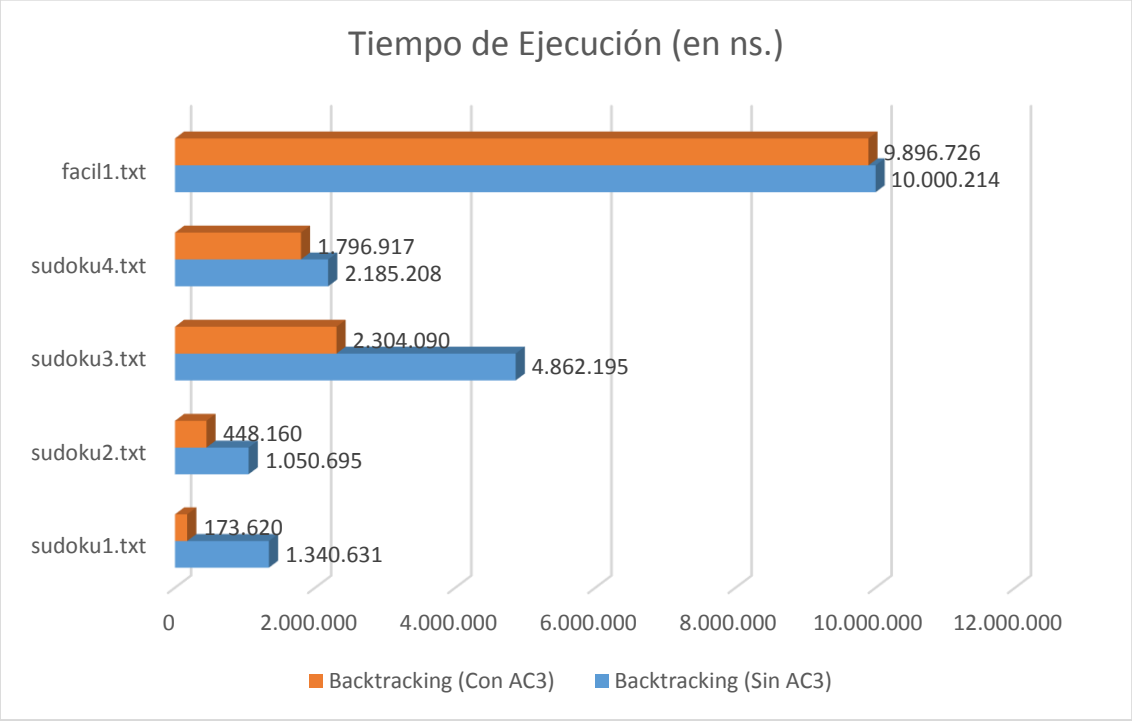
Finalmente mostramos una tabla comparativa con los tiempos de ejecución de los diferentes sudokus, antes y después de ejecutar el algoritmo AC3.

Sudoku	**Backtracking (Sin AC3)**	**Backtracking (Con AC3)**
sudoku1.txt	1.340.631	173.620
sudoku2.txt	1.050.695	448.160
sudoku3.txt	4.862.195	2.304.090
sudoku4.txt	2.185.208	1.796.917
facil1.txt	10.000.214	9.896.726
voiddomain.txt	∞	-
iniciomalo1.txt	-	-
iniciomalo2.txt	-	-
iniciomalo3.txt	-	-

Los tiempos mostrados en la tabla anterior están expresados en nanosegundos.
En aquellas casillas con un “-” en lugar de una cifra, el algoritmo backtracking no se ha llegado a ejecutar debido a algún error de los mostrados anteriormente.

Como hemos comentado anteriormente, “facil1.txt” no tiene ni dominios vacíos ni incongruencias al inicio, sin embargo no posee una solución que cumpla con los requisitos del problema, por tanto la ejecución tarda más que en los casos anteriores.

Por último mostramos una gráfica comparativa con los datos proporcionados en la tabla anterior. En el siguiente gráfico únicamente mostramos los datos de los sudokus que no han tenido errores en alguna de sus ejecuciones para poder ver la mejora temporal.



5. BIBLIOGRAFÍA

- Tema 4 de teoría de Sistemas Inteligentes: Búsqueda CSP - Materiales del Campus Virtual.