Memoria práctica 3

Para esta práctica se han implementado 3 algoritmos de búsqueda distintos para resolver el problema de las N-Reinas, que consiste en colocar tantas reinas como sea posible en un tablero de ajedrez y que ninguna se ataque entre sí.

Hill Climbing Search

El primero de los algoritmos consiste en realizar un número aleatorio de experimentos, que en mi caso ha sido de 10000 aunque se puede poner el número que desees, solo que cuanto mayor el número mejores resultados y luego tras tener esos estados iniciales aleatorios, se calcula el porcentaje y coste medio de los fallos y de los aciertos.

Resultado final:

```
NQueensDemo HillClimbing con 10000 estados iniciales diferentes -->
Fallos: 85,08%
Coste medio fallos: 3,06
Éxitos: 14,92%
Coste medio éxitos: 4,10
```

Random Restart Hill Climbing

Si bien el anterior no arroja el resultado de la resolución, dado que solo calcula costes, este coge el estado inicial y lo va reiniciando hasta obtener el resultado final y va anotando el número de intentos, coste de encontrar la solución y coste medio tanto de éxitos como de fallos.

Resultado final:

```
NQueensDemo HillClimbing Restart -->
Search Outcome = SOLUTION_FOUND
Final State=
---Q----
-----Q--
Q-------
----Q---
-Q------
-------Q
--Q-----
------Q-

Número de intentos = 10
Coste medio fallos: 3,33
Coste éxito: 34
Coste medio éxito: 4
```

Simulated Annealing

Para que el algoritmo fuese más fácil de implementar y más comprensible para mi a la hora de hacerlo, he implementado dos métodos nuevos. Además, es necesario el uso de un Scheduler que usa 3 parámetros distintos acordes a las variables presentadas, que serían:

δ (lo rápido que desciende la temperatura), k (lo que tarda en que empiece a decrecer la temperatura) y los valores que tome la variable T.

Statics

(Explicacción similar a la usada en Hill Climbing Search, pero usando 1000 casos)

Resultado final:

```
NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (10,0.1,500)
Fallos: 44,00%
Coste medio fallos: 58,14
Éxitos: 56,00%
Coste medio éxitos: 44,84
```

Restart

(Explicación similar a la de Random Restart Hill Climbing)

Resultado final:

```
NQueensDemo Simulated Annealing Restart -->
Search Outcome = SOLUTION_FOUND
Final State=
---Q----
------Q-
----Q---
--Q-----
Q-------
-----Q--
-------Q
-Q------

Número de intentos = 1
Fallos: 0
Coste éxito: 47
```

Genetic Algorithm

Para este último se usa un método que busca la solución en base a la probabilidad de mutación y la población inicial del experimento.

Resultado final:

```
NQueensDemo GeneticAlgorithm  -->
Parámetros iniciales:     Población: 50, Probabilidad mutación: 0.15
Mejor individuo=
----Q---
--Q-----
Q-------
------Q-
-Q------
-------Q
-----Q--
---Q----


Tamaño tablero = 8
Fitness = 28.0
Es objetivo = true
Tamaño de población = 50
Iteraciones = 470
Tiempo = 1261ms.
```

Notas finales:

Para poder llevar a cabo la práctica, he modificado otras clases como la de NQueensBoard, para poder agregar la función random() para así poder generar aleatoriamente los tableros iniciales para su posterior uso con los algoritmos de búsqueda solicitados.

También se ha creado una función en el scheduler para ver sus parámetros, ya que antes arrojaba esto por pantalla:

```
Parámetros Scheduler: aima.core.search.local.Scheduler@79fc0f2f
```