DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Sudoku

28 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

1. Introducción

Sudoku es un juego de lógica cuyo objetivo es rellenar una cuadricula 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 cuadricula inicial respetando las reglas del juego. A pesar de las simpleza del objetivo y las reglas del juego, hay 6670903752021072936960 formas posibles de ser rellenada una cuadricula. 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:

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

La formulación como programación lineal entera es la siguiente:

$$\begin{aligned}
\min \quad & \text{FuncionCosto(X)} & \\
\text{sujeto a} \quad & \sum_{i=1}^{9} X_{ijk} = 1, j = 1:9, k = 1:9 & (1)\\
& \sum_{j=1}^{9} X_{ijk} = 1, i = 1:9, k = 1:9 & (2)\\
& \sum_{j=3q-2}^{3q} \sum_{i=3p-2}^{3p} X_{ijk} = 1, k = 1:9, p = 1:3, q = 1:3 & (3)\\
& \sum_{k=1}^{9} X_{ijk} = 1, i = 1:9, j = 1:9 & (4)\\
& X_{ijk} = 1 \forall (i,j,k) \in INICIALES & (5)\\
& X_{ijk} \in 0, 1 &
\end{aligned}$$

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 numero 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 martiz esten rellenas. La restricción (5) fuerza que las variables fijas de la instancia permanezcas sin alterar.

2.1. Hormigas

En un principio intentamos resolver este problema con la metaheurística colonia de hormigas. Pero los resultados que obteníamos no eran buenos. Visto esto optamos por hacer algunos cambios en el algoritmo original para ver si podíamos optimizar las soluciones. Efectivamente logramos mejorar, a esta nueva metaheurística la llamamos Hormigas. Por el hecho de que las hormigas no salen en grupo sino individualmente. A continuación explicamos como quedó el proceso con nuestros cambios.

Clase Hormigas

```
function RESOLVER(Inicio )
    feromonas ← new Feromonas()
    for 1 to 400 do
        sudoku = new Sudoku()
        hormiga = new Hormiga(sudoku,feromonas, probaSeguiFeromona)
        if la iteración es multiplo de 6 then
            feromonas→evaporar()
        end if
        if hormiga→resolver() then
            devolver true
        end if
    end for
    Devolver false
end function
```

Clase Hormiga

```
function RESOLVER( )
    casillas = sudoku → obtenerCasillasSinValor()   ▷ Obtengo las casillas que aun no tienen valor seteado ordenadas por cantidad de valores posibles a insertar
    while casilla = casillas→obtener do
        seMovio = seguirFeromona(casilla)
        if no seMovio then
            seMovio = elegirRandom()
            if no seMovio then
                Devolver false   ▷ si no se movio quiere decir que no era posible insertar ningún valor, por lo tanto este camino no tiene solución
            end if
        end if
        casillas = sudoku → obtenerCasillasSinValor()   ▷ Vuelvo a obtener las casillas, porque si inserte un valor cambian los posibles de las muchas casillas
    end while
    Devolver true   ▷ Si pude insertar valores en todas las que estaban vacias, entonces resolvi el sudoku
end function

function SEGUIRFEROMONA(casilla )
    valoresConFeromonas = feromonas→ obtenerFeromonas(casilla)   ▷ obtengo cada valor posible para esa casilla y cuanta feromona hay depositada en cada uno
    while valor = valoresConFeromonas→obtener do
        probaMoverse = random(0,probaSeguiFeromona)
        if ( then (valor→feromona ∗ probaMoverse)   100)
            moverse(casilla,valor)
            eliminarUltimaFeromonaDepostada()   ▷ elimino la ultima feromona depositada
```

```
porque seguro el ultimo valor insertado era incorrecto
            Devolver true
        end if
    end while
    Devolver false
end function

function elegirRandom(casilla )
    posiblesValores→obtenerPosiblesValores()
    if posiblesValores no es vacio then
        valor = posiblesValores→elegirAlAzar()
        moverse(casilla,valor)
        Devolver true
    end if
    Devolver false
end function

function moverse(casilla,valo )
    feromonas→depositarFeromona(casilla,valor)
    sudoku→insertarValor(casilla,valor)
end function
```

Coloquialmente lo que hace este algoritmo es, dado un sudoku incial lanzar 400 hormigas a intentar resolverlo. Cada una recorre las casillas vacias, comenzado por las que menos posibiles valores a insertar tienen (según las reglas del sudoku antes mencionadas). Para cada casilla se fija si alguno de los valores posibles tienen feromonas depositadas y en base a un calculo probabilistico inserta ese valor o no. Si no, elige uno de los posibles valores al azar y deposita una feromona en el mismo. Depositar una feromona consiste en incrementar en una unidad la probabilidad de que en esa casilla se elija ese valor.

Cada hormiga intenta solucionar el sudoku inicial pero utilizando las feromonas insertadas por sus predecesoras. Cada 6 hormigas, se produce la evapotación de las feromonas, este proceso consiste en restarle una unidad a cada valor de cada casilla (excepto que el mismo sea 0). Decidimos que sea cada 6 porque probando distintos sudokus vimos que aumentar este número mejoraba algunas soluciones pero empeoraba otras y lo mismo sucedía la inversa, 6 era el punto intermedio.

Son 400 porque probando varias veces el algoritmo y viendo que numero de hormiga es la que solucionaba el sudoku el mayor numero obtenido fué 350, por lo tanto 400 nos pareció una buena cota que no limite el llegar a la solución y a la vez tampoco aumente el computo innecesariamente.

Por último el valor "probaSeguiFeromona" utilizado fué 2 porque también en base a las pruebas realizadas fue el que mejor equilibrio establecía para llegar a las soluciones en disintos sudokus (entre 1 y 8 variaba su efectividad entre los sudokus, a partir de este valor directamente no llegaba nunca a una solución)

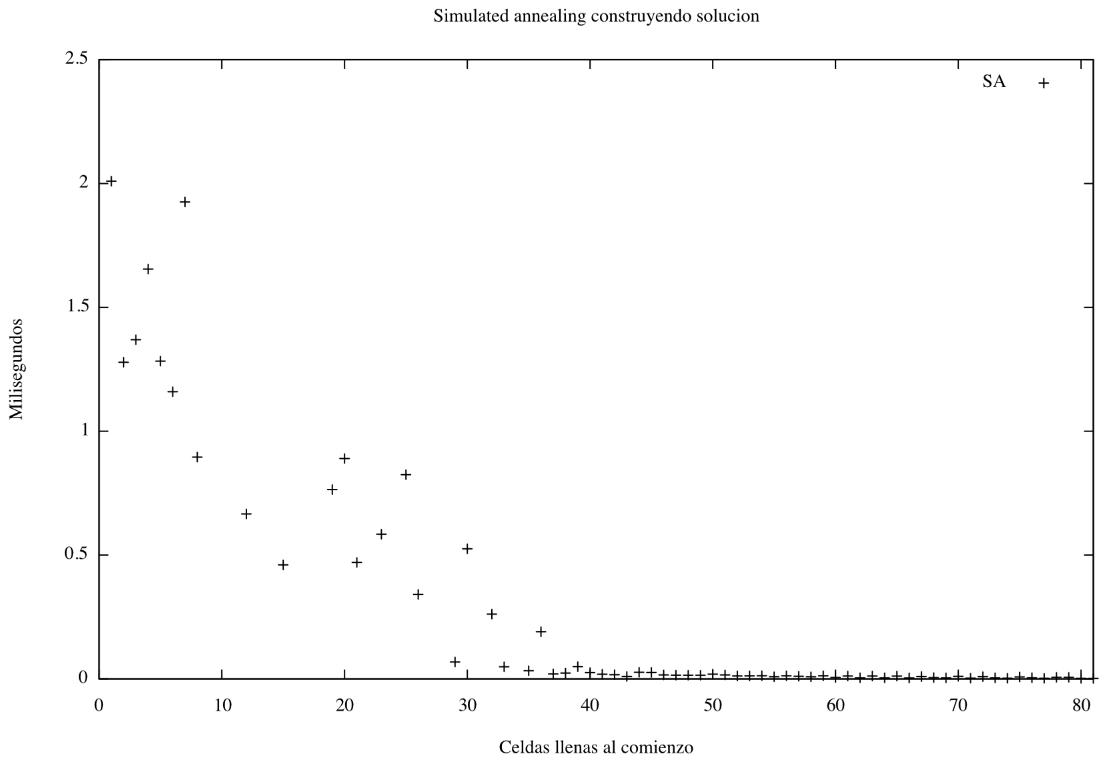
	Simulated Annealing	**Colonia de Hormigas**
Escenarios probados	115	
Solucionados	70 (60 %)	
Dificultad baja	71 %	
Dificultad moderada	40 %	
Dificultad alta	16 %	

3. Experimentos

3.1. Simulate annealing

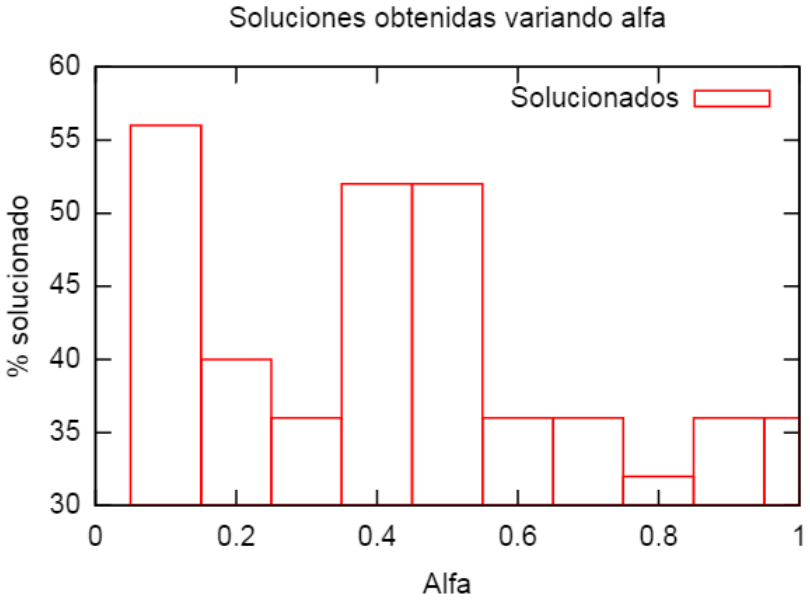
Para los experimentos elegimos un factor de enfriamiento en 0.5 y una temperatura inicial de 1. La justificación es en base a los experimientos realizados y que mostraremos en esta sección.

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:



Clarmente 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 el factor de enfriamiento debería ser siempre mayor de 0.5 para que sea algo "lentoz no siempre elija ir por soluciones vecinas pero que tampoco las restrinja totalmente.

4. Conclusiones

5. 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 Pro- gramming 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