Algoritmos genéticos y evolutivos
Práctica 2

Alejandro Parrado Arribas. NIA: 100383453

Índice

Índice gráficas

Introducción

En este documento aparece el desarrollo de un algoritmo genético evolutivo aplicado a la calibración de los motores de un brazo robot.

El **contexto** de la práctica se basa calibrar los motores de un brazo robot solo con un láser, una cámara y un objetivo de referencia. Inicialmente se trabaja con un brazo sencillo que tiene 4 motores, finalmente se aplicará a un brazo con 10 motores. El brazo robot será para realizar soldaduras de alta precisión.

El **objetivo** es desarrollar al menos una técnica evolutiva que permita calibrar de forma automática y con la mayor precisión posible un brazo robot. El brazo realiza los movimientos programados desde un punto de partida fijo, y una vez finalizados, con el láser activo, la cámara devuelve una medido del error cometido.

Para encontrar una solución se ha decidido utilizar una **estrategia evolutiva** ya que se tratan de datos reales, la función fitness tiene como objetivo **minimizar** el error.

El código de esta práctica se ha realizado con el lenguaje de programación Python3 en el un entorno de ejecución Linux Ubuntu y el control de versiones se ha llevado a cabo en GitHub.

La codificación del problema se basa en **individuos** con números reales en el que cada gen pertenece a al ángulo del motor de un brazo robot, además se cuenta con un vector de varianzas para encontrar la mejor solución. En la práctica se ha tomado un brazo robot que tiene 4 motores por lo que se tendrá **4 genes**.

El resultado de las **funciones fitness** se obtiene de una llamada a un servidor de la Universidad. El resultado de una llamada con el mismo individuo varias veces va a dar el **mismo resultado**, a diferencia de la anterior practica que la función tenía un comportamiento estocástico.

El desarrollo de la practica va a seguir la siguiente estructura; primero se explican las distintas fases del algoritmo, empezando con la inicialización de la población, seguido de los métodos de evaluación, selección, cruce y mutación y por último como resultado se devolverá el **mejor individuo** obtenido durante todas las generaciones.

A continuación, con los datos obtenidos se hacen distintos análisis con gráficas y tablas que ayudaran a encontrar los mejores **hiperparámetros**.

Algoritmo genético - Fases

A continuación se muestran las fases que sigue el algoritmo desde el inicio hasta el fin de la ejecución.

Codificación

En esta primera parte de **codificación**, los brazos robot solo van a tener 4 motores. Para ello, se va a emplear una codificación con números reales, cada dato simboliza el ángulo del motor respecto el origen. Cada individuo está compuesto por un vector de 4 números reales correspondientes a los motores y otro vector de 4 números reales de las varianzas.

Inicialización

El primer paso del algoritmo genético es la creación de la población inicial, para ello se generan tantos individuos como se desee. Además todos los genes se inicializan de manera aleatoria según la codificación descrita.
También, se ha desarrollado una funcionalidad para que se pueda variar el tamaño de la población introduciendo un número como argumento, este dato será guardado en la variable **num_pop**.
La población generada se guarda en una variable de tipo lista llamada **population** que será usada en el resto de las fases del algoritmo, además las varianzas se guardan en otra variable de tipo lista llamada **population_variance**.

Evaluación

Para evaluar a las poblaciones, se dispone de una **función fitness** dada en el enunciado que proporciona un valor acorde al individuo. El objetivo de esta fase es evaluar a todos los individuos de la población para conocer cuál es el mejor.
Para obtener el mejor cromosoma de cada población, pasamos a la web que devuelve los valores de la función de fitness cada cromosoma, uno a uno.
Al final de la evaluación, se obtiene una variable de tipo lista llamada **population_fitness**, en ella se guardan los resultados de evaluación de todos los individuos de la población.
Además en esta practica se ha añadido la funcionalidad de **cache de individuos** de este modo se reducen las evaluaciones totales.

Selección

Para la selección de los cromosomas se ha decidido utilizar el método de los **torneos**.
Su funcionamiento otorga más posibilidades a los mejores individuos de la generación en pasar a la siguiente pero sin dejar de lado cromosomas algo peores para mantener **variabilidad genética**.
En los torneos **participan** tantos individuos como se introduzcan por argumento al ejecutar el programa, este número se guarda en la variable **prob_t**. Los individuos que entran a un torneo son elegidos aleatoriamente de la población y el mejor según su función fitness pasa a la siguiente generación.
Debido a que se va a seguir el **Procedimiento (μ, λ)** de las Estrategias Evolutivas, se van a celebrar tantos torneos como se haya seleccionado en el parámetro **lambda** introducido por argumento **multiplicado por** el número de **familia**, también introducido por parámetro, de este modo cuando se crucen los individuos resultantes del torneo se mantendrá el tamaño de la población inicial.
Es decir, si tenemos 100 individuos, lambda = 40 y familia = 3 , se harán 120 torneos para que en el siguiente paso del cruce, a partir de 3 individuos se genere uno nuevo, de este modo se tendrán 40 individuos que remplazaran a los 40 peores de la población y dará como resultado una población de 100 individuos respetando el tamaño de la población inicial.
El resultado de la fase selección del algoritmo genético se guarda en una variable llamada **population_selection** y las varianzas se guardan en **population_selection_var**.

Cruce

La función de cruce es la encargada de crear la nueva generación que se utilizará en la próxima iteración del algoritmo. El proceso consiste en; a partir de los individuos resultantes del torneo, escoger tantos parientes como se hayan seleccionado por agumento guardado en la variable **num_fam**, que al cruzarlos, generarán **un hijo**, de esta manera el tamaño de la población se mantiene. Al generar individuos con configuraciones genéticas distintas a las obtenidas de los torneos, es la clave para hacer que el algoritmo siga progresando y no se estanque.

Mutación

La mutación consiste en recorrer toda la población y por cada gen de cada cromosoma se muta.

Unión de poblaciones

Se realiza un **reemplazo por inserción**, los descendientes siempre pasan a la siguiente generación y sustituyen a los progenitores menos aptos

Pruebas

En el próximo apartado, se van a estudiar los resultados obtenidos con distintas configuraciones de los **hiperparámetros**. Para la ejecución de las pruebas, se dispone de un **script.sh** programado en bash script para realizar varias llamadas al algoritmo, con variaciones en cada uno de los hiperparámetros disponibles.

Respecto al parámetro del **porcentaje de participación en el torneo**, como ya se evaluó en la anterior practica no será necesario hacer sus pruebas.

El **número de ciclos** para todas las pruebas no va a ser el mismo ya que el algoritmo esta programado para que cuando se encuentre un individuo con fitness 0 se acabe la ejecución.

Porcentaje de lambda

El primer hiperparámetro evaluado en las pruebas es el **lambda**. Es importante encontrar un valor óptimo en el número de **individuos** nuevos que entran a la población que permita avanzar al algoritmo con el aprendizaje pero que no suponga un gran incremento en el número de evaluaciones. Si el número es pequeño, existe la posibilidad de que el algoritmo necesite muchos ciclos para avanzar y encontrar la solución óptima. Por otro lado, si el número de individuos es muy grande se producirían saltos muy grandes en el espacio de búsqueda y el número de evaluaciones seria mayor.
La siguiente gráfica muestra el mejor valor fitness de cada generación de individuos.

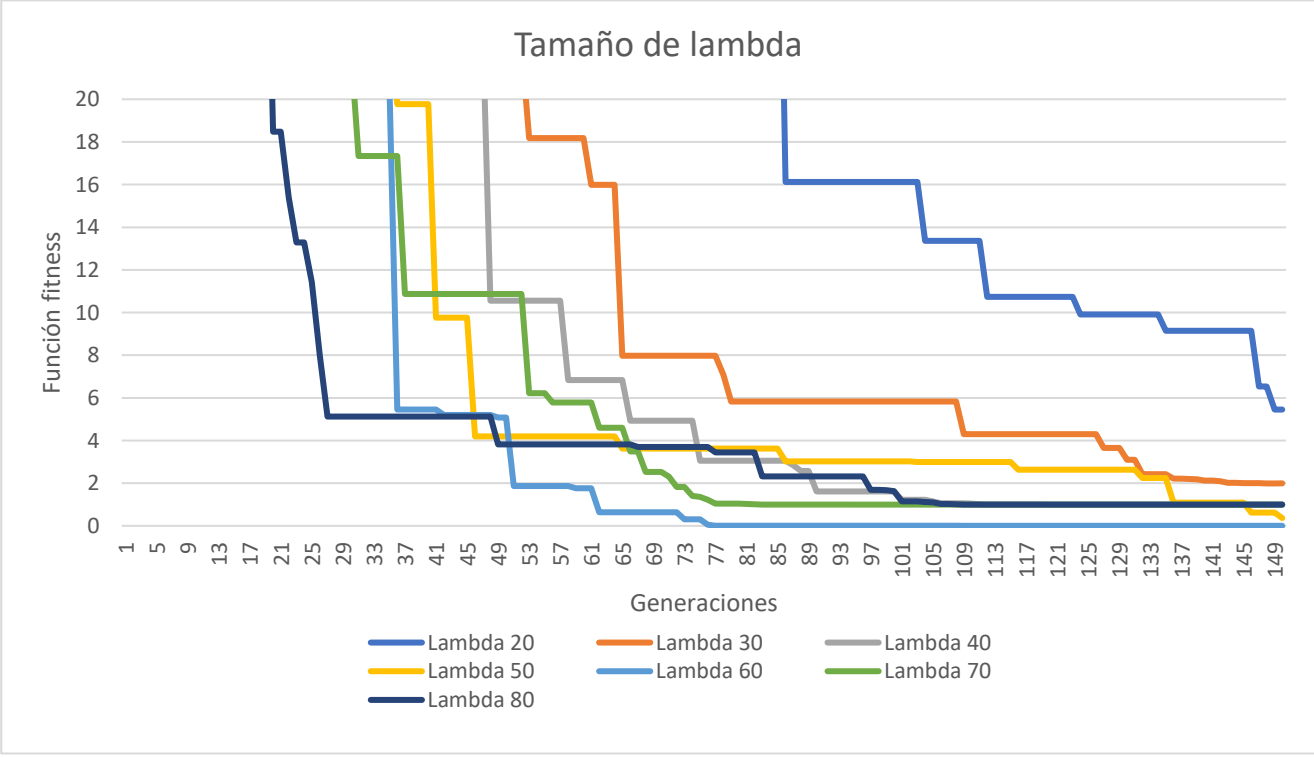

Tabla 1. Tamaño de lambda

Se puede apreciar en los resultados que si el tamaño de **lambda es pequeño**, el algoritmo tarda mucho en evolucionar, perdiendo la posibilidad de encontrar soluciones en un tiempo razonable.
Cuando **lambda es grande** se puede ver que la función fitness decrece a un mejor ritmo, pero se tendrá que ver comprobar si compensan más el número de evaluaciones.
Un dato curioso de las pruebas es que existe un **mínimo local** en 0.994 donde se estancan un las pruebas de lambda 40, 70 y 80

Aunque los resultados son muy parecidos, se decide que el valor **óptimo** para el tamaño de lambda es un valor cercano de 50 para evitar que el número de evaluaciones aumente rápidamente y que no tome muchos ciclos de entrenamiento para acabar.

En cuanto al número de evaluaciones se han obtenido los siguientes resultados:

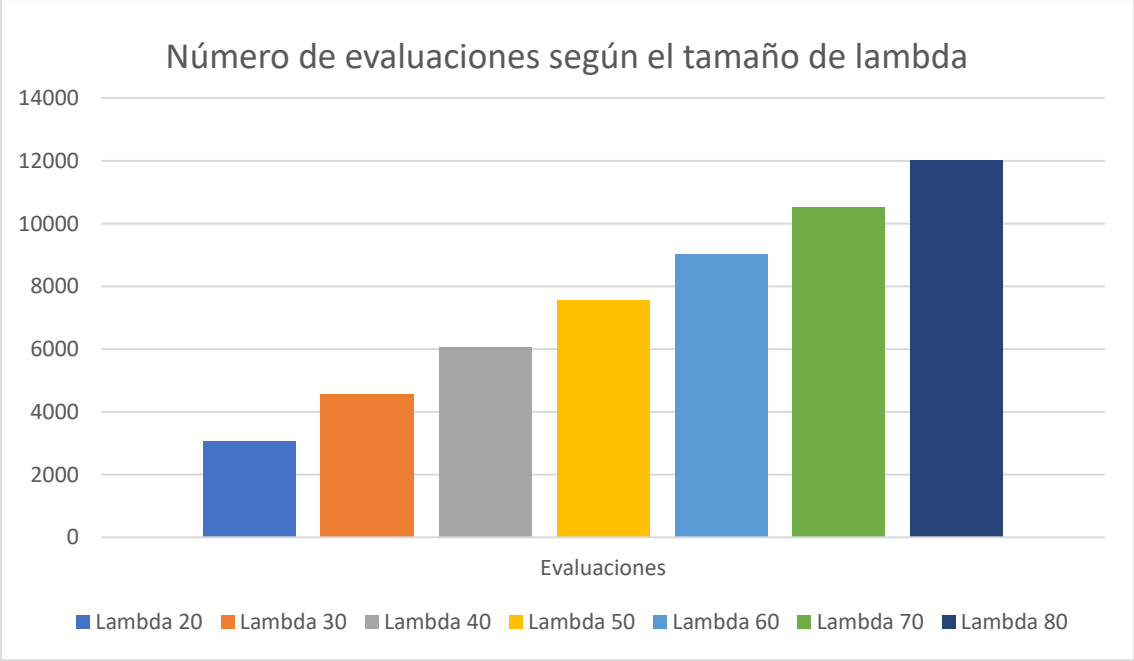

Tabla 2. Número de evaluaciones según el tamaño de lambda

La diferencia entre el número de evaluaciones es la esperada, además es proporcional al tiempo de ejecución.

Tamaño de la población

El tamaño de la población es determinante a la hora de ejecutar un algoritmo ya que el **tiempo de ejecución** crece linealmente con el tamaño de la población.
Si el tamaño de la población es pequeño es posible que el algoritmo se estanque en una solución subóptima ya que al inicializarse la población no cubrirá el suficiente espacio de las regiones de soluciones. Para ello, habría que elegir un tamaño de población que permita al algoritmo y al mismo tiempo no se vea lastrado por los largos tiempos de ejecución.

Los resultados de esta prueba son los siguientes:

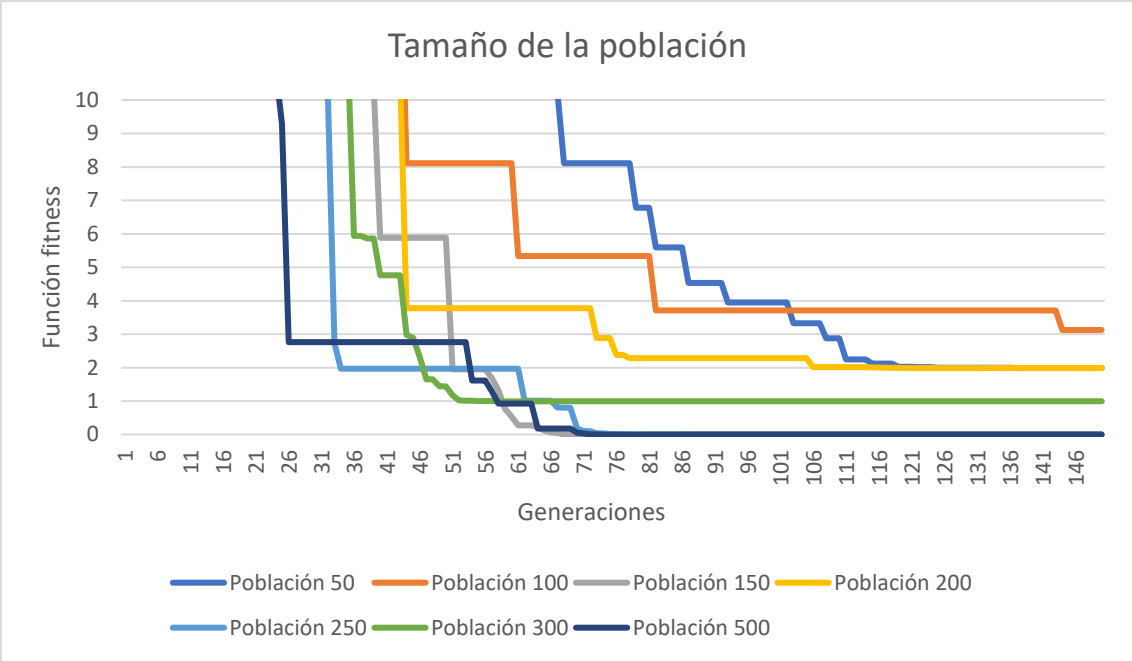

Tabla 3. Tamaño de la población

Debido a que si el algoritmo encuentra un mínimo local muy profundo le va a ser complicado salir. Esto mismo ocurre con el tamaño de población de 200 y 300.
Además para las poblaciones pequeñas como pueden ser 50 y 100 les cuesta más ciclos bajar que el resto porque hacen menos evaluaciones y cubren menos superficies de soluciones.

Respecto a los tiempos de ejecución se han obtenido los siguientes resultados:

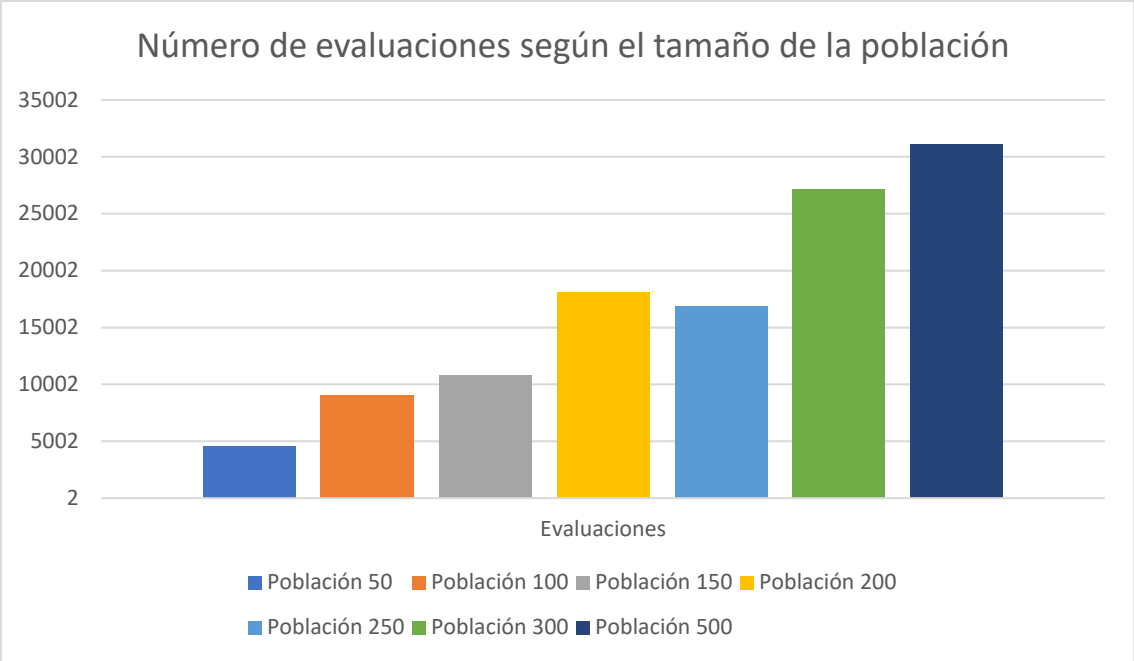

Tabla 4. Número de evaluaciones según el tamaño de la población

Con esta gráfica se confirma la afirmación anterior sobre las poblaciones 200 y 300 que se estancan en mínimos locales ya que sus números de evaluaciones es anómalamente superior al resto.

Tamaño de la familia

El tamaño de la familia es interesante a la hora de ejecutar un algoritmo genético evolutivo ya que es uno de los nuevos hiperparámetros introducidos en esta práctica.

El tamaño de familia se refiere al **número de parientes** que va a tener cada individuo en el cruce. En la anterior practica se tenían 2 pero en esta pueden ser más.

Los resultados de esta prueba son los siguientes:

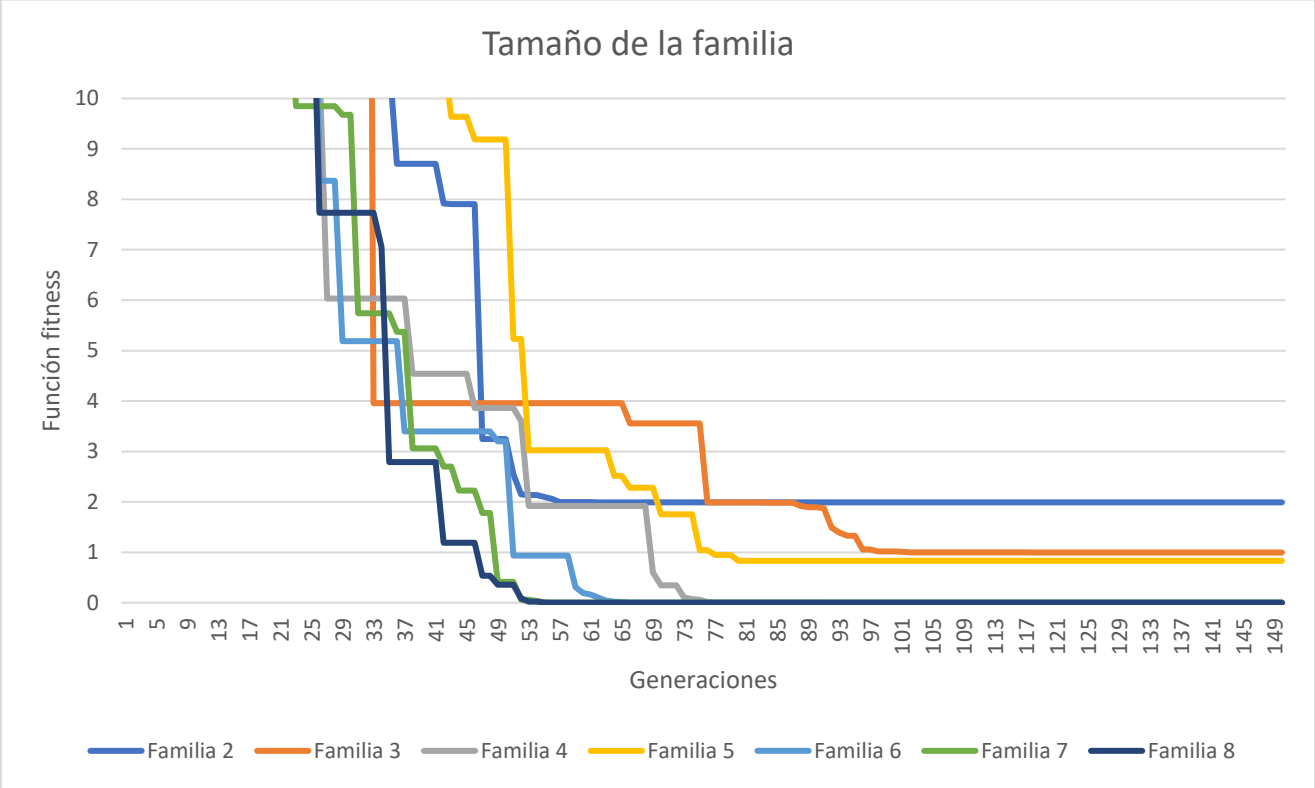

Tabla 5. Tamaño de la familia

Se puede apreciar que los resultados son especialmente buenos para todos los parámetros, quitando los que se estancan en el mínimo local que eso depende más si hay suerte en la inicialización de la población.

Por cada variable no se nota que exista gran diferencia entre los distintos tamaños de poblaciones, pero por lo menos se aprecia que el tiempo de ejecución por generación es mayor cuanto mayor sea el tamaño de la población.
A continuación se muestra el número de **evaluaciones** según el tamaño de la familia:

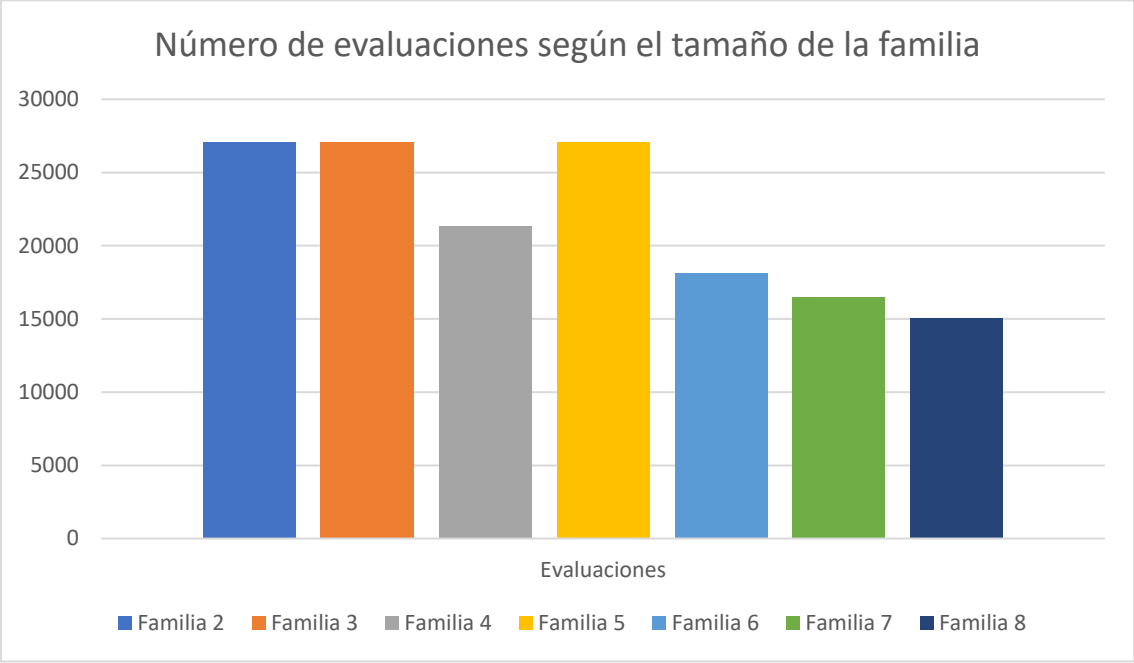

Tabla 6. Número de evaluaciones según el tamaño de la familia

Se puede ver que si elegimos un numero de **familia grande** los resultados son especialmente buenos ya que el algoritmo encuentra mucho antes la solución y el número de evaluaciones es más bajo.

Prueba con la mejor parametrización

A continuación se van a mostrar los resultados obtenidos de las funciones fitness aportadas por los profesores

4 Motores

El mejor fitness obtenido de esta prueba es **0.0**, a continuación se va a mostrar una gráfica con la evolución de los mejores valores de la función fitness de cada generación.

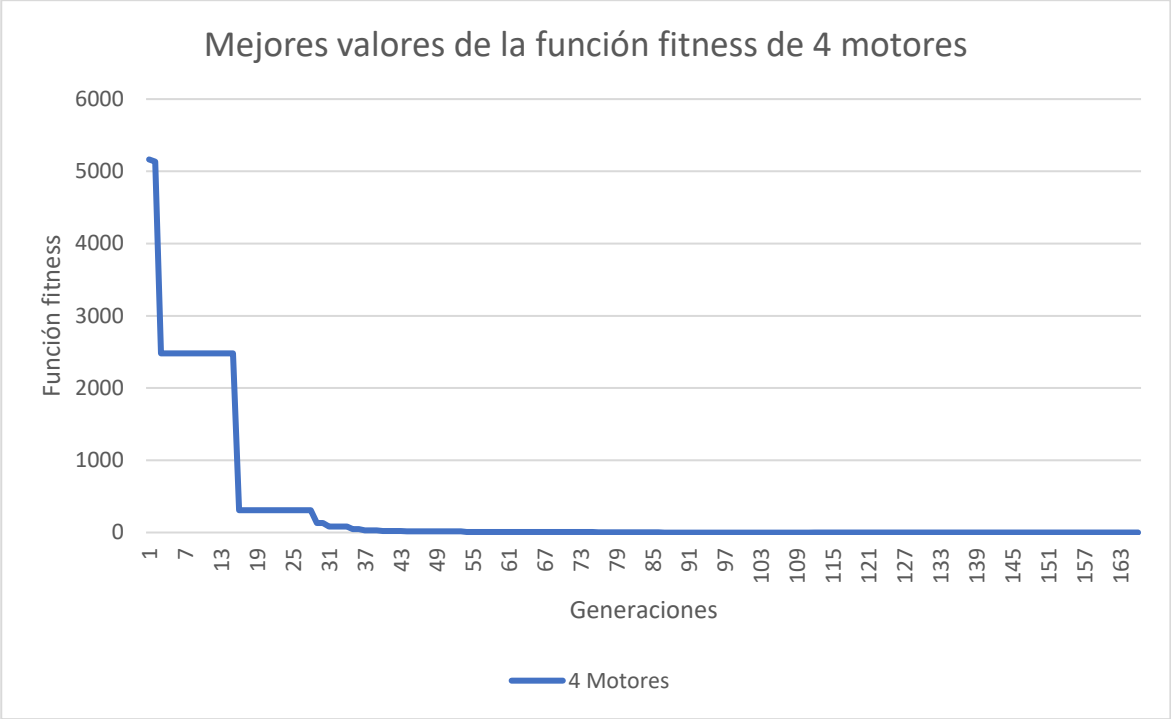

Tabla 7. Mejores valores de la función fitness 4 motores

En total se han requerido 10.000 evaluaciones para esta prueba.

Conclusiones

Con la práctica se ha podido observar cómo los algoritmos genéticos evolutivos son bastante buenos para la tarea de calibrar motores de un brazo robot, consiguiendo una **precisión máxima**, esto demuestra el potencial de estas técnicas.

Como problemas encontrados, cabe mencionar la **lentitud** a las llamadas al servidor ya que si no se ejecutaba el programa desde el servidor de *Guernika* o se configuraba una conexión VPN… el tiempo de ejecución era 10 veces mayor.