



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico

## Sudoku

23 de diciembre de 2015

Metaheurísticas  
2do Cuatrimestre de 2015

| Integrante                   | LU     | Correo electrónico  |
|------------------------------|--------|---------------------|
| Kujawski, Kevin              | 459/10 | kevinkuja@gmail.com |
| Ortiz de Zarate, Juan Manuel | 45/10  | jmanuoz@gmail.com   |

| Instancia       | Docente | Nota |
|-----------------|---------|------|
| Primera entrega |         |      |
| Segunda entrega |         |      |



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

## Índice

|  |           |
|--|-----------|
| <b>1. Introducción</b>                               | <b>2</b>  |
| <b>2. Descripción y formulación matemática</b>       | <b>2</b>  |
| <b>3. Descripción de la Solución</b>                 | <b>3</b>  |
| 3.1. Simulated annealing . . . . .                   | 3         |
| 3.1.1. Definición . . . . .                          | 3         |
| 3.1.2. Implementación para resolver Sudoku . . . . . | 3         |
| <b>4. Experimentos</b>                               | <b>8</b>  |
| 4.1. Simulate annealing . . . . .                    | 8         |
| <b>5. Conclusiones</b>                               | <b>10</b> |
| <b>6. Referencias</b>                                | <b>10</b> |

## 1. Introducción

Sudoku es un juego de lógica cuyo objetivo es rellenar una cuadrícula de tamaño 9x9, divididas a su vez en 9 cajas de 3x3, estas últimas llamadas cajas. Inicialmente contiene cierta cantidad de celdas con valores fijos pre-definidos y válidos, es decir, sus valores están entre [1,9], no hay repetidos por filas, columnas o cajas.

Las reglas del juego son:

1. Cada fila debe contener todos los números en el intervalo [1,9]
2. Cada columna debe contener todos los números en el intervalo [1,9]
3. Cada caja debe contener todos los números en el intervalo [1,9]

El objetivo del juego es rellenar las celdas en blanco de la cuadrícula inicial respetando las reglas del juego. A pesar de la simpleza del objetivo y las reglas del juego, hay 6670903752021072936960 formas posibles de ser rellenada una cuadrícula. Buscar una solución intentando todas las formas posibles es claramente imposible, y eso incentiva la utilización de una metaheurística para solucionar el juego.

En el presente trabajo, presentaremos una posible formulación matemática del Sudoku como un problema de optimización e dos propuestas de implementación utilizando dos metaheurísticas: Simulated annealing y Colonia de hormigas

## 2. Descripción y formulación matemática

Nuestra formulación matemática es una adaptación de (2). En esa formulación, las variables  $X_{ijk}$  son variables de decisión, definidas de la siguiente manera:

$$X_{ijk} = \begin{cases} 1 & \text{si el elemento (i,j) de la cuadrícula contiene el valor k} \\ 0 & \text{en caso contrario} \end{cases}$$

La formulación como programación lineal entera es la siguiente:

$$\begin{aligned} \min \quad & \text{FuncionCosto}(X) \\ \text{sujeto a} \quad & \sum_{i=1}^9 X_{ijk} = 1, j = 1 : 9, k = 1 : 9 \end{aligned} \quad (1)$$

$$\sum_{j=1}^9 X_{ijk} = 1, i = 1 : 9, k = 1 : 9 \quad (2)$$

$$\sum_{j=3q-2}^{3q} \sum_{i=3p-2}^{3p} X_{ijk} = 1, k = 1 : 9, p = 1 : 3, q = 1 : 3 \quad (3)$$

$$\sum_{k=1}^9 X_{ijk} = 1, i = 1 : 9, j = 1 : 9 \quad (4)$$

$$X_{ijk} = 1 \forall (i, j, k) \in \text{INICIALES} \quad (5)$$

$$X_{ijk} \in \{0, 1\}$$

Hay que notar que como se trata de un problema de satisfacibilidad, la formulación no necesita de una función objetivo, por eso la definimos como 0. Las restricciones (1),(2) y (3) garantizan que cada número en el intervalo posible de la instancia solo aparezca una vez en cada columna, fila, y caja, respectivamente. La restricción (4) garantiza que todas las posiciones de la matriz estén rellenas. La restricción (5) fuerza que las variables fijas de la instancia permanezcan sin alterar.

### 3. Descripción de la Solución

#### 3.1. Simulated annealing

##### 3.1.1. Definición

Simulated annealing (Recocido Simulado en español), respecto a las ciencias de la computación, es una metaheurística para problemas de optimización global, aplicándose a la heurística de Búsqueda Local Aleatoria. La técnica fue presentada por Kirkpatrick, Gelatt y Vecchi (1982) y ha sido muy empleado para resolver problemas combinatorios. La idea surge del proceso físico conocido como recocido en el cual, se eleva la temperatura de un sólido hasta el punto que se vuelve líquido, a continuación la temperatura se disminuye de forma paulatina para obtener una estructura cristalina sin defectos y que puede considerarse como un estado de mínima energía. Cada descenso de temperatura debe ser lo suficientemente pequeño para que el sistema no adquiera una estructura cristalina con defectos, además el sistema debe permanecer un tiempo suficiente a una misma temperatura para permitir alcanzar un estado estacionario, en otras palabras, que las partículas vuelvan a acomodarse.

Se parte de una solución inicial posible  $S_i$ , luego se selecciona una solución posible  $S_j$  dentro de una vecindad, enseguida se evalúa la calidad de la solución posible empleando una función de costo  $f(x)$  asociada a cada posible solución  $S$  del problema.

Si la nueva solución posible  $S_j$  es mejor que la actual (de acuerdo al costo), se acepta, de lo contrario se selecciona de acuerdo a una probabilidad, para Simulated annealing dicha probabilidad de selección está dada por la ecuación (1) que se conoce como el criterio de Metropolis; a  $temp$  se le conoce como el parámetro de control; este valor se inicia con un valor suficientemente grande para que cualquier solución tenga una probabilidad alta de ser seleccionada, a medida que  $temp$  disminuye, la probabilidad de aceptar soluciones factibles de mala calidad disminuye. El enfriamiento se realiza empleando el sistema geométrico dado por la ecuación (2), donde  $\alpha$  es un parámetro fijo.

$$criterio(S_i, S_j) = \exp((costo(S_i) - costo(S_j))/temp) \quad (1)$$

$$temp_{k+1} = temp_k * \alpha \quad (2)$$

##### 3.1.2. Implementación para resolver Sudoku


Antes de ir directo al algoritmo hay que tener en cuenta las diferentes elecciones que hicimos para encontrar una solución inicial, mejorarla y decidir cuando una solución es mejor que otra.

**Solución inicial** Como el algoritmo así lo requiere, debemos partir de una solución inicial para intentar mejorarla y en el mejor de los casos poder resolverla. Una primera opción que se puede pensar es, a partir de las celdas iniciales, rellenar todas las celdas vacías con valores aleatorios directamente. Sin descartar totalmente esta idea fuimos un poco más allá. Básicamente intentamos resolver el Sudoku lo más que podamos, estando siempre seguro de que los valores que pongamos en las celdas son los correctos. Para esto analizamos los posibles valores que pueden ir en cada celda, eliminando de esas posibilidades valores que estén en la misma fila, columna o caja. Luego de esto, en caso de detectar que una celda tiene un solo valor posible, se rellenará la celda con ese valor y se lo considerará correcto ya que fue deducido de la configuración inicial. En caso de que no haya valores posibles para esa celda, indicará que esa configuración inicial no tiene solución posible. Por cada celda que se rellene se deberá recalcular todo el proceso de nuevo, hasta que en algún ciclo no haya modificaciones.

Una vez que se terminó de deducir lo máximo posible será el momento de rellenar las celdas que quedaron vacías. Para este punto intentaremos ser lo menos aleatorio posible, y para tal caso seleccionaremos entre los posibles valores que pueden tener la celda, que ya habíamos calculado

en el punto anterior. Y en caso de que para una celda ya no haya posibles valores, es el momento en que se le asignará uno aleatorio, siempre respetando la formulación (3) que se menciona en la sección anterior, es decir, se respetará en todo momento que no haya celdas inválidas en las cajas.

Para dar una idea más gráfica de como se intenta resolver el sudoku en "primera fase.<sup>a</sup> partir del caso inicial tenemos las dos siguientes figuras. En la primera se observa un primer barrio total en donde a cada celda le asignamos los posibles valores, donde luego, al encontrar una celda donde solo tenga un único posible valor (celda con estrella roja) se lo asigna y se vuelve a calcular a partir de ese punto. Luego, en la figura 2 se ve como a partir de establecer el valor en la figura 1 desencadena (circulo celeste) que quede otra celda con un único valor posible (flecha amarilla) al cual como en el paso anterior se le asignará y se volverá a recalcular.

|        |       |      |      |    |      |   |       |       |
|--------|-------|------|------|----|------|---|-------|-------|
|        |       | 1    | 9    |    |      |   |       | 8     |
| 2345   | 245   |      |      | 23 | 237  | 567   | 4567  |       |
| 6      |       |      |      | 8  | 5    |  | 3     |       |
|        | 249   | 29   | 1247 |    |      | 7   |       | 249   |
|        |       | 7    |      | 6  |      | 1   |       |       |
| 234589 | 24689 |      | 234  |    | 23   |   | 459   | 2469  |
|        | 3     | 4    |      | 9  |      |   |       |       |
| 1258   |       |      | 267  |    | 2678 | 568   | 15678 | 156   |
|        |       |      | 5    |    | 4    |   |       |       |
| 1289   | 26789 | 2689 |      | 23 |      | 368   | 16789 | 1369  |
|        |       |      |      | 1  |      | 4   | 2     |       |
| 589    | 56789 | 689  | 367  |    | 3678 |   |       | 3569  |
|        |       | 5    |      | 7  |      | 9   |       |       |
| 2348   | 2468  |      | 1236 |    | 1236 |   | 1468  | 1346  |
|        | 1     |      | 8    | 4  |      |   |       | 7     |
| 239    |       | 2369 |      |    | 236  | 356   | 56    |       |
| 7      |       |      |      |    | 9    | 2   |       |       |
|        | 468   | 368  | 136  | 35 |      |   | 14568 | 13456 |

|        |       |      |      |    |      |     |       |       |
|--------|-------|------|------|----|------|-----|-------|-------|
|        |       | 1    | 9    |    |      |     |       | 8     |
| 2345   | 245   |      |      | 23 | 237  | 567 | 4567  |       |
| 6      |       |      |      | 8  | 5    | ★   | 3     |       |
|        | 249   | 29   | 127  |    |      | 7   |       | 249   |
|        |       | 7    |      | 6  |      | 1   |       |       |
| 234589 | 24589 |      | 234  |    | 23   |     | 459   | 2459  |
|        | 3     | 4    |      | 9  |      |     |       |       |
| 1258   |       |      | 267  |    | 2678 | 568 | 15678 | 156   |
|        |       |      | 5    |    | 4    |     |       |       |
| 1289   | 26789 | 2689 |      | 23 |      | 368 | 16789 | 1369  |
|        |       |      |      | 1  |      | 4   | 2     |       |
| 589    | 56789 | 689  | 367  |    | 3678 |     |       | 3569  |
|        |       | 5    |      | 7  |      | 9   |       |       |
| 2348   | 2468  |      | 1236 |    | 1236 |     | 1468  | 1346  |
|        | 1     |      | 8    | 4  |      |     |       | 7     |
| 239    |       | 2369 |      |    | 236  | 356 | 56    |       |
| 7      |       |      |      |    | 9    | 2   |       |       |
|        | 468   | 368  | 136  | 35 |      |     | 14568 | 13456 |

**Función de costo** El costo (o penalización) va a depender de la cantidad de celdas ilegales” que tenga la cuadrícula. Es decir, cuando haya una celda que se repita en la fila, columna o caja se sumará una penalidad. El caso ideal, que indica que el tablero está solucionado, es cuando el costo devuelve 0.

|                   |       |                   |   |       |       |       |   |
|-------------------|-------|-------------------|---|-------|-------|-------|---|
| 9 6 3             | 1 7 4 | 4 5 8             | 1 | 9 6 3 | 1 7 4 | 2 5 8 | 0 |
| 1 7 8             | 3 2 5 | 6 2 9             | 1 | 1 7 8 | 3 2 5 | 6 4 9 | 0 |
| 2 5 4             | 6 9 8 | 7 3 1             | 0 | 2 5 4 | 6 8 9 | 7 3 1 | 0 |
| 8 2 1             | 4 3 7 | 7 9 6             | 1 | 8 2 1 | 4 3 7 | 5 9 6 | 0 |
| 4 9 6             | 8 5 2 | 3 1 5             | 1 | 4 9 6 | 8 5 2 | 3 1 7 | 0 |
| 7 3 5             | 9 6 1 | 8 2 4             | 0 | 7 3 5 | 9 6 1 | 8 2 4 | 0 |
| 5 9 8             | 7 4 3 | 4 6 2             | 1 | 5 8 9 | 7 1 3 | 4 6 2 | 0 |
| 3 1 7             | 2 1 6 | 9 8 5             | 1 | 3 1 7 | 2 4 6 | 9 8 5 | 0 |
| 6 4 2             | 5 9 8 | 1 7 3             | 0 | 6 4 2 | 5 9 8 | 1 7 3 | 0 |
| 0 1 1 0 1 1 2 0 1 | 13    | 0 0 0 0 0 0 0 0 0 | 0 |       |       |       |   |

**Solución vecina** Partiendo de una posible solución parcial, se procede a generar una solución vecina diferente pero a su vez cercana. Para ello realizamos entre 1 y 9 cambios de celdas, elegidas de manera aleatoria, siempre intercambiando entre la misma caja para seguir asegurando la formulación (3), siempre y cuando las celdas no sean las iniciales o las correctas. Estos "pequeños cambios", son parte de la búsqueda local, y como se verá más adelante, puede no ser descartada como camino a seguir.

**Párametros de control** El algoritmo consta de 2 parámetros de control, uno es *temp* que representa la temperatura inicial y  $\alpha$  que representa el factor de enfriamiento de la temperatura. En base a sucesivas pruebas determinados que los valores para los cuales obtuvimos mejores resultados son con *temp* entre 0.9 y 1, y  $\alpha$  entre 0.9 y 0.99, lo que significa un enfriamiento bastante lento. Además se cuenta con un parámetro para marcar la cantidad de iteraciones máximas, en este punto no llegamos a un valor ideal, pero notamos que después de 100 mil iteraciones, el resultado difícilmente sigue mejorando.

### Algoritmo

#### Algorithm 1 Sudoku con Simulated annealing

```

1: function SASUDOKUSOLVER(Inicio,  $\alpha$ , MaxIter, Temp0 )
2:   sol0  $\leftarrow$  generarSolucionInicial(inicio)
3:   solucionActual  $\leftarrow$  mejorSudoku  $\leftarrow$  sol0
4:   costoActual  $\leftarrow$  mejorCosto  $\leftarrow$  costo(mejorSudoku)
5:   Mientras no exceda las iteraciones máximas y no haya encontrado una solución
6:     nuevaSol  $\leftarrow$  busquedaLocal(solucionActual)  $\triangleright$  Se busca solución vecina
7:     nuevoCosto  $\leftarrow$  costo(nuevaSol)  $\triangleright$  Se calcula el nuevo costo
8:     aceptacion  $\leftarrow$  Rand(0,1) < exp((costoActual - nuevoCosto)/temp)  $\triangleright$  Se calcula la
       aceptación, con un valor aleatorio entre (0,1)
9:     Si se acepta como la solución actual y no fue usada antes
10:       solucionActual  $\leftarrow$  nuevaSol
11:       costoActual  $\leftarrow$  nuevoCosto
12:       Marcar la solución como ya usada
13:     Fin Si
14:     Si el costo de la actual es menor que la mejor solución
15:       mejorSudoku  $\leftarrow$  solucionActual
16:       mejorCosto  $\leftarrow$  costoActual
17:       temp  $\leftarrow$  Temp0  $\triangleright$  Reiniciamos la temperatura
18:     Fin Si
19:     Si el costo de la nueva solución vecina es 0
20:       Devolver nuevaSol  $\triangleright$  Devolvemos esa solución del Sudoku
21:       Finalizamos la función
22:     Fin Si
23:     temp  $\leftarrow$  Máximo entre 0.1 y  $\alpha$  * temp  $\triangleright$  Le aseguramos un piso mínimo a la
       temperatura
24:   Fin Mientras
25:   Devolver mejorSudoku  $\triangleright$  Devolvemos la mejor solución del Sudoku que hayamos
     encontrado
26: end function

```

end

#### Algorithm 2 Búsqueda local

```

1: function BUSQUEDALOCAL(Solucion)
2:   Repetir entre 1 y 9 veces
3:     celdaAleatoria  $\leftarrow$  seleccionarCeldaAleatoria(Solucion)  $\triangleright$  Seleccionar una celda
       aleatoria

```

```

4:      otraCelda ← seleccionarOtraCeldaAleatoria(celdaAleatoria) ▷ Seleccionar otra
      celda aleatoria del mismo cuadrante
5:      Si ninguna de las dos celdas seleccionadas está protegida
6:          Swap(Solucion, celdaAleatoria, otraCelda) ▷ Intercambiar celdas
7:      Fin Si
8:  Fin del ciclo
9:  Devolver solución vecina
10: end function
end

```

**Algorithm 3** Generar solución inicial

```

1: function GENERARSOLUCIONINICIAL(SudokuInicial)
2:   Mientras haya cambios
3:     Por cada celda del suduoku
4:       Recorrer toda su fila y toda su columna recolectando los valores
5:       Intersectar la recolección con los valores del 1 al 9 ▷ Para calcular los posibles
       valores de la fila
6:       Si sólo queda un posible valor
7:         Asignarselo a la celda
8:         Proteger la celda
9:         Marcar que hubo cambio
10:      Fin Si
11:    Fin del ciclo
12:  Fin Mientras
13:  Por cada celda vacia
14:    Seleccionar uno de los posibles valores que puede tener la celda
15:    En caso de que no tenga, elegir un valor aleatoria que falte en la misma caja que
    se encuentra la celda
16:  Fin del ciclo
17: end function
end

```



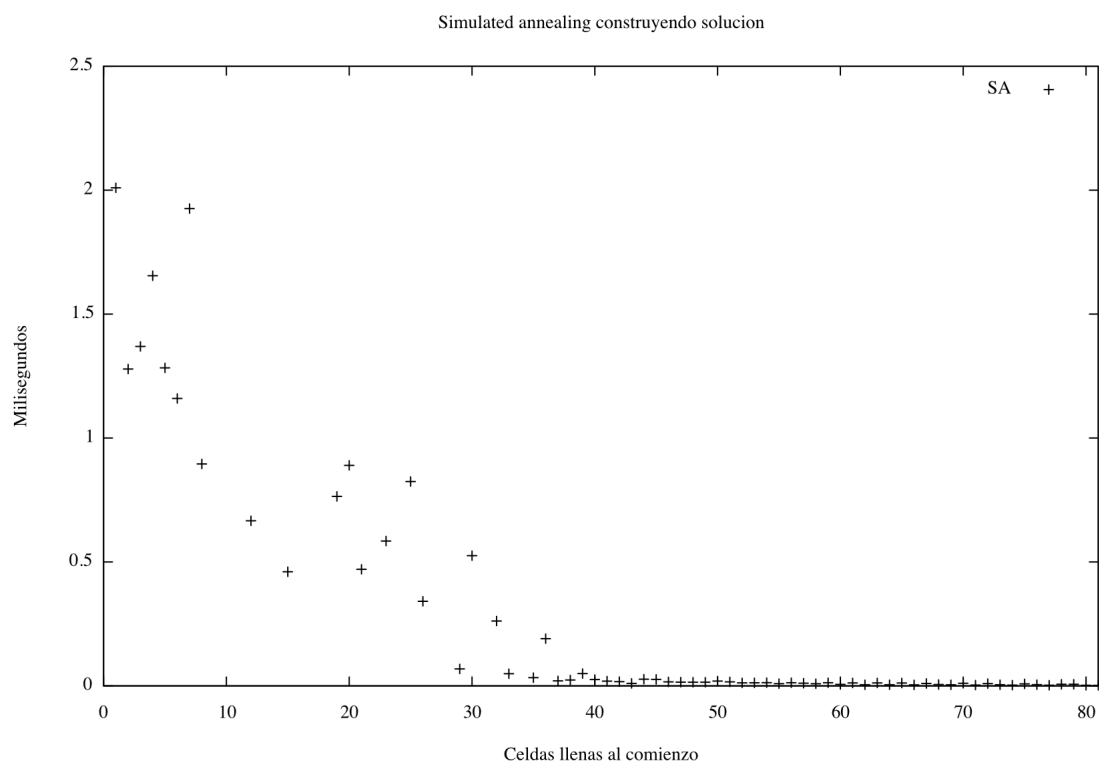
|                     | Simulated Annealing | Colonia de Hormigas |
|---------------------|---------------------|---------------------|
| Escenarios probados | 115                 |                     |
| Solucionados        | 70 (60 %)           |                     |
| Dificultad baja     | 71 %                |                     |
| Dificultad moderada | 40 %                |                     |
| Dificultad alta     | 16 %                |                     |

## 4. Experimentos

### 4.1. Simulate annealing

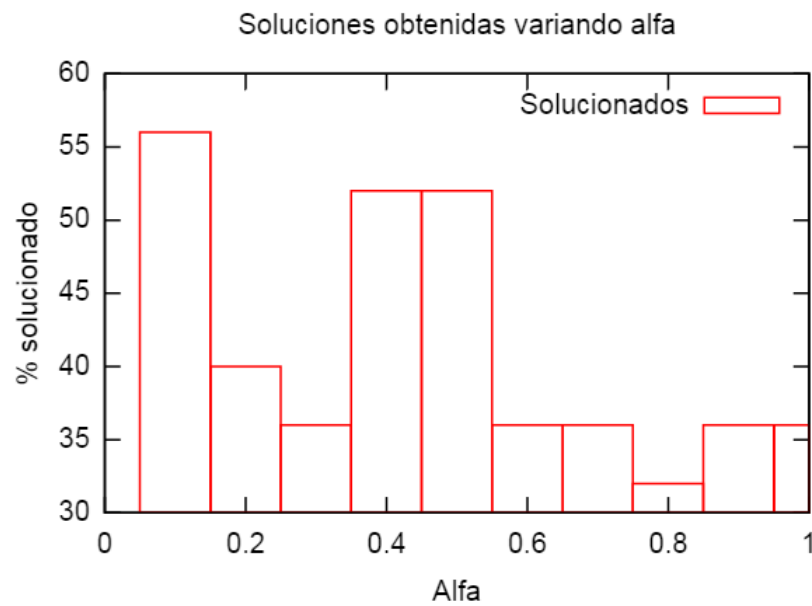
Para los experimentos elegimos un factor de enfriamiento en 0.5 y una temperatura inicial de 1. La justificación es en base a los experimentos realizados y que mostraremos en esta sección.

El siguiente grafico es el resultado del tiempo que toma solucionar un Sudoku en donde partiendo de una solución existente blanqueamos celdas random y observamos como lo soluciona el algoritmo:



Claramente se puede observar que a medida que el tablero esta mas lleno el tiempo es menor, eso es debido a la suma de varios factores, como que inicialmente infiere más celdas, son menos las vacias (que se convierte en menos iteraciones) y menos búsqueda de soluciones vecinas.

A continuación corrimos el algoritmo con 25 soluciones de diferentes niveles de dificultad, variando el factor de enfriamiento (o alfa):



Si bien con valores intermedios (aprox 0.5) es donde se concentran las mejores soluciones, y que la mayor cantidad se logra con un 0.1, creemos en base a anteriores pruebas que el factor de enfriamiento debería ser siempre mayor de 0.5 para que sea algo "lento" no siempre elija ir por soluciones vecinas pero que tampoco las restrinja totalmente.

## 5. Conclusiones

## 6. Referencias

1. B. Felgenhauer, e F. Jarvis. (2006, January). Mathematics of Sudoku I.
2. A. Bartlett, T. Chartier, A. N. Langville, e T. Rankin. (2008). An Integer Programming Model for the Sudoku Problem. *Journal of Online Mathematics and its Applications* MAA, (8):1-14.
3. Algunas soluciones sacadas de <http://es.websudoku.com>