# Diplomatura de Especialización en Desarrollo de Aplicaciones con IA

**Optimizacion Industrial con Computacion Evolutiva**

**Proyecto Final: Resolviendo tableros Sudoku con Algoritmos Genéticos**

En este desafío se pide implementar un algoritmo genético para solucionar un tablero de sudoku proporcionado. Para ello se les está dando un código base (sudokusolver.py). Dicho código contiene una implementación del algoritmo Simulating Annealing para solucionar un tablero dado. Puede probar esa implementación tipeando el comando:

>> python sudokusolver.py -p puzzleA.txt -s sa -a T0=0.5,DR=0.99999,maxIter=100000

Los argumentos que se pueden pasar a sudokusolver.py pueden ser obtenidos con el comando: python sudokusolver.py --help

Se pide implementar el solucionador basado en Algoritmo Genético (método ga\_solver()). Los parámetros que se le debería pasar al algoritmo genético son:

**w**: número de individuos de la población

**Cx**: operador de cruzamiento. Puede ser single (single point) o uniform

**m**: taza de mutación (fracción de individuos que son mutados)

**maxGener**: número máximo de generaciones

Un ejemplo de comando de ejecución con dicho algoritmo implementado seria:

>> python sudokusolver.py -p puzzleA.txt -s ga -a w=10,Cx=single,m=0.1,maxGener=10000

que estaría instanciando el algoritmo genético con una población de 10 individuos, cruzamiento single point , 10% de individuos son mutados en cada generación y un máximo de 10000 generaciones.

Se les está proporcionando 3 tableros sudoku para experimentar (puzzleA.txt, puzzleB.txt y puzzleC.txt).

Para cada uno de los tableros y cada configuración de la tabla de abajo realice 5 ejecuciones de sudokusolver.py con el algoritmo Simulating Annealing como solver y anote los valores promedios indicados. En todos los casos pruebe con T0=0.5 y maxIter=100000

|  |  |  |  |
| --- | --- | --- | --- |
|  | SA Config. 1:  (DR=0.999) | SA Config 2  (DR=0.9999) | SA Config 3  (DR=0.99999) |
| Score de la mejor solución encontrada |  |  |  |
| Tableros evaluados (# llamadas a score\_board()) |  |  |  |
| Tiempo total de procesamiento |  |  |  |

Similarmente, Para cada uno de los tableros y cada configuración de la tabla de abajo realice 5 ejecuciones de sudokusolver.py con el Algoritmo Genético como solver y anote los valores promedios indicados. En todos los casos pruebe con w=10 y maxGener=10000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | AG  solo mutación  (m=1) | AG  solo cruzamiento  (m=0) | | AG  cruzamiento + mutación  (m=0.1) | | AG cruzamiento + mutación  (m=0.5) | |
| single | uniform | single | uniform | single | uniform |
| Score de la mejor solución encontrada |  |  |  |  |  |  |  |
| Generación de la mejor solución |  |  |  |  |  |  |  |
| Tableros evaluados hasta la generación de la mejor solución (llamadas a score\_board()) |  |  |  |  |  |  |  |
| Tiempo total de procesamiento |  |  |  |  |  |  |  |

**TIPS:**

* Se recomienda representar cada individuo con un cromosoma de 9 genes, cada gen representando un bloque diferente del tablero. Mantener siempre **alelos válidos** (9 números diferentes en el bloque)
* En el cruzamiento, los genes (bloques) pueden ser considerados indivisibles. Así, si se hace cruzamiento single point y la posición de cruzamiento es entre el 3er y 4to gen, entonces un individuo hijo heredaría los 3 primeros bloques de un padre y los otros 6 bloques del otro padre. El otro hijo seria creado de forma recíproca. Para cruzamiento uniforme se puede hacer de manera análoga
* La mutación de un individuo debe alterar solo un gen del individuo (escogido aleatoriamente) y debe producir un nuevo alelo válido (respetando no alterar los numeros del tablero original). Puede usar el método make\_neighborBoard() para ello.
* Los individuos iniciales pueden ser creados basándose en el método randomAssign()

**Bonus:**

* Implementar más de un operador de cruzamiento y/o mutación y comparar sus desempeños
* Implementar alguna forma más eficiente del algoritmo genético (demostrar su efectividad en los resultados). Aquí algunas ideas:
  + Ajuste dinámico de la taza de mutación de acuerdo al grado de convergencia que se tiene
  + Usar un pool de operadores de cruzamiento y mutación y aplicarlos de acuerdo a su eficacia en las generaciones anteriores;
  + Cruzamiento con más de dos padres

**Entregables:**

1. Archivos findroute.py y sudokusolver.py para verificar la obtención de los resultados
2. Informe conteniendo:

## Breve descripción de su algoritmo (representación de individuos, operadores de cruzamiento y mutación, etc). Si ha hecho desarrollos para ganar puntos bonus debe indicarlo explícitamente aquí como “Desarrollos para puntos bonus”

## Tablas de resultados para los 3 tableros dados. Si hay algún desarrollo de puntos bonus, extender las tablas de resultados con los resultados de dichos desarrollos.

## Análisis de resultados, comparando los desempeños de las diferentes configuraciones experimentadas.

## Conclusión del mejor método y configuración de parámetros a ser usada para resolver tableros de sudoku.