Optimización basada en colonia de hormigas

Aplicación sudoku

DANIELA VELÁSQUEZ

Índice

1. Introducción

El sudoku es un juego tradicional japonés que se caracteriza por ser un problema NP-Duro, es un juego de alta dificultad que puede ser resuelto de forma rápida por una meta-heurística; La optimización por colonia de hormigas (**OCH**) es una meta-heurística basada en una colonia de hormigas artificial que busca una solución a un problema de optimización combinatorio. El objetivo del presente documento es describir detalladamente el funcionamiento de **OCH**; además se presentará la implementación de la meta-heurística en el juego del sudoku, el modelo propuesto y los resultados obtenidos de estos.

2. Sudoku

El sudoku es un juego matemático de origen japonés, es un caso especial del *cuadro latino* [1]; este rompecabezas fue publicado por primera vez en los años 70.

2.1. Terminología

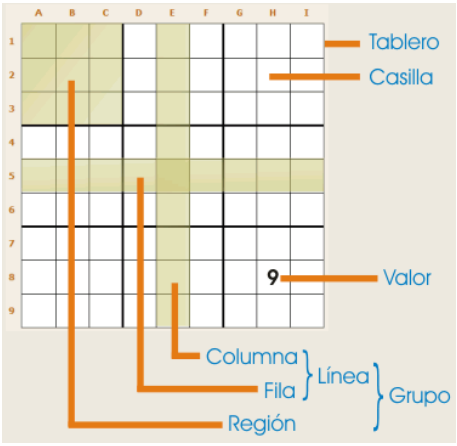


Figura 1: Terminología sudoku. Recuperado de `https://goo.gl/k9rgGq`

Tablero Cuadricula de $9 * 9$ casillas, 81 casillas en total.

Casilla Elemento del tablero que almacena un número entre el 1 y el 9, cada casilla pertence a una fila, columna y región.

Valor Número de una casilla.

Fila Línea horizontal del tablero compuesta por 9 casillas, cada tablero tiene 9 filas en total.

Columna Línea vertical del tablero compuesta por 9 casillas, cada tablero tiene 9 columnas en total.

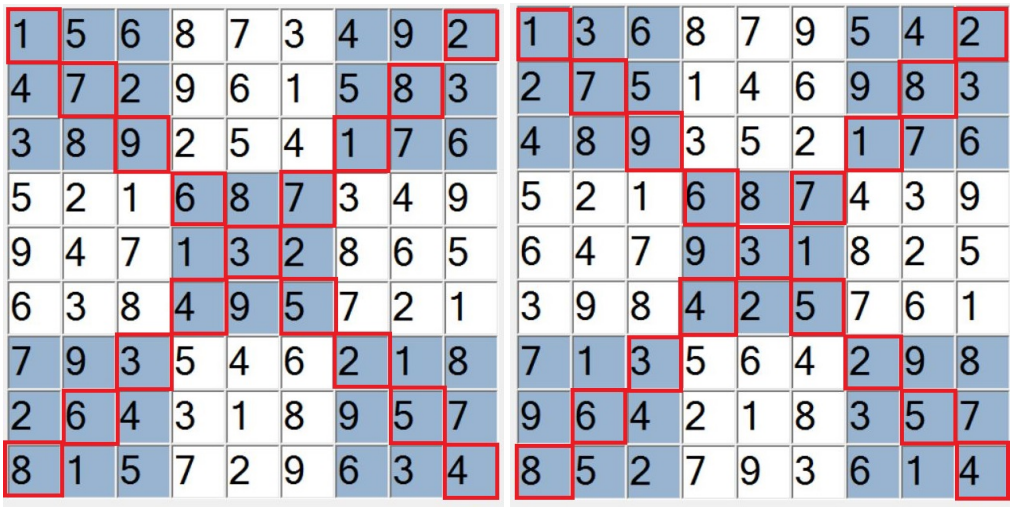
Región Cuadrícula de $3 * 3*$ casillas ubicada en el tablero, cada tablero tiene 9 regiones.

Tamaño tablero Cantidad de regiones tiene el tablero, denotado por n donde cada región es una cuadricula de $n * n$

[1] Un cuadro latino es una matriz de $n * n$ elementos en la que cada casilla está ocupada por uno de los n símbolos de tal modo que cada uno de ellos aparece exactamente una vez en cada columna y en cada fila

2.2. Descripción del problema

El sudoku consiste de un tablero de $9 * 9$ casillas, subdivido en regiones de $3 * 3$, el propósito del juego consiste en que cada región, fila y columna del tablero contenga los números del $1 - 9$, es decir que cada dígito sea único en la fila, columna y region que se encuentra.
El tablero tiene inicialmente un conjunto de números dispuestos en él, estos se conocen como *pistas* que cumplen con las reglas mencionadas anteriormente; se ha demostrado que para que un sudoku tenga una única solución debe iniciar con almenos 17 pistas [3] es decir , si un sudoku tiene menos de 16 pistas existe más de una solución asociada a él, únicamente a partir de 17 pistas iniciales puede existir una única solución pero no todas las configuraciones de tableros con 17 pistas tiene solución única.



(a) Solución 1 sudoku (b) Solución 2 sudoku

Figura 2: Soluciones diferentes a un sudoku que inicia con 17 pistas (Casillas marcadasd en rojo)

Los sudokus tienen diferentes niveles de dificultad, la cual está asociada a la cantidad de pistas que contenga el tablero inicial, un ejemplo de un tablero inicial de sudoku se presenta a continuación en 10

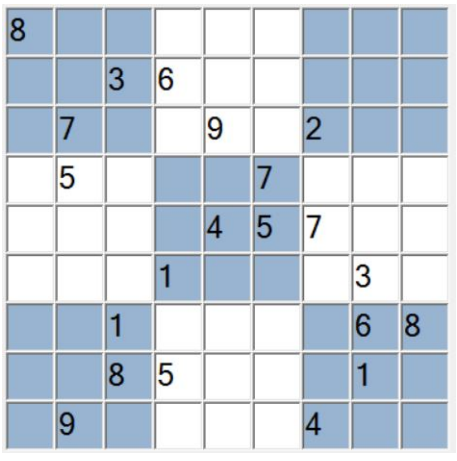


Figura 3: Ejemplo sudoku con alta dificultad según `https://goo.gl/WIf6ja`

3. Colonia de hormigas

3.1. Colonia hormigas naturales

Las hormigas son insectos donde muchas de sus especies son ciegas, conviven en colonias y presentan un comportamiento complejo gracias a la colaboración mutua, son capaces de encontrar los caminos más cortos entre su hormiguero y una fuente de alimentos, este es un aspecto interesante dado que muchas de ellas no tienen registros visuales de los caminos. El mécanismo utilizado por las hormigas consiste en depositar *feromonas* - una sustancia química que ellas pueden olfatear - entre el hormiguero y la fuente de alimentos, inicialmente si estas no encuentran ningún rasto de feromonas se mueven de forma aleatoria, sin embargo cuando se encuentran con feromonas, estas tienen una mayor tendencia a seguir el rastro; las hormigas eligen el camino a seguir con base en la concentración de fermonas que hay en un cruce de caminos, la probabilidad de un camino aumenta entre más feromonas contenga, sin embargo la decisión es probabilistica y no significa que las hormigas siempre tomen el camino con más feromonas en él, evitando así caer en un mínimo local.
El rastro de feromonas incrementa entre más hormigas pasen por un camino, eventualmente los caminos obtienen una alta concentración de feromonas y resalta un camino entre el hormiguero y la fuente de comida caracterizado por tener la mayor cantidad de feromonas, esto es gracias a que los caminos más cortos permite a las hormigas iniciar su viaje de retorno más rápidamente; Las feromonas son sustancias que se evaporan con el tiempo y así los caminos más largos empiezan a perder su deseabilidad de forma progresiva. La persistencia de las feromonas en los caminos depende de muchos factores como el tipo de suelo, la especies de las hormigas entre otros [1].
A continuación en 4 se presenta un ejemplo de su comportamiento; se puede observar que inicialmente el camino entre el hormiguero y la fuente de alimentos no tiene feromonas depositadas en él, de modo que las hormigas escogen de forma aleatoria, posteriormente la concentración de feromonas incrementa en el camino de la izquierda (el cual es en este caso el más corto) y finalmente aunque la concentración de feromonas es más alta en el camino de la izquierda, algunas hormigas aún deciden tomar el camino de la derecha pero la densidad de hormigas que transita por dicho camino ha disminuido considerablemente y por lo mismo la cantidad de feromonas en él también.

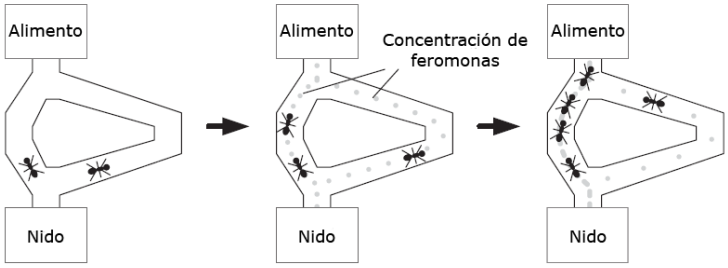


Figura 4: Comportamiento hormigas. Recuperado de `https://goo.gl/8kAlrw`

3.2. Colonia de hormigas ártificiales

El *Sistema de hormigas* fue el primer algoritmo basado en el comportamiento de las colonias de hormigas, propuesto por Dorigo, Maniezzo y Colorni en 1991 en [2]; El mecanismo de las hormigas para encontrar los caminos más cortos fue la inspiración para la meta-heurística de optimización conocido como *Sistema de hormigas*, el cual es una analogía del funcionamiento de las colonias de hormigas que permite resolver problemas de optimización combinatoria, de acuerdo a [2] el *Sistema de hormigas* tiene como principales carácteristcicas una retroalimentación positiva la cual permite converger a buenas soluciones rápidamente, cálculo distribuido y el uso de una heurística constructiva a través de iteraciones.
Existen diferentes variaciones del *Sistema de hormigas* que surgieron con la intención de mejorar la cálidad de la soluciones que se encuentran, estas metaheurísticas también son conocidos como *Optimización*

basada en Colonias de Hormigas (OCH) ,entre ellos se encuentran:

- Sistema de hormigas MAX - MIN
- Sistema de colonia de hormigas adaptativo
- Sistema de hormigas con ordenación
- Sistema de hormigas mejor - peor
- Sistema de hormigas - ciclo

El presente documento únicamente se enfocará en presentar el sistema de hormigas de [2].

3.2.1. Modelo matemático para OCH

Los tipos de problemas que pueden ser resueltos a través de OCH son aquellos que pueden ser representados a través de una gráfica ponderada conexa $G(C, T)$, que siga los siguientes aspectos [1]:

- Conjunto finito de componentes $C = \{c_1, c_2, c_3, ..., c_m\}$; representan los vértices de G.
- La transicion a_{xy} representa una conexión entre el componente x y el componente y; representan el conjunto de aristas de G.
- Una solucion S es un camino en G
- Las transiciones a_{xy} tienen un costo c_{xy} asociado
- Las transiciones o vértices tiene asociados un rastro de feromonas, denotadas como τ_{xy} si las feromonas se encuentran en las transiciones o τ_x si las feromonas están en los componentes.
- Un componente x tiene un conjunto de vecinos $N(x) = \{y | a_{xy} \in T\}$
- Las hormigas se denotan como $k = 1, 2, 3, ..., N$
- Una hormiga k se mueve de un componente x a un componente y
- $\rho \in (0, 1]$ es el coeficiente de evaporación de las feromonas
- $C(S)$ es la función de costo de una solución S, determina la cálidad de la solución

3.2.2. Problemas estáticos

Las características del problema que se dan inicialmente se mantienen durante la ejecución del programa, es decir que la grágica G no cambia durante la ejecución. Un ejemplo de estos es el problema del agente viajero, la información respecto a sus conexiones o distancias no presenta cambios durante el tiempo de ejecución.

3.2.3. Problemas dinámicos

La instancia del problema cambia en tiempo de ejecución, es decir que G debe ser configurada en tiempo de ejecución.

3.2.4. Hormiga artificial

Las hormigas artificiales son agentes computacionales que cooperan de forma conjunta para construir a través de iteraciones una solución S a un problema. La probabilidad de un moverse de un componente x a un componente y se denota como P_{xy}^k y esta depende de [1]:

Información heurística Nivel de deseabilidad de moverse de un componente x a un componente y; es denotado por η_{xy}

Rastros de feromonas artificiales Deseabilidad aprendida de movimientos anteriores por dichas transiciones a_{xy} o componentes x

Nota La diferencia que existe entre ubicar las feromonas en un componente x o una transición a_{xy} consiste en que al ubicar las feromonas en x se indica la importancia de escoger el elemento; mientas que al ubicar las feromonas en a_{xy} se da prioridad a escoger el componente y después de escoger a x. La ubicación de las feromonas depende del problema que se desea resolver.

Las hormigas artificiales tienen las siguientes carateristicas [1]:

- Busca soluciones válidas
- Tiene una memoria en la cual almacena el camino que ha seguido para encontrar una solución
- Tiene un estado inicial desde el cual inician la construcción de su solución
- La construcción de una solución termina cuando se satisface una condición de parada o se alcanza un estado objetivo
- Las hormigas pueden reconstruir el camino recorrido y actualizar los rastros de feromona depositados en componentes/transiciones visitados. La actualización de las feromonas se realiza una vez todas las hormigas terminan de construir su solución

La principal diferencia entre las hormigas artificiales y naturales radica en que las hormigas artificiales además de tomar decisiones con base en las concentraciones de feromonas de un camino, también utilizan información heurística (η) que les permite saber de antemano la deseabilidad de un camino.

3.2.5. Pseudocódigo Sistema de Hormigas

La operación de una colonia de hormigas artificiales consiste en tener N hormigas que se van a mover en G para encontrar cada una de ellas una solucion s_k; todas las hormigas son ubicadas en un componente inicial x_0 y a partir de este cada hormiga k evalua entre $N(x_0)$ sus respectivas posibilidades y toma una decisión con base en ellos, así cada hormiga k construye una solución s_k, una vez todas las hormigas han construido una solución, se procede a depositar los rastros de feromonas en los componentes/transiciones y a realizar el proceso que simula la evaporación de las feromonas; este proceso se repite hasta que una condición de parada se cumpla.
Para la construcción de una solución algorithm 2 una hormiga k se encuentra en un componente x y necesita moverse a un componente y, para decidir cuál componente y seleccionar, la hormiga k aplica P_{xy}^k [2] sobre cada uno de $N(x)$, donde $\alpha \in (0, 1]$ y $\beta \in (0, 1]$; con base en la probabilidad de cada $y \in N(x)$ escoge su próximo componente y lo adiciona a su solución; este proceso se lleva a cabo hasta que s_k esté completa o se cumpla alguna otra condición de paro, en el pseudocodigo esta última se omite, sin embargo se debe tener en cuenta que la hormiga puede no encontrar una solución.

La selección del siguiente componente y se hace, como ya se mencionó anteriormente, de forma probabilistica, notese que $\sum(P_{xy}^k) = 1$, lo cual significa que cada P_{xy}^k representa una porción de esta sumatoria; la operación que se realiza es elegir un número de forma aleatoria $[0, 1]$ y tomar el componente y que se encuentre en dicha porción de la sumatoria; supongamos que $N(x) = \{y_1, y_2, y_3, y_4\}$ y

Algorithm 1 Sistema de hormigas

function OCH(Componentes,Transiciones)
    IniciarFeromonas()
    CrearHormigas(N)
    repeat
        UbicarPosicionInicial(Hormigas) ▷ Hormigas posición inicial
        for $k \leftarrow 0$ to N **do**
            ConstruirSolucion(Hormiga$_k$,Componentes,Transiciones) ▷ Solución hormiga k
        end for
        MejorSolucion $\leftarrow$ SeleccionarMejorHormiga(Hormigas)
        ActualizarFeromonas(Componentes,Transiciones,Hormigas)
    until CondicionParada()
    return Best
end function

Algorithm 2 Construcción de una solución

procedure CONSTRUIRSOLUCION(Hormiga$_k$,Componentes,Transiciones)
    while noCompleta(Hormiga$_k$.s) **do** ▷ x estado actual de la hormiga
        for all y $\in$ N(Hormiga$_k$.x) **do**
            $P_{xy}^k \leftarrow \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{u \in N_x}(\tau_{xy}^{\alpha})(\eta^{\beta})}$
        end for
        Siguiente $\leftarrow$ SeleccionarComponente(P,N(Hormiga$_k$.x))
        Hormiga$_k$.x $\leftarrow$ Siguiente ▷ s solución de la hormiga
        Hormiga$_k$.s $\leftarrow$ Hormigass$_k$.s $\cup$ Siguiente
    end while
end procedure

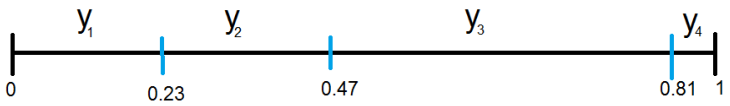


Figura 5: Representación visual de P_{xy}^k

Algorithm 3 Seleccionar componente

function SELECCIONARCOMPONENTE(P,N(Hormiga$_k$.x)
    Num $\leftarrow$ Random(0,1) ▷ Elige un número al azar entre 0 y 1
    Sum $\leftarrow 0$
    for all $P_{xy}^k \in$ P **do**
        Sum $\leftarrow$ Sum + P_{xy}^k
        if Num $<$ Sum **then**
            return y
        end if
    end for
end function

$P_{xy}^k = \{0{,}23, 0{,}24, 0{,}34, 0{,}19\}$ como se observa en fig. 5, si por ejemplo el número seleccionado al azar es 0.45, el componente seleccionado sería y_2.
Finalmente la actualización de feromonas en componentes/transiciones es llevada a cabo, donde $\Delta\tau_{xy}^k \leftarrow f(C(Hormiga_k.s))$ es la cantidad de feromonas que deposita una hormiga en la transición a_xy, $\Delta\tau_{xy}^k = 0$ si la hormiga no utilizó la transición a_xy; f es una función que depende de la cálidad de la solución encontrada por la hormiga, f debe ser definida según el problema; la evaporación de feromonas es representada por $(1-\rho)\tau_{xy}$. El procedimiento es análogo si las feromonas están en los componentes, en algorithm 4 se presenta el caso donde las feromonas están en las transiciones.

Algorithm 4 Actualización y evaporación de feromonas

```
1: procedure ACTUALIZARFEROMONAS(Componentes, Transiciones, Hormigas)
2:     for all a_xy ∈ Transiciones do          ▷ Δτ_xy^k ← f(C(Hormiga_k.s))
3:         τ_xy ← (1 − ρ)τ_xy + Σ_k Δτ_xy^k
4:     end for
5: end procedure
```

4. Descripción implementación

La implementación desarrollada se encuentra subdividida en dos partes, la primera modela los elementos que pertenecen a OCH y la segunda es la encargada de representar el juego del Sudoku bajo los parámetros propuesto por la implementación de OCH.

4.1. Optimización basada en colonia de hormigas: Sistema de hormigas

El modelo propuesto para OCH es genérico, es capaz de resolver cualquier problema siempre y cuando este se configure bajo sus parámetros; se identificaron diferentes objetos para modelar a OCH, dado que este necesita de una gráfica G ponderada se plantea la clase **Grafica** la cual tiene un conjunto de componentes C y estos se representan por medio de la clase **Componente** los cuales son los elementos del problema, dicha clase se propone como *abstracta* de modo que un componente se pueda modelar según el problema que se desea resolver; las transiciones entre los componentes se configuraron en la clase **Transición**, esta contiene los dos componentes que comunica en G, como ya se mencionó anteriormente las feromonas pueden estar ubicadas en los componentes o en las transiciones, para esto tanto la clase **Componente** como **Transicion** se modelan con el atributo *feromonas* que hace referencia a la cantidad de feromonas que almacena, todavía cabe señalar que de la ubicación de las feromonas también depende donde se asigna el atractivo, el atractivo y las feromonas deben asignarse en el mismo elemento Componente o Transición; a continuación se puede observas que un **Componente** conoce las transiciones que tiene con los demás elementos, lo cual permite recorrer la gráfica a partir de un componente, esto se puede afirmar dado que G es conexa.

En segunda instancia obsérvese la clase abstracta **Solucion** se compone de una gráfica que en este caso es el camino sobre G que se tomó para llegar a dicha solución; el método *funcion_Costo()* permite cuantificar la calidad de una solución (representa a $C(s)$); la razón por la cual **Solucion** es abstracta radica en que esta cuantificación depende del problema que se desea resolver y su cálculo se hace con base en el camino que se tomó sobre la gráfica; Con respecto al método *vertice_Actualizado()* (abstracto) es invocado para informar cuando un nuevo componente hace parte de la solución,con el objetivo de que si el problema lo requiere realice operaciones adicionales.

La clase **Hormiga** representa al agente computacional que construye una solución, cada hormiga se identifica con un número único y se encarga de construir una solución al problema; el método *funcion_feromonas(...)* es la representación de f, indica la cantidad de feromonas que una hormiga deposita en un Componente/Transición, este se modeló como sigue:

$$f(C(s)) = \begin{cases} (1-\rho) * C(s) & si \quad maximizar \\ \frac{(1-\rho)}{C(s)} & si \quad minimizar \end{cases}$$

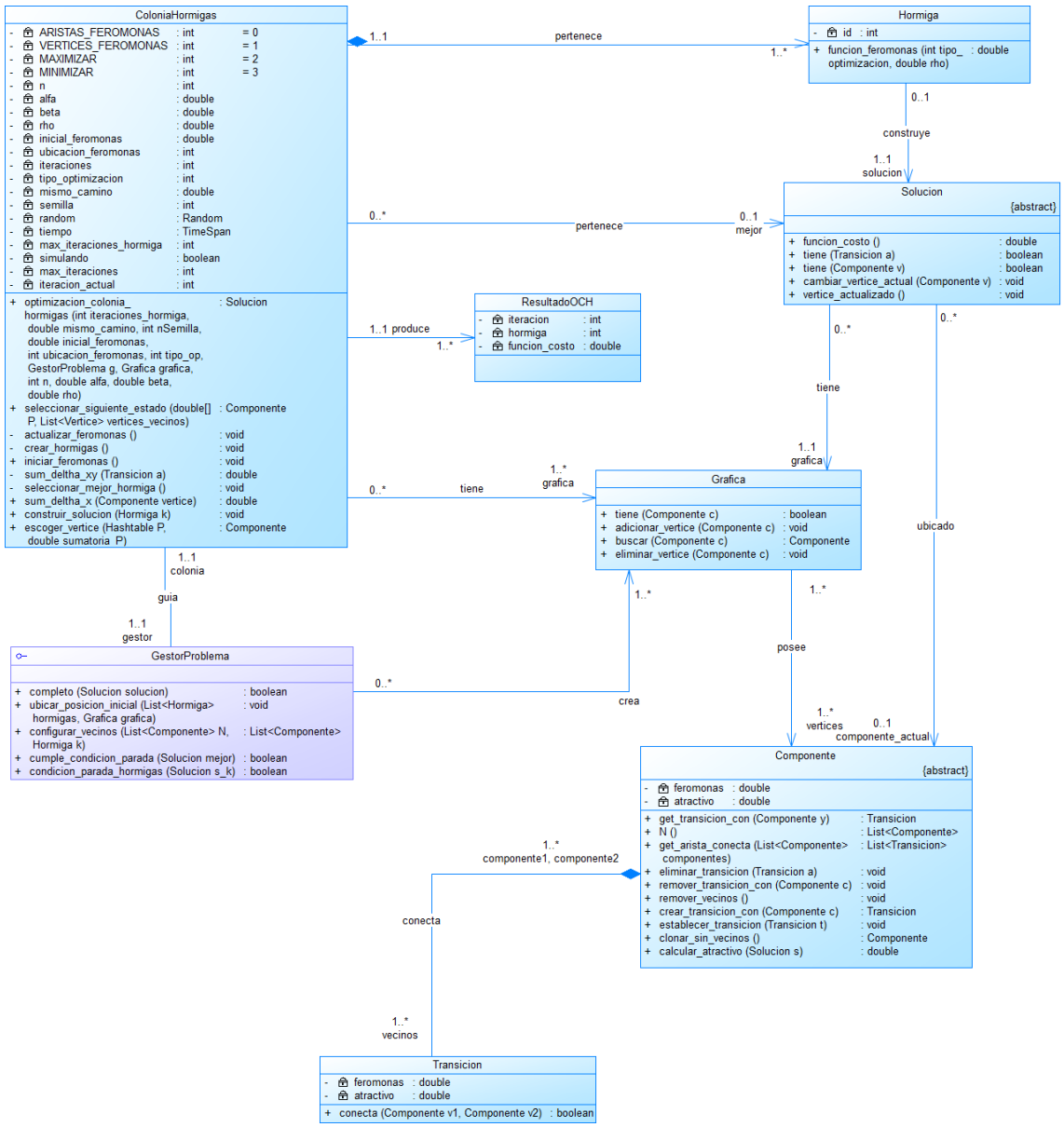


Figura 6: Diagrama de clases para el sistema de hormigas

Finalmente se tiene a la clase **ColoniaHormigas** la cual implementa el pseudocodigo seccion 3.2.5; tiene un conjunto de hormigas,en esta implementación se definió como condición de parada que si un porcentaje de hormigas ha construido la misma solución la simulación puede terminar; la clase **ColoniaHormigas** se guía con base en **GestorProblema** el cual es una interface que se debe implementar para cada problema, este será el conocedor de las reglas del problema y podrá determinar condiciones de parada adicionales a la simulación que dependan del problema. La clase **ColoniaHormigas** se implementó con dos condiciones de paradas adicionales a las descritas en el pseudocódigo, los cuales delimitan la cantidad de iteraciones que una hormiga puede tomar construyendo una solución y la cantidad máxima de iteraciones del algoritmo 1; adicionalmente teniendo en cuenta que los problemas pueden ser dinámicos o estáticos se define el

método *configurar_vecinos(...)* el cual se invoca cuando se obtienen los vecinos del componente actual de una hormiga, con esto **GestorProblema** puede configurar los vecinos de un componente con base en la solución actual de una hormiga.

4.2. Sudoku

El sudoku se modeló en términos de una gráfica G, donde los vértices de la gráfica son todos los posibles valores válidos que puede tomar cada casilla del tablero, estos se modelan por medio de la clase **Casilla** la cual representa un valor válido en el tablero del sudoku, esta indica la fila y columna a la cual pertenece y un posible valor de dicha posición del tablero; por ejemplo, supongamos el sudoku fig. 7.

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

Figura 7: Ejemplo de sudoku

Se crean entre otros los siguientes objetos casilla (Las filas y columnas del tablero se enumeran del 0 al 8):

- Casilla (fila: 0, columna:0, valor: 5)
- Casilla (fila: 0, columna:1, valor: 3)
- Casilla (fila: 0, columna:2, valor: 1)
- Casilla (fila: 0, columna:3, valor: 9)
- Casilla (fila: 0, columna:4, valor: 8)
- Casilla (fila: 0, columna:5, valor: 5)
- Casilla (fila: 0, columna:5, valor: 7)
- Casilla (fila: 0, columna:5, valor: 9)
- ...
- ...
- ...

- Casilla (fila: 8, columna:8, valor: 3)

La gráfica G que se crea es completa, es decir cada casilla es adyacente a todos los demás; el problema se considera dinámico dado que una vez se selecciona una casilla por las restricciones del problema se deben evitar todas aquellas casillas que al ubicarlas en el tablero incumplan las reglas.

El problema del sudoku se configuró para que las feromonas se almacenen en los componentes ya que lo importante aquí es saber qué casillas se adaptan mejor al problema, no interesa el orden en que estas se escojan, si no cuales se escogen. El atractivo de una casilla se determina en tiempo real y se le adiciona valor si:

- El tablero actual de la hormiga k tiene vacia la posición de la casilla
- Si el valor de la casilla es uno de los valores que falta en la fila de la casilla
- Si el valor de la casilla es uno de los valores que falta en la columna de la casilla
- Si el valor de la casilla es uno de los valores que falta en la region de la casilla
- Si el valor de la casilla es el único valor que falta en la fila de la casilla
- Si el valor de la casilla es el único valor que falta en la columna de la casilla
- Si el valor de la casilla es el único valor que falta en la región de la casilla
- Si el valor de la casilla es el único valor que falta en la fila, columna y región de la casilla
- Si el valor de la casilla es un valor que falta tanto en la fila, columna y región de la casilla

Dichas decisiones se toman con base en la solución que está construyendo la hormiga, es decir se observa el tablero actual solución de la hormiga, se observa una casilla, su posición y cómo ubicarla sobre dicho tablero beneficia a la solución.
La clase **GestorSudoku** es el encargado de hacer cumplir las restricciones del problema, ubica las hormigas en un componente seleccionado de forma aleatoria sobre los componentes de la gráfica y define como condiciones de parada adicionales que si el sudoku ya está completo se detenga la construcción de la solución por parte de una hormiga y/o se detenga la simulación y retorne dicha solución; por otro lado la clase **Sudoku** es considerada una solución, la cual además de estar contenida en la gráfica que construye una hormiga por cuestiones de facilidad se modela también una matriz que represente el tablero y el cual se va actualizando conforme la solución se construye, esto con la intención de evitar la búsqueda sobre la gráfica que tiene la superclase **Solucion**; La función de costo de un sudoku se mide en términos de la cantidad de casillas que quedan vacías en el tablero(la función de costo es la cantidad de casillas vacias + 1, de modo que un sudoku resuelto tiene función de costo 1), esto porque se sabe de antemano que los valores que se ubican en la solución nunca incumplen las reglas, sin embargo puede llegar un punto en que la hormiga no pueda adicionar ningún valor sin incumplir las restricciones del sudoku, lo cual implica que algunas casillas en el tablero quedarían vacías. El modelo finalmente concluye como se puede ver en

5. Descripción técnica de la implementación

La implementación de OCH para el juego del Sudoku, se desarrolló en en el lenguaje C# sobre la plataforma Visual studio express 2013, no se utilizó ninguna biblioteca adicional para su implementación.
El sistema desarrrollado cuenta con una interfaz gráfica que permite a los usuarios ingresar el problema de sudoku que desea resolver y el valor de la semilla para el generador de números aleatorios (**RNG** por sus siglas en inglés), a partir de esto el sistema puede iniciar la simulación y el usuario puede observar en pantalla cómo se está intentando dar solución al problema, además se puede observar que tan cerca se encuentra la simulación de acabar con respecto a las máximas iteraciones que se asignaron; una vez la simulación conlcuye se presenta el tiempo que le tomó al sistema intentar resolver el problema y se muestra el estado final de los componentes del problema. Adicionalmente se puede ver cómo fueron las soluciones

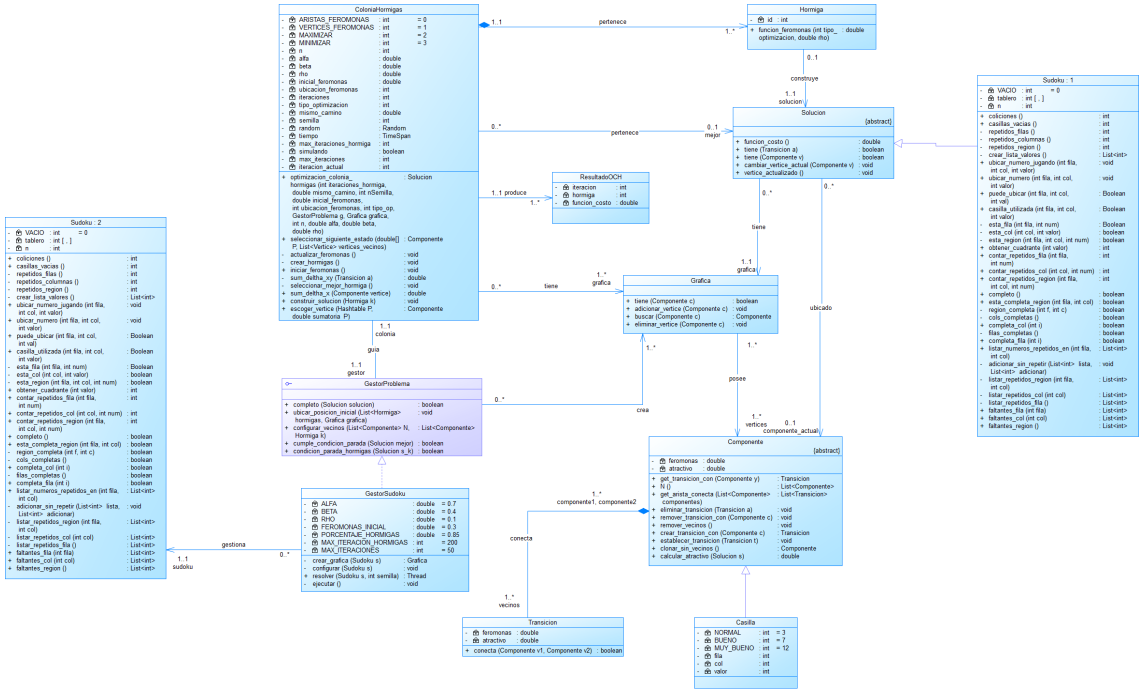


Figura 8: Ejemplo de sudoku

encontradas por cada hormiga por cada iteración a través de una gráfica que lo presenta; en esta se puede observar como las hormigas empiezan a converger hacia una misma solución, se debe tener en cuenta que una vez la solución es encontrada,en este caso, la simulación se detiene por lo tanto no siempre todas las hormigas construyen una solución en una iteración.

6. Parámetros Libres

Los parámetros libres son valores que determinan la calidad del sistema, estos son ρ, β, α, los cuales deben escogerse de forma apropiada, β determina la influencia de η y α la influencia de τ_{xy} en P_{xy}^{k}, dichas influencias depende del problema, se debe tener en cuenta qué es más importante para el , si su atractivo o la cantidad de feromonas que almacena el Componente/Transición, para el caso del sudoku, es más representativo la cantidad de feromonas que almacene una Casilla, por que significa que es más probable que sea el valor correcto en el tablero.

Por medio de experimentación y realizando varias simulaciones del sistema con diferentes valores de β y α se optó por seleccionar a $\beta = 0{,}4$ y $\alpha = 0{,}7$.

7. Funcionamiento del sistema

El sistema desarrollado inicia una interfaz gráfica que permite ingresar un tablero de sudoku, las pistas se pueden escribir sobre el tablero dispuesto en pantalla o cargar de una archivo de texto que debe tener los número del tablero separados por coma, en una sola linea, según el orden de las filas del tablero, los espacios en blanco se representan como un cero; en ambos casos el sistema se asegura que las pistas ingresadas cumplan con las reglas de un sudoku; el sistema también permite guardar un tablero de sudoku en un archivo de texto que sigue el formato anterior para que este se pueda ver posteriormente.

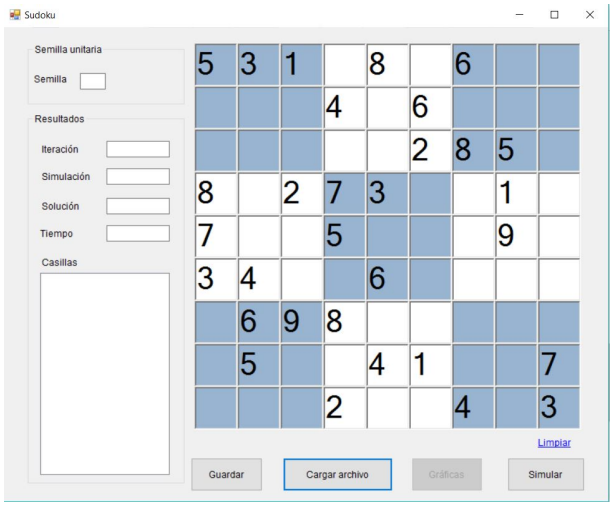


Figura 9: Inicio aplicación

La aplicación permite ingresar el valor de la semilla para el generador de números aleatorios, una vez se selecciona el botón *Simular* el sistema presenta al usuario las soluciones en intervalos de dos segundos cómo va la solución que está construyendo una hormiga, una vez la simulación concluye, se presenta el estado final de cada casilla del problema, el tiempo total de simulación y si la solución que se encontró es correcta o no.

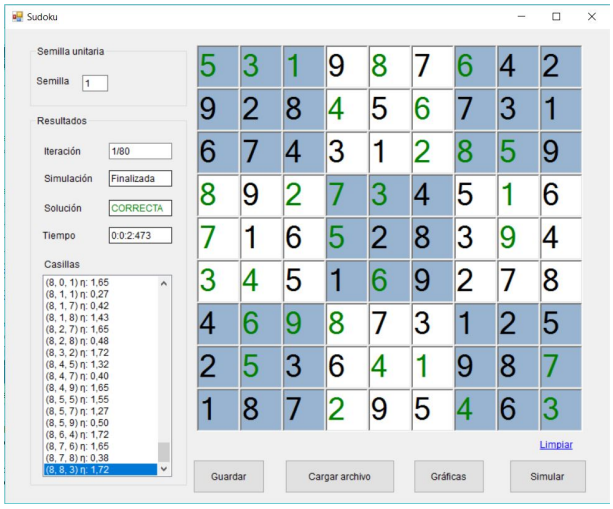


Figura 10: Resultado de una simulación

Adicionalmente el usuario puede ver un reporte gráfico que presenta la función de costo de la solución que encontró una hormiga en una iteración, el eje x representa la iteración, el eje y es la función de costo de la solución de una hormiga, cada hormiga tiene su propia serie en la gráfica. Dado que la clase **GestorSudoku** determina otras condiciones de parada, no siempre todas las hormigas construyen una solución en una iteración.

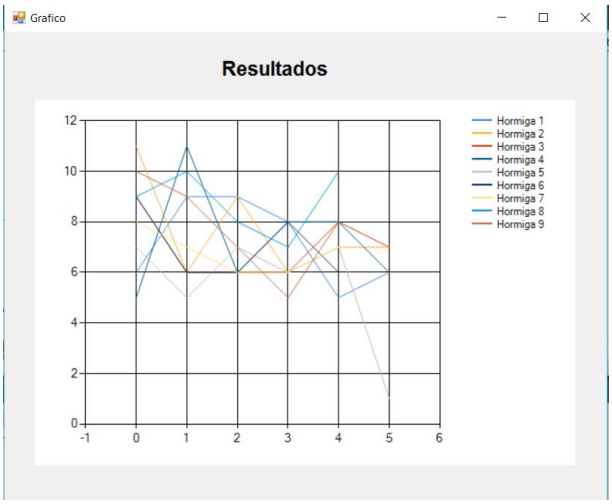


Figura 11: Reporte de una simulación

8. Resultados

El sistema implementado logra resolver un sudoku de complejidad baja en segundos,la mayoría de las personas tardan alrededor de 30 minutos [4], de acuerdo a la dificultad del sudoku en algunas ocasiones el sistema no es capaz de dar solución al problema y cuando logra encontrar su solución, normalmente tarda un aproximado de 30 a 40 minutos; comparando con algoritmo implementado en [4] que resuelve un sudoku en milisegundos, se considera que el sistema es exitoso.
Por medio de experimentación se observó que de acuerdo a la dificultad del problema el sistema obtiene los siguientes resultados:

Nivel dificultad	Porcentaje de sudokus resueltos
Fácil	91.30 %
Medio	84.20 %
Difícil	61 %
Muy difícil	15.39 %

Cuadro 1: Resultados obtenidos

Los sudokus utilizados para obtener los resultados fueron tomados de `http://www.websudoku.com/`, con base en los niveles de dificultad de los problemas allí propuestos.

Referencias

[1] Sergio Alonso, Oscar Cordón, Iñaki Fernández de Viana, and Francisco Herrera. La metaheurística de optimización basada en colonias de hormigas: Modelos y nuevos enfoques. 2003.

[2] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. 1991.

[3] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue sudoku solving the sudoku minimum number of clues problem. 2013.

[4] Krzysztof Schiffi. An ant algorith for the sudoku problem. 2015.