Memoria TP6 – Parte 1

Inteligencia artificial 2019-2020

Pedro Tamargo Allué (758267)

2019

Contenido

[Parte 1: Algoritmos de búsqueda local 2](#_Toc25175315)

[Algoritmo Hill Climbing 2](#_Toc25175316)

[nQueensHillClimbingSearch\_Statistics 2](#_Toc25175317)

[nQueensRandomReestartHillClimbing 2](#_Toc25175318)

[Algoritmo Simulated Annealing 3](#_Toc25175319)

[Tabla comparativa del porcentaje de éxito con distintos parámetros del Scheduler 3](#_Toc25175320)

[nQueensSimulatedAnnealingSearch\_Statistics 3](#_Toc25175321)

[nQueensHillSimulatedAnnealingRestart 4](#_Toc25175322)

[Algoritmo Genetic 4](#_Toc25175323)

[Tabla comparativa midiendo el tiempo y las iteraciones y variando la población 5](#_Toc25175324)

[Tabla comparativa midiendo el tiempo y las iteraciones y variando la probabilidad de mutación 5](#_Toc25175325)

[nQueenGeneticAlgorithmSearch 6](#_Toc25175326)

[Parte 2: Resolución de Sudokus mediante propagación de restricciones. 6](#_Toc25175327)

[Ejecución de la búsqueda 7](#_Toc25175328)

[Parte 3: Propagación de restricciones y búsqueda local. 7](#_Toc25175329)

[Ejecución del problema 8](#_Toc25175330)

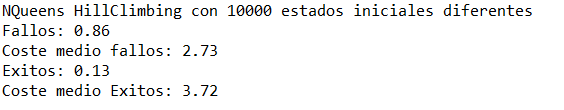
# Parte 1: Algoritmos de búsqueda local

Para la realización de esta tarea se han elegido los algoritmos: Hill Climbing, Simulated Annealing y Genetic Algorithm para resolver el problema de las N-reinas (N = 8).

## Algoritmo Hill Climbing

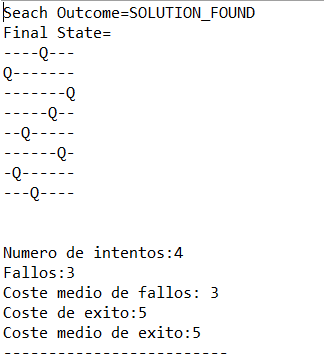
Este algoritmo busca optimizar la solución ya que, partiendo de un estado inicial completo, busca el sucesor que maximice el resultado. Es un algoritmo de búsqueda local ya que solamente se fija en sus sucesores para encontrar el resultado óptimo.

### nQueensHillClimbingSearch\_Statistics



Captura de pantalla del resultado de la ejecución del algoritmo Hill Climbing, partiendo de 10000 estados iniciales distintos.

### nQueensRandomReestartHillClimbing



Captura de pantalla del resultado de la ejecución del algoritmo Hill Climbing con reinicio, cuando no encuentra una solución, busca un nuevo estado inicial y reinicia el experimento.

## Algoritmo Simulated Annealing

Este algoritmo es un símil de la industria metalúrgica, donde, durante los procesos de templado (de aceros), la aleación se enfría drásticamente para maximizar el empaquetamiento de átomos de carbono en la estructura cristalina, lo cual implica mayor resistencia y dureza, así como fragilidad.

En este algoritmo, se utiliza una función , siendo constantes del problema. Para determinar si se pasa de un cierto estado a otro sucesor se utiliza una función de probabilidad dependiente del incremento de los costes entre el nuevo estado y el actual, . Por lo tanto, definimos la función de probabilidad como: , si , siendo la variable que se corresponde con el paso de iteración en la ejecución, o , si .

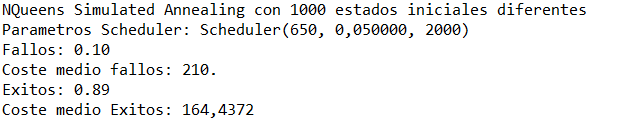
### Tabla comparativa del porcentaje de éxito con distintos parámetros del Scheduler

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NQUEENS Simulated Annealing | | | (midiendo % éxito | limit = 2000) | | |
| **∂ \ k** | **500** | **550** | **600** | **650** | **700** |
| 0,005 | 60% | 62% | 62% | 60% | 57% |
| 0,01 | 82% | 85% | 86% | 84% | 86% |
| 0,05 | 91% | 91% | 90% | 90% | 90% |
| 0,1 | 92% | 90% | 91% | 90% | 90% |
| 0,5 | 88% | 85% | 85% | 86% | 87% |

Para medir el éxito en esta tabla se ha utilizado una *“temperatura”* límite de 2000.

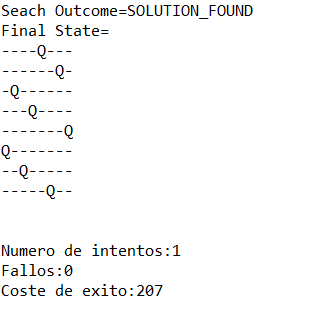
Para la realización de las pruebas se han utilizado los parámetros:

### nQueensSimulatedAnnealingSearch\_Statistics



Captura de pantalla del resultado de la ejecución del algoritmo Simulated Annealing, partiendo de 1000 estados iniciales distintos.

### nQueensHillSimulatedAnnealingRestart



Captura de pantalla del resultado de la ejecución del algoritmo Simulated Annealing, cuando no encuentra una solución, busca un nuevo estado inicial y reinicia el experimento.

## Algoritmo Genetic

Este algoritmo se basa en las mutaciones y cruces entre individuos dentro de una población. Su funcionamiento consiste en que, dada una población inicial y una probabilidad de mutación, seleccionamos a los individuos, usando una y una probabilidad (los individuos cuyo resultado después de aplicar la sea mayor, tendrán más probabilidades de ser elegidos), que transmitirán sus genes a la siguiente generación. Tras seleccionar a los individuos, procedemos a realizar combinaciones, para ello, seleccionamos (de forma aleatoria) el punto de cruce (), dividiendo el individuo en 2 partes. Después procedemos a realizar la mutación, creando dos nuevos individuos combinando las 4 partes de los individuos que se han seleccionado en base a la . En algunos casos pueden ocurrir mutaciones (dependiendo de la probabilidad de mutación), esto implica que alguno de los elementos generados en el paso anterior, cambie alguno de sus “genes” para mantener la diversidad de la población.

### Tabla comparativa midiendo el tiempo y las iteraciones y variando la población

|  |  |  |
| --- | --- | --- |
| **Población** | **Tiempo (ms)** | **Iteraciones** |
| 10 | 4869 | 14452 |
| 20 | 3878 | 3020 |
| 30 | 3614 | 1222 |
| 40 | 3857 | 732 |
| 50 | 3728 | 449 |
| 60 | 5634 | 475 |
| 70 | 8795 | 531 |
| 80 | 5723 | 262 |
| 90 | 7713 | 290 |
| 100 | 6464 | 335 |

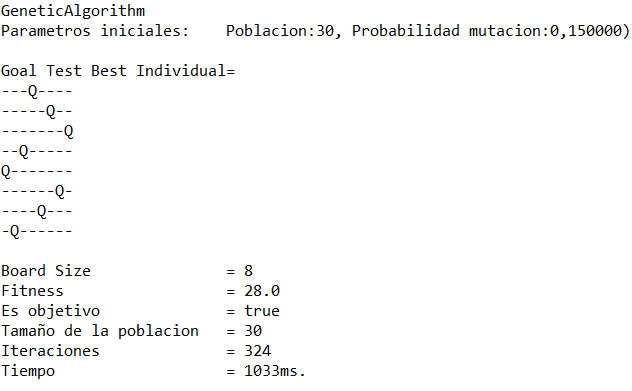
Para la realización de las mediciones, se ha establecido una probabilidad de mutación de 0,15

### Tabla comparativa midiendo el tiempo y las iteraciones y variando la probabilidad de mutación

|  |  |  |
| --- | --- | --- |
| **Probabilidad** | **Tiempo (ms)** | **Iteraciones** |
| 0,15 | 2095 | 1214 |
| 0,25 | 3229 | 1846 |
| 0,35 | 3626 | 2116 |
| 0,45 | 2776 | 1522 |
| 0,55 | 3743 | 2087 |
| 0,65 | 4835 | 2541 |
| 0,75 | 3946 | 2071 |
| 0,85 | 5319 | 2756 |
| 0,95 | 5726 | 3276 |
| 1,05 | 2071 | 1024 |
| 1,15 | 5343 | 3117 |

Para la realización de las mediciones, se ha establecido una población de 30 elementos, ya que, como resultado de la prueba anterior, ha mostrado los mejores resultados.

### nQueenGeneticAlgorithmSearch

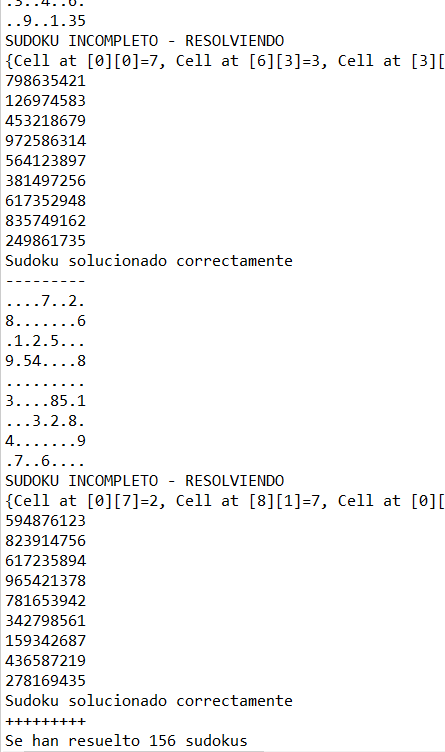


Captura de pantalla del resultado de la ejecución del algoritmo Genetic Algorithm.

# Parte 2: Resolución de Sudokus mediante propagación de restricciones.

Las clases a realizar en esta parte son: . En hereda de y representa una celda del sudoku, tiene . La clase hereda de e impide que una asignación de variables sea igual a otra (como ). La clase contiene el código del método , en este se leen los ficheros de sudokus y se resuelven.

### Ejecución de la búsqueda



Captura de pantalla de la resolución de los 156 Sudokus, ubicados en los 3 ficheros proporcionados junto con el enunciado del trabajo.

# Parte 3: Propagación de restricciones y búsqueda local.

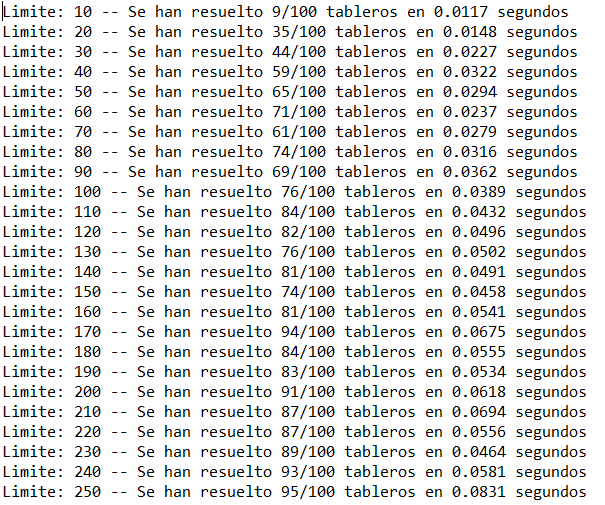
Para la realización de esta parte se han reutilizado las clases creadas en la parte 2, con modificaciones.

La clase hereda de , y contiene 2 enteros, que simbolizan la fila y la columna.

La clase hereda de , y comprueba que una asignación de variables cumple con las restricciones (no son la misma reina, no están en la misma fila y no están en la misma diagonal).

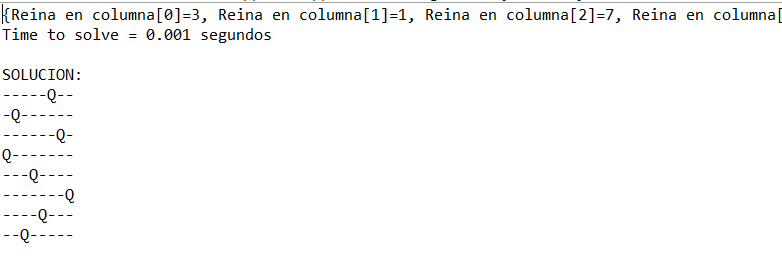
La clase , contiene el método que utiliza para resolver el tablero.

### Ejecución del problema



Captura de pantalla del resultado de la ejecución de la búsqueda CSP con la estrategia *“MinConflicts”* con distinto límite de profundidad.

Se observa que, como resultado de la ejecución, con un límite de 250 pasos podemos obtener la solución para un 95% de los casos.



Captura de pantalla del resultado de una ejecución con límite de profundidad = 100.