DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Sudoku

17 de diciembre de 2018

Metaheurísticas
Trabajo Práctico Final

Integrante	LU	Correo electrónico
Emanuel Lamela	021/13	lamelaemanuel@gmail.com
Lucía Parral	162/13	luciaparral@gmail.com

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) 4576-3300
http://www.exactas.uba.ar

Índice

1. Introducción **2**

2. Desarrollo **2**
2.1. Simulating Annealing . . . 2
2.2. Sudoku . . . 3
2.3. Simulating Annealing para resolución de Sudokus . . . 3

3. Experimentación y Análisis **5**
3.1. Fijación de parámetros . . . 5
3.2. Efectividad en Sudokus de 3x3 de dificultad variada . . . 6
3.3. Efectividad en Sudokus de 4x4 y 5x5, de dificultad alta . . . 7

4. Comparación **7**
4.1. Colonia de hormigas . . . 7
4.1.1. Introducción . . . 7
4.1.2. Aplicación en Sudoku . . . 8
4.2. Algoritmos Genéticos . . . 9
4.2.1. Introducción . . . 9
4.2.2. Aplicación en Sudoku . . . 9

5. Conclusión **10**

6. Referencias **10**

1. Introducción

Este Trabajo Práctico busca explorar soluciones al problema de Sudoku, que es NP-Hard utilizando Metaheurísticas.

Por una parte, se explicará y desarrollará una solución basada en la conocida Heurística Simulated Annealing. La efectividad de la misma será luego puesta a prueba utilizando distintos datasets.

Por otra parte, describiremos posibles soluciones a este problema utilizando Metaheurísticas basadas en población, en particular Colonia de Hormigas y Algoritmos Genéticos.

Finalmente, detallaremos en unos párrafos algunas conclusiones que se pueden desprender de todo lo anterior y posibles rutas prometedoras para mejora.

2. Desarrollo

2.1. Simulating Annealing

Simulating Annealing es un algoritmo probabilístico, aleatorio y sin memoria. Esta metaheurística está basada en el proceso de *annealing* del metal. Fuera del mundo de la computación, este proceso involucra el calentamiento y enfriamiento del material para poder alterar sus propiedades físicas de manera que cambie su estructura interna. A medida que el metal se enfría, su nueva estructura se va fijando y consecuentemente, el metal retiene sus nuevas propiedades. Si la temperatura inicial no es lo suficientemente alta, o bien el enfriamiento se realiza muy rápidamente, el resultado final puede contener imperfecciones considerables.

Los algoritmos basados en el proceso de Simulating Annealing simulan los cambios de energía en el sistema sujeto a un proceso de enfriamiento hasta que converge en un estado de equilibrio. Una temperatura inicial alta permite cubrir una gran porción del espacio de búsqueda. Sin embargo, a su vez debe ser lo suficientemente baja como para no incrementar demasiado el costo computacional.

El trabajo del algoritmo es explorar el espacio de búsqueda observando en cada iteración un vecino aleatorio. Cualquier movimiento que mejore la función objetivo es aceptado siempre. Si no mejora, dicho vecino es considerado con una probabilidad dependiente de la temperatura actual y de la cantidad de degradación ΔE, que es la diferencia entre el valor de la función objetivo sobre la solución actual y la solución vecina.

A medida que la temperatura disminuye la probabilidad también, ya que está definida como:

$$P(\Delta E, T) = e^{-\Delta E/T} = \frac{1}{e^{\Delta E/T}} \tag{1}$$

Al decrementar la temperatura el cociente $\Delta E/T$ se incrementa, por ende el denominador del calculo también lo hace, disminuyendo el resultado final, o sea la probabilidad. Este fenómeno permite que el algoritmo comience con mayor tendencia a la exploración y finalice favoreciendo casi exclusivamente a configuraciones mejores.

Los cambios de temperatura se producen al alcanzarse el estado de equilibrio en una temperatura. Esto sucede a partir de una serie de movimientos, pudiendo ser **estáticos** (proporcionales al tamaño del vecindario) o **adaptativos** (el enfriamiento se produce cuando hay una mejor solución).

El criterio de parada de la técnica suele estar dado por el alcance de una temperatura mínima, pero por cuestiones de eficiencia puede que se incluya un criterio de parada temprano el cuál toma efecto cuando la solución no es mejorada en una cierta cantidad de iteraciones.

Un pseudocódigo del algoritmo es el siguiente:

Algorithm 1 Algoritmo de Simulated Annealing general

```
1: s := s0; /* Se comienza con una solución inicial */
2: T := T_max; /* Se comienza con una temperatura máxima */
3: while No se alcance el criterio de parada do
4:     while No se alcance la condición de equilibrio do
5:         Obtener s', un vecino aleatorio;
6:         ΔE := f(s') − f(s);
7:         if ΔE <= 0 then
8:             s := s' /*Se acepta la nueva solución ya que es mejor*/
9:         else
10:            Aceptar s' con probabilidad e^{−ΔE/T}
11:        end if
12:        Actualizar temperatura;
13:    end while
14: end while
```

2.2. Sudoku

El Sudoku es un *puzzle* lógico matemático inventado hacia finales de la decada de 1970.

Dado un tablero de n^2 x n^2 dividido en n^2 cuadrantes de n x n, con algunas celdas con valores prefijados y otras con valores variantes, se busca rellenar las celdas libres de manera que valga que:

- Cada fila contiene números enteros comprendidos entre el 1 y n^2 sin repetidos.
- Cada columna contiene números enteros comprendidos entre el 1 y n^2 sin repetidos.
- Cada cuadrante contiene números enteros comprendidos entre el 1 y n^2 sin repetidos.

Una instancia de Sudoku es válida si existe una forma de llenar las celdas de manera que se cumplan todas las condiciones previas.

2.3. Simulating Annealing para resolución de Sudokus

Para resolver el problema del Sudoku con Simulating Annealing, comenzamos por generar una solución inicial de Sudoku, que cuenta con ciertas posiciones fijas y para cada cuadrante no tiene repetidos. Consideramos que esta solución inicial es un poco mejor que llenar de forma aleatoria el tablero, ya que se asegura al menos una de las condiciones que necesita cumplir una solución de Sudoku válida:

- Las filas no tienen valores repetidos.
- Las columnas no tienen valores repetidos.
- Los cuadrantes no tienen valores repetidos.

La función objetivo que utilizamos para comparar dos soluciones es bastante intuitiva:

$$\sum_{i,j=1}^{n^2} repeticiones_{i,j} \tag{2}$$

Donde $repeticiones_{i,j}$ es la suma de las apariciones repetidas para el valor de la posición i, j en filas, columnas y cuadrantes. Ésta fórmula considera que cualquier repetición tiene el mismo peso aún cuando nuestra configuración inicial lo previene en cuadrantes. Este detalle tiene relación con el hecho de que no todas las alternativas de vecindad que planteamos utilizan esa particularidad a favor.

Consideramos distintas estrategias al momento de armar las vecindades, para poder luego experimentar con el funcionamiento del algoritmo las distintas variaciones:

1. Elegir una posición cualquiera y sumarle 1, tomando módulo n^2 de la solución.
2. Intercambiar dos valores en posiciones aleatorias del tablero.
3. Intercambiar dos valores en posiciones aleatorias dentro de un mismo cuadrante, también elegido aleatoriamente.

En todos los casos se tomó en cuenta no alterar posiciones fijas.

El *scheduling* de temperatura que utilizamos está dado por la fórmula:

$$T_{i+1} = (1 - \alpha) * T_i \tag{3}$$

Dónde α se suele elegir dentro del rango [0,85...0,99]. Nosotros seleccionamos 0,9, por ende la temperatura se reduce a un 10 % de su valor anterior en cada paso.

Otro parámetro que fue fijado de antemano es la temperatura inicial. Se seleccionó el valor 1000.

Las temperaturas mínimas que consideramos son 0,1 y 0,01. No parece una diferencia sustancial, pero en el fondo esto agrega un paso más de reducción de temperatura.

Finalmente, se eligió una condición de equilibrio estática la cuál es alcanzada cuando exploran n^6 vecinos distintos en un mismo peldaño de temperatura.

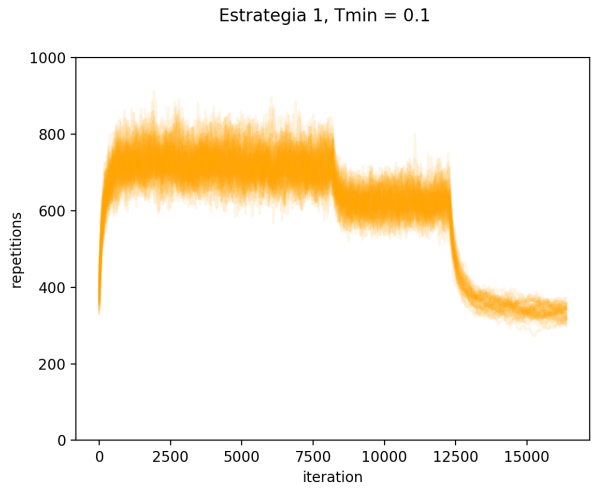
Con esto en mente, nos proponemos evaluar qué combinación de parámetros tienen mejor *performance* y luego utilizar dicha combinación para medir la efectividad de la heurística contra ciertos *datasets*.

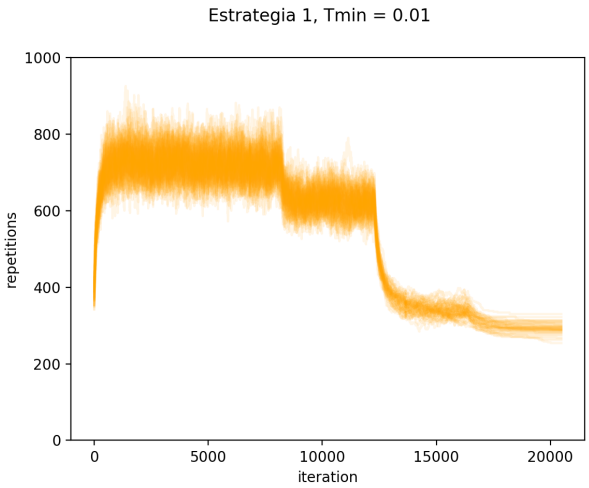
3. Experimentación y Análisis

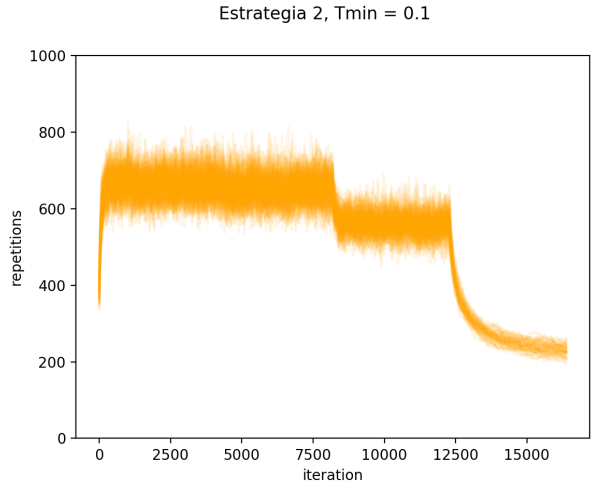
3.1. Fijación de parámetros

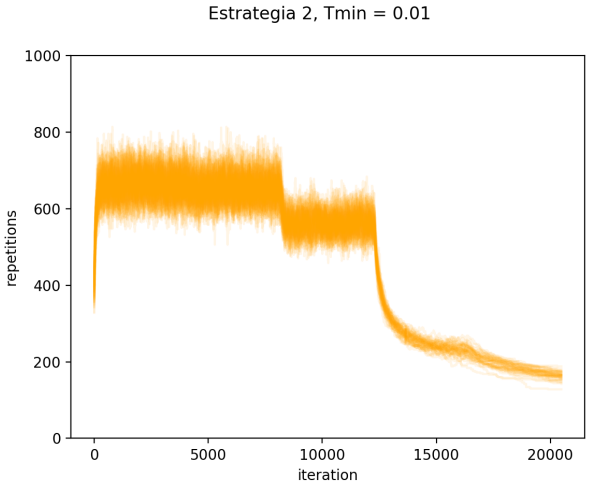
En esta etapa de la experimentación, nos proponemos probar las combinaciones de parámetros que son variables y encontrar la mejor. Para ello, corrió la implementación del algoritmo contra un *dataset* de 50 Sudokus de 4x4 con un cubrimiento fijo inicial del 20 %, un total de 52 posiciones, generados de manera aleatoria. Por lo mencionado anteriormente, las únicas aristas variantes son **la estrategia de vecindad** y la **temperatura mínima**.

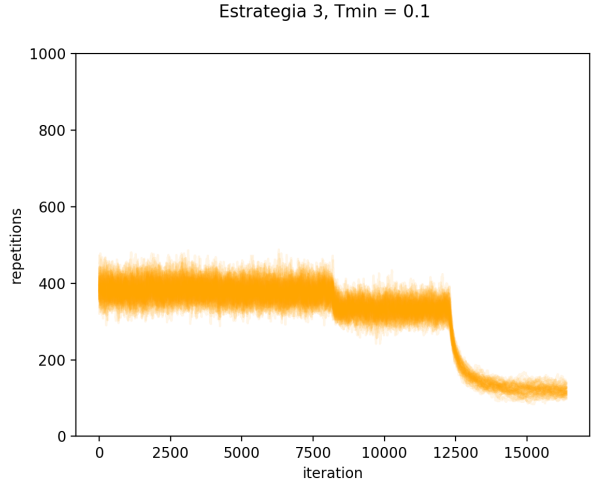
A continuación, los resultados en gráficos:

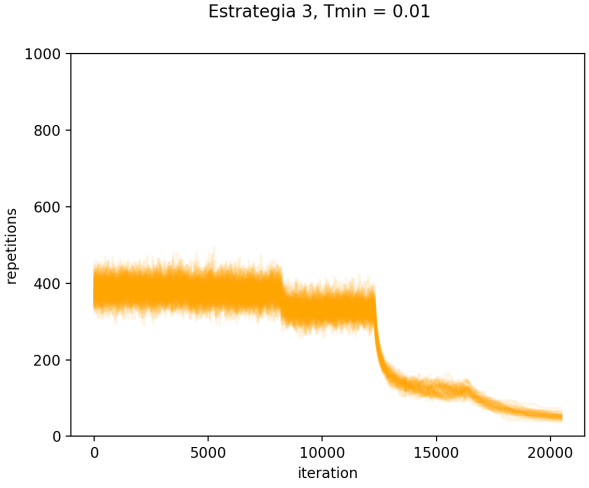












Rápidamente se puede observar que la combinación que ofrece mejores resultados es **Estrategia 3 y Temperatura mínima 0,01**. Analizaremos el por qué de cada uno por separado.

El hecho de que la Estrategia 3 sea la mejor está directamente relacionado con cómo populamos inicialmente las instancias. Recordando lo mencionado en la sección 2.3, la solución inicial garantiza que no hay repeticiones intracuadrantes. La Estrategia 3 es la única que honra ese hecho. Las restantes comienzan generando soluciones peores que la inicial, ya que rompen la garantía que poseían, y terminan generando soluciones con menor magnitud de función objetivo cerca de la etapa final de la corrida. Esta ventaja le permite a la Estrategia 3 explorar vecinos con menores costos por mayor tiempo y últimamente consigue las mejores soluciones finales.

Por otra parte, la temperatura mínima 0,01, en contraposición a 0,1, hace que haya un refinamiento apreciable en las ~ 5000 iteraciones finales. Es un costo no despreciable a cambio de una mejora notable.

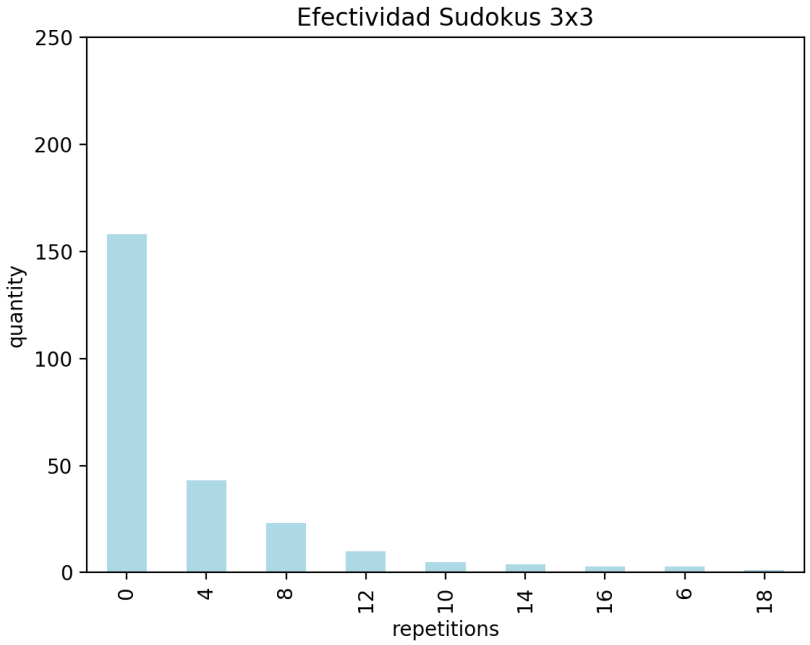
Un aspecto que pasamos por alto y evidenciamos al ver los gráficos es la cantidad de iteraciones intrascendentes que ocurren al principio. Creíamos que una elección alta de temperatura, y en consecuencia una capacidad de exploración más abarcativa, iba a tener un efecto notablemente positivo en la efectividad. No resultó así. Esto nos da la pauta que hubiese sido más sensato elegir una temperatura inicial sustancialmente más pequeña. Las temperaturas altas son una mejor elección para problemas dónde es más difícil salir de un máximo local, que no parecería ser el caso en este contexto.

A modo de resumen, a continuación detallamos la totalidad de la configuración de parámetros que se van a utilizar en el resto de los experimentos:

- Función objetivo: Total de repeticiones en filas, columnas y cuadrantes
- Solución inicial: Populado *random* sin repeticiones intracuadrantes
- Estrategia de Vecindad: *Estrategia 3*
- # Iteraciones estado de equilibrio: n^6
- Temperatura inicial: 1000
- Temperatura mínima: 0,01
- α de *scheduling* de enfriamiento: 0,9

3.2. Efectividad en Sudokus de 3x3 de dificultad variada

Utilizando la configuración previamente mencionada, vamos a probar contra un dataset de 250 Sudokus de 3x3 obtenido en `Kaggle`. Estos son los resultados:



Para nuestra alegría, la efectividad fue de $\sim 60\,\%$. No sólo eso, sino que un $\sim 20\,\%$ tuvo nada más 4 repeticiones. En definitiva, casi un $80\,\%$ tuvo 4 repeticiones o menos. Para ser una implementación relativamente simple, creemos que es un buen número.

3.3. Efectividad en Sudokus de 4x4 y 5x5, de dificultad alta

Para la última etapa de experimentación, extrajimos instancias de 4x4 y 5x5 de dificultad alta de `http://sudoku-puzzles.merschat.com/`. La complejidad de dichas instancias radica en la cantidad de casilleros fijos y las técnicas que hacen falta para resolverlos por un ser humano.

Detallamos los resultados para cada uno a continuación:

- Para el *dataset* de instancias de tamaño 4x4 y vimos que el costo final obtenido a lo largo del *dataset* era del orden de 30.
- Para el *dataset* de instancias de tamaño 5x5 y vimos que el costo final obtenido a lo largo del *dataset* era del orden de 100.

Veamos cuán es la efectividad relativa a una cota del valor máximo de repeticiones de un Sudoku:

$$(n^4 - m) * ((n^2 - 1) + (n^2 - 1) + (n^2 - 1)) = (n^4 - m) * 3(n^2 - 1) \tag{4}$$

Siendo m la cantidad de casilleros fijos del tablero, $n^4 - m$ son la cantidad de casilleros alterables. Para cada uno de ellos, puede haber $n^2 - 1$ repeticiones por fila, columna y cuadrante. Ahora, esto último es dónde la cota es no ajustada puesto que habría que restar las posiciones fijas de dicha fila, columna o cuadrante. No obstante, creemos que es lo suficientemente ajustada para ilustrar el punto.

Para el caso de 4x4, $m = 35$. Por lo tanto, el valor resultante es $(4^4 - 36) * 3(4^2 - 1) = 9900$. 30 en relación a 9900 es bajísimo.

Un análisis similar para los de 5x5 revela una conclusión similar. Con $m = 100$, primero obtenemos el valor de la cota de repeticiones que es $(5^4 - 100) * 3(5^2 - 1) = 37800$. Nuevamente, observamos que están en escalas totalmente distintas. 100 es casi despreciable al lado de 37800.

En definitiva, consideramos que nuestra implementación fue altamente efectiva para estos Sudokus también.

4. Comparación

4.1. Colonia de hormigas

4.1.1. Introducción

Colonia de hormigas es método basado en población que se inspira en el comportamiento de las hormigas, en particular, en su forma de comunicación basada en el depósito y detección de feromonas.

Las hormigas reales depositan feromonas que sirven para orientar a las demás hormigas a determinados recursos mientras exploran. Las *hormigas artificiales* modeladas a través de agentes de simulación, encuentran soluciones óptimas moviéndose a través de un espacio que represente todas las posibles soluciones. Para esto, emulando el comportamiento de las hormigas reales, registran su posición y la calidad de las soluciones encontradas, para que en posteriores iteraciones de la simulación las siguientes hormigas artificiales puedan encontrar aún mejores soluciones.

Los algoritmos de Colonia de hormigas Optimization han sido aplicados y estudiados en distintos problemas de optimización combinatoria, por ejemplo en ruteo de vehículos o TSP. Tienen como ventaja frente a algoritmos genéticos o Simulated Annealing que funcionan bien en escenarios que pueden cambiar dinámicamente: un algoritmo de Colonia de hormigas puede ser ejecutado continuamente y adaptarse a cambios en tiempo real. Esta característica lo vuelve especialmente útil en problemas de ruteo de redes y sistemas de transporte urbanos.

4.1.2. Aplicación en Sudoku

En cada iteración, cada hormiga comienza con una copia del puzzle, y su objetivo es definir los valores de tantas celdas como sea posible.

Todas las hormigas comienzan en celdas diferentes, elegidas de manera aleatoria, y luego van moviéndose a través de todas las celdas del tablero. Cuando llegan a una celda que no tiene un valor fijo deben tomar la decisión de qué valor asignar. Al hacerlo, las restricciones que este nuevo valor introduce son propagadas por el tablero.

Elegir un valor y no otro es una decisión que está basada en los niveles de feromona que cada valor tiene asignado. Éstos se almacenan en una matriz de feromonas.

Para un Sudoku de dimensión n, definimos una matriz de feromonas global τ en la cual cada elemento se denota como τ_i^k, donde i es el índice de la celda y k $\in [1, n]$ es el posible valor para la celda. τ_i^k representa el nivel de feromona asociado al valor k para la celda i.

Al momento de elegir qué valor colocar en una celda se tiene en cuenta el conjunto de valores disponibles para esa celda v_i. Para realizar la selección existen dos alternativas: realizar una selección *greedy*, tomando el valor para el cuál hay mayor cantidad de feromona en la matriz:

$$f(x) = \underset{n \in v_i}{\operatorname{argmax}} \tau_i^n \tag{5}$$

o bien hacer una elección basada en probabilidades:

$$wp(s) = \frac{\tau_i^s}{\sum\limits_{n \in v_i} \tau_i^n}$$

teniendo en cuenta un factor nivel de *greediness*.

Una vez que se fija el valor para una celda, se intenta reducir la probabilidad de que la siguiente hormiga vuelva a elegir el mismo valor y de esta forma, se previene una convergencia demasiado temprana.

La actualización local de las feromonas, que se realiza para cada hormiga al elegir un valor para una celda, se realiza haciendo

$$\tau_i^s \leftarrow (1 - \xi)\tau_i^s + \xi\tau_0 \tag{7}$$

donde τ_0 es el nivel de feromona inicial para todos los elementos de la matriz de feromonas y ξ es 0.1 (standard en algoritmos de Colonia de hormigas).

Una vez que todas las hormigas cubrieron cada celda del tablero se realiza una actualización global de los niveles de feromona, que solamente premia a la mejor solución encontrada, en forma directamente proporcional a la calidad de la solución. Se caracteriza la mejor solución como aquella que tiene la mayor cantidad de celdas con valores fijados, es decir, la mejor solución es encontrada por la hormiga que elige correctamente el mayor número de veces. Luego, se actualizan todos los niveles de feromonas teniendo en cuenta los valores obtenidos por la mejor solución.

Algorithm 2 Algoritmo ACO para resolución de Sudoku

```
1: procedure RESOLVER
2:     leer el puzzle;
3:     for celdas con valores fijos do
4:         propagar restricciones
5:     end for
6:     inicializar matriz global de feromona;
7:     while puzzle sin resolver; do
8:         dar a cada hormiga una copia local del puzzle;
9:         asignar cada hormiga a una celda diferente;
10:    end while
11:    for numero de celdas do
12:        for cada hormiga do
13:            if la celda actual no tiene un valor fijado then
14:                elegir un valor para esa celda dentro de sus valores posibles;
15:                fijar el valor de la celda;
16:                propagar restricciones;
17:                actualizar la feromona localmente;
18:            end if
19:            moverse a la siguiente celda;
20:        end for
21:    end for
22:    elegir mejor hormiga;
23:    actualizar matriz global de feromonas;
24: end procedure
```

4.2. Algoritmos Genéticos

4.2.1. Introducción

Los algoritmos genéticos están basados en conceptos de evolución biológica, inspirándose en particular en la mutación, entrecruzamiento y selección.

Por lo general, comienzan con una población inicial de soluciones posibles, codificadas en forma de cadenas. Estas cadenas son ordenadas según una función de adecuamiento, que determina que tan buena es una solución.

Las cadenas que representan las mejores soluciones son seleccionadas y recombinadas utilizando un proceso análogo al del entrecruzamiento en la recombinación genética: la primer parte de una cadena de un bit es concatenada con la última parte de una segunda cadena de un bit.

Luego se aplica una mutación de bajo nivel intercambiando de manera aleatoria algunos bits de la cadena. La nueva cadena de solución, junto con los individuos más aptos de generaciones previas, componen la población de la siguiente generación.

El proceso continua hasta que se genere una solución lo suficientemente buena o bien, no pueda realizarse más progreso.

Algunas aplicaciones particularmente interesantes de los algoritmos genéticos se encuentran dentro del dominio de los problemas de scheduling.

4.2.2. Aplicación en Sudoku

Como mencionamos anteriormente, el algoritmo genético codifica una población de soluciones, las ordena según una función de adecuamiento y luego aplica análogos computacionales para los procesos de selección, recombinación y mutación para evolucionar a mejores soluciones.

La codificación en este caso, se basa en cadenas de soluciones que consisten en arreglos uni-dimensionales de símbolos (del 1 al 9). Cada cadena de solución para una grilla de n x n se descompone en n segmentos de n símbolos, donde cada segmento es una subgrilla. Para una grilla de $9x9$, cada subgrilla de $3x3$ se representa como una cadena de tamaño 9.

La población inicial, a partir de la cuál se va intentando evolucionar, se obtiene haciendo, para cada cadena de solución inicial, una permutación aleatoria de sus dígitos. Esto asegura que las subgrillas son soluciones correctas de Sudoku.

Las cadenas de solución son evaluadas por una función de adecuamiento, que en este caso cuenta el número de símbolos repetidos en filas y columnas. Una cadena de solución es mejor que otra si tiene menos repetidos.

Una vez que la población ha sido ordenada según la función de adecuamiento se obtiene un ranking que ordena las soluciones de mejor a peor. Un cierto porcentaje de soluciones es seleccionado para supervivencia y reproducción. Las mejores soluciones son seleccionadas probabilísticamente, de manera tal que incluso las soluciones que quedaron abajo en el ranking tengan posibilidad de reproducirse.

Además, en cualquier momento pueden realizarse entrecrtuzamientos entre las subgrillas y mutaciones. Las mutaciones ocasionan que dos elementos de una subgrilla intercambien posiciones (los elementos fijos, es decir, los que no han sido completados en la resolución del puzzle, no pueden ser intercambiados).

Un algoritmo basado en una metaheurística genética podría seguir los siguientes pasos:

```
1: procedure AlgoritmoGenético
2:     Generar una población inicial;
3:     repeat
4:         Rankear las soluciones y seleccionar solamente aquellas que superaron el ratio de selección especificado
5:         repeat
6:             Seleccionaer aleatoriamente dos soluciones de la población
7:             Seleccionar aleatoriamente un punto de entrecruzamiento
8:             Recombinar las soluciones para producir dos nuevas cadenas de soluciones.
9:             Aplicar las mutaciones a las soluciones
10:        until Se produce una nueva población
11:    until Se halla una solución o se alcanza el máximo número de generaciones
12: end procedure
```

El algoritmo se reinicia si se alcanza un mínimo local del que no pude salir. Las mejores soluciones de cada generación se almacenan para luego ser usadas como poblaciones iniciales.

5. Conclusión

Primero que nada, es reconfortante saber que aún problemas de la categoría NP-Hard pueden ser resueltos de una manera relativamente eficiente y con buena eficacia utilizando metaheurísticas. Eso abre el abanico de problemas que pueden ser atacados. Esto es particular interés en contextos más importantes como el problema del Viajante de Comercio.

Creemos que hay varias aristas para explorar en este problema las cuáles podrían generar mejoras. Por ejemplo:

- Utilizar temperaturas iniciales menores
- Combinar la heurística con otras técnicas, por ejemplo pasos estandarizados como el X-Wing
- Combinar con otras heurísticas, por ejemplo Taboo Search

Siendo una metaheurística muy sencilla de comprender e implementar, consideramos que dio buenos resultados en Sudokus de tamaño pequeño, lo cuál fue muy útil para poder probar rápidamente cómo era el comportamiento y experimentar con distintas mejoras, tales como las distintas vecindades o temperaturas posibles. Nos queda pendiente probar en data sets de Sudokus de tamaño más grande para poder evaluar la performance de nuestro algoritmo y en ese sentido, considerar distintas optimizaciones, como las mencionadas anteriormente.

6. Referencias

- Torosdagli, Neslisah (2016). Ant Colony Optimization for Solving Sudoku Puzzles, Computer Science Department, University Of Central Florida.

- Mullaney, David (2006). Using ant systems to solve sudoku problems, School of Computer Science & Informatics University College, Dublin.
- Weiss, John (2009), Genetic Algorithms and Sudoku, Department of Mathematics and Computer Science South Dakota School of Mines and Technology.
- Lewis, Rhyd (2007). Metaheuristics can Solve Sudoku Puzzles, Napier University, Scotland.
- Lloyd, H. y Amos, M. (2015). Solving Sudoku with Ant Colony Optimization, Centre for Advanced Computational Science, Manchester Metropolitan University, United Kingdom.
- Schiff, Krzysztof (2014). An Ant Algorithm for the Sudoku Problem, Journal of Automation, Mobile Robotics & Intelligent Systems.
- Yato, T. y Seta, T (2003). Complexity and Completeness of Finding Another Solution and Its Application to Puzzles, Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo.