## Resumen de las modificaciones realizadas:

* Cambios en la plantilla de la clase EvolutionaryAlgorithm para permitir la ejecución paso a paso de los algoritmos evolutivos.
* Histórico de calidad en la clase EvolutionaryAlgorithm.
* Estrategia de reinicio de algoritmos evolutivos.
* Nueva implementación del algoritm CMA-ES.

## Modificaciones en la clase Evolutionary Algorithm

La implementación de la clase *EvolutionaryAlgorithm* ha sido modificada para permitir la ejecución del algoritmo paso a paso. Para ello se han añadido dos métodos:

* El método ***next*** ejecuta un paso de una generación de acuerdo con el estado actual del algoritmo evolutivo.
* El método ***step*** ejecuta una generación completa.

Además se modificó la implementación del método ***resolve*** pero no su comportamiento. Este método ejecuta una evolución completa utilizando los métodos  ***next*** y ***step.***

En la Figura 1 se muestra el diagrama de estados de la clase ***EvolutionaryAlgorithm*** y las transiciones de estos estados.



Figura - Diagrama de estados de la clase *EvolutionaryAlgorithm*

## Histórico de mejores individuos

Algunos algoritmos, como el CMA-ES, utilizan un histórico de calidades en sus test de parada. Por este motivo se ha añadido un atributo más a la clase ***EvolutionaryAlgorithm***. Este atributo, ***fitness\_history***, almacena un histórico de los mejores individuos en cada generación. Este histórico puede ser utilizado por cualquier clase de la librería de evolutivos. En la versión actual dos test de parada lo utilizan (TolFunStopTest y TolFunHistStoptext).

Para crear este histórico de calidad hay que indicar en el fichero de configuración del algoritmo la capacidad del mismo con el tag ***<FitnessHistoryCapacity>.*** Si este tag no aparece, no se crea histórico de calidad excepto en el caso del algoritmo CMA en el cual es obligatorio. En este caso, si no se especifica capacidad se utiliza el valor por defecto que es *10.0 + 30.0\*D/lambda* siendo D la dimensión del problema.

El histórico de calidades funciona como una cola FIFO. En cada generación se introduce el mejor individuo de la misma. En caso de alcanzar la capacidad máxima se elimina el elemento de la cabeza de la cola y se introduce el nuevo elemento por la cola de la misma.

Posibles mejoras

* Implementar como un plugin de manera que pueda implementarse como una cola FIFO, LIFO o cualquier otro tipo de lista. Por ejemplo, que sólo se almacenen individuos que mejoran la calidad o cualquier otra implementación.

## Estrategias de reinicio

Se ha añadido a la librería JEAF la posibilidad de que los algoritmos evolutivos incorporen estrategias de reinicio. Esto ha sido incluido para dar soporte al algoritmo IPOP-CMA-ES pero su utilización es posible independientemente del algoritmo evolutivo escogido.

Para su implementación se han añadido dos variables a la clase EvolutionaryAlgorithm:

* Una lista de test de reinicio (restartTest) implementados como test de parada ya que su labor es la misma: comprobar si se cumplen ciertos criterios dados y en ese caso lanzar la estrategía de reinicio.
* La estrategía de reinicio (restartStrategy). Este atributo es de la clase RestartStrategy, una interfaz que define los métodos necesarios para la ejecución de la estrategia de reinicio. En la versión actual de la librería se ha implementado una estrategia de reinicio concreta, IPOPRestartStrategy, para dar soporte al algoritmo IPOP-CMA-ES. Esta estrategia incrementa el tamaño de la población del algoritmo y lo reinicia. Cada estrategia es la encargada de decidir si se vuelve a llamar al método configure o no.

Para su configuración hay que añadir en el fichero correspondiente los siguientes tags:

* Para indicar la estrategia de reinicio el tag RestartStrategy con el subtag obligatorio Class y los subtags opcionales dependiendo de la clase concreta seleccionada.
* Para indicar los test de reinicio el tag RestartTests con los subtags RestartTest (uno por cada test de reinicio) y sus subtags correpondientes según la clase concreta seleccionada.

## Nueva implementación del CMA-ES

La implementación del algoritmo CMA-ES en la librería JEAF ha sido modificada para no hacer uso de la implementación original del INRIA. Los elementos añadidos para dar soporte a este algoritmo se muestran en la Figura 2. Además de dar soporte a la versión básica del CMA-ES, JEAF da soporte a las siguientes versiones:

* IPOP-CMA-ES: utilizando para ello las estrategias de reinicio y los test de reinicio implementados para ello.
* sep-CMA-ES: utilizando el parámetro del algoritmo diagonalCovarianceMatrix. Se configura en el fichero de configuración correspondiente utilizando el tag DiagonalCovarianceMatrix. Es un valor entero que indica el número de generaciones en las cuales solamente se recalcula la diagonal principal de las matrices.
  + Si toma el valor de -1 entonces toma el valor por defecto que es 150\*D/N, siendo D la dimensión del problema y N el número de individuos de la población.
  + Si toma el valor de 1, el cálculo es siempre diagonal por lo tanto será la versión sep-CMA-ES.
  + En caso de que dicha variable tome el valor de 0, siempre se calcula la matriz completa.

CMAEvolutionaryAlgorithm

Se ha implementado una nueva clase CMAEvolutionaryStrategy que extiende de EvolutionaryStrategy por ser una estrategia evolutiva. Es necesario implementar esta clase, porque el algoritmo CMA-ES tiene parámetros comunes a varios de sus operadores y se necesita una clase para centralizarlos. Además de servir como nexo de todos los operadores, es la clase encargada de inicializar los valores por defecto de esos parámetros del CMA-ES en el método init().

Operadores implementados

Para implementar el CMA-ES se adaptó la implementación original del INRIA al esquema que siguen los algoritmos evolutivos en la librería JEAF. Para ello fue necesario implementar dos nuevos operadores:

* CMAUpdateDistributionOperator: es el operador encargado de actualizar la matriz de covarianzas y el resto de parámetros que se utilizan para generar la población. Ha sido implementado como un operador de selección, por lo tanto extiende de la clase SelectionOperator.
* CMASamplePopulation: es el operador encargado de generar la nueva población en cada generación. Se ha considerado como un operador de reproducción, por lo tanto extiende la clase ReproductionOperator.

RecombinationType Plugin

En el algoritmo CMA-ES, la actualización de los parámetros se realiza utilizando mu padres de la población cuya influencia en el ajuste de parámetros está ponderada según unos pesos. En la implementación original del INRIA existen tres opciones para generar estos pesos. Estas tres opciones han sido implementadas en JEAF mediante plugins que extienden la clase RecombinationType. Estas tres opciones son:

* Equitativa: implementada en la clase EqualRecombinationType. El peso de cada padre es 1, es decir, todos contribuyen de la misma manera a la actualización de parámetros.
* Lineal: implementada en la clase LinearRecombinationType. El peso de cada padre depende linealmente de su posición en una lista ordenada de acuerdo a su calidad.
* Superlineal (valor por defecto): implementada en la clase SuperlinearRecombinationType. El paso de cada padre depende logarítmicamente de la posición en una lista ordenada de acuerdo con su calidad. Log(mu+1) – log(i+1) siendo i la posición que ocupa en la lista ordenada.

Otras consideraciones

* Se han eliminado las herramientas de logs que habían sido implementadas explícitamente para el CMA. A partir de esta versión ya no son necesarias.



Figura - Elementos añadidos a la librería JEAF para la implementación del CMA-ES