Algoritmos genéticos y evolutivos
Práctica 1

Alejandro Parrado Arribas. NIA: 100383453

Índice

Introducción 2
Algoritmo genético - Fases 3
Codificación 3
Inicialización 3
Evaluación 3
Selección 4
Cruce 4
Mutación 4
Pruebas 5
Porcentaje de participantes en los torneos 5
Porcentaje de mutación 7
Tamaño de la población 9
Prueba con la mejor parametrización 11
Alfa 11
Madriz 12
Isidro 12
Conclusiones 13

Índice gráficas

Tabla 1. Porcentaje de participación en el torneo 5
Tabla 2. Tiempo ejecución por generación según el tamaño de los torneos 6
Tabla 3. Porcentaje de mutación 7
Tabla 4. Tiempo ejecución por generación según el porcentaje de mutación 8
Tabla 5. Tamaño de la población 9
Tabla 6. Tiempo ejecución por generación según el tamaño de la población 10
Tabla 7. Número de evaluaciones según el tamaño de la población 10
Tabla 8. Mejores valores de la función fitness Alfa 11
Tabla 9. Mejores valores de la función fitness Madriz 12
Tabla 10. Mejores valores de la función fitness Isidro 12

Introducción

En este documento aparece el desarrollo de un algoritmo genético evolutivo aplicado a la optimización de sensores en Smart cities.

El **contexto** de la práctica se basa en el control de la calidad del aire en la ciudad de Madrid a través de unos sensores que hay localizados en varias estaciones por toda la localidad.
El **objetivo** es optimizar el número de sensores que tiene cada estación para reducir el coste sin perder precisión en los datos. Las estaciones tienen un coste que depende de la base, los sensores y el mantenimiento.
Para encontrar una solución se ha decidido utilizar un **algoritmo genético** evolutivo de tal forma que su función fitness tenga como objetivo **minimizar** el costo.

El código de esta práctica se ha realizado con el lenguaje de programación Python3 en el entorno de ejecución Linux Ubuntu.

La codificación del problema se basa en un **cromosoma binario** en el que cada gen pertenece a un sensor y su valor depende de si el sensor está instalado o no. Cada agrupación de 16 genes representaría una estación.

Se disponen de 24 estaciones repartidas por todo Madrid y 16 sensores por cada estación. Cada **individuo** está compuesto por un total de **384 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 un resultado distinto, esto se debe a que la función tiene un **comportamiento estocástico**.

Los algoritmos genéticos dan buenos resultados a pesar de que allá **ruido** en los datos es por eso por lo que da igual que la función fitness tenga ruido ya que el algoritmo va a ser capaz de aprender.

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, el Ayuntamiento de Madrid solo permite la instalación o la no instalación de los sensores medioambientales. Para ello, se va a emplear una codificación binaria, que representa como '0' la no instalación de un sensor y como '1' la instalación de un sensor. Cada estación está compuesta por 16 genes que representan a cada uno de los tipos de sensores. Por lo tanto, al existir 24 estaciones el tamaño de los cromosomas es de 384 genes.

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 numero como argumento.
La población se guarda en una variable de tipo lista llamada **population** que será usada en el resto de las fases del algoritmo.

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 que cromosoma 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_sorted**, en ella se guardan los resultados de evaluar a todos los individuos de la población; además, se ordenan los valores para que se trabaje más fácil.
La variable population también se ordena de igual modo que se ha hecho con population_fitness_sorted para poder acceder al valor de los fitness de cada individuo con el índice de la lista, el resultado se guarda en una variable llamada **population_sorted**.
La evaluación de los individuos se ha hecho con la **función fitness Madrid** proporcionada por los profesores.

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 argumentos al ejecutar el programa. 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.
El tamaño de población se mantiene durante toda la ejecución del algoritmo; es decir, si tenemos 100 individuos, se harán 100 torneos para que en la siguiente generación haya el mismo número de individuos.
Para tener cada vez más variabilidad genética se ha decidido implementar una sustitución de los peores individuos de toda la población por unos nuevos cromosomas totalmente nuevos generados aleatoriamente.
El resultado de la fase selección del algoritmo genético se guarda en una variable llamada **population_selection**.

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 escoger **un padre y una madre**, que al cruzarlos, generarán dos hijos que mantendrán información de ambos padres, 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.
Debido a que en anteriores fases del algoritmo se ha ordenado la población, para evitar posibles sesgos, se ha optado por utilizar la función **shuffle**.

Mutación

La mutación consiste en recorrer toda la población y por cada gen de cada cromosoma, calcular una probabilidad aleatoria que determinará si el gen debe cambiar o no. Esta probabilidad aleatoria es asignada por argumento cuando se ejecuta el programa y en el caso de que se de que haya mutación, el gen invertirá su valor.

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.

Porcentaje de participantes en los torneos

El primer hiperparámetro evaluado en las pruebas es el **porcentaje de participación** en los torneos. Es importante encontrar un valor óptimo en el número de participantes de los torneos que permita avanzar al algoritmo con el aprendizaje. Si el número es pequeño, existe la posibilidad de que los mejores cromosomas no participen ya que la elección es aleatoria y se podría perder su información genética. Por otro lado, si el número de participantes es muy grande se produciría elitismo y se perdería variabilidad genética.
La siguiente gráfica muestra el mejor valor fitness de cada generación de individuos.

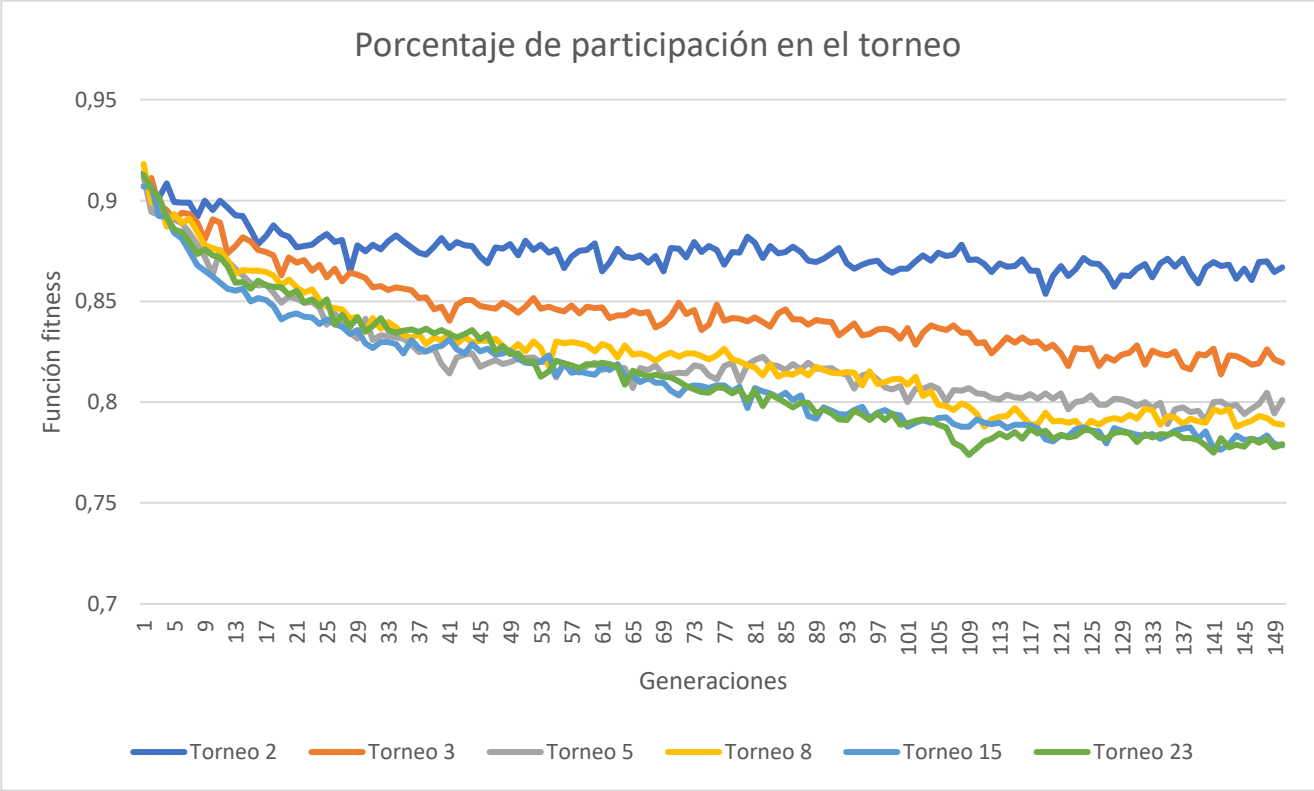

Tabla 1. Porcentaje de participación en el torneo

Se puede apreciar en los resultados que si el tamaño de los torneos es muy pequeño, el algoritmo se **estanca**, perdiendo la posibilidad de encontrar soluciones mejores. Los torneos con más participantes como pueden ser de 15 o 23 tienen mejores resultados ya que como se comentaba anteriormente los mejores individuos tienen

muchas más posibilidades de pasar a la siguiente fase pero como los torneos sean más grandes es posible que se produzca elitismo y se pierda variabilidad genética. Aunque los resultados de los torneos de 15 y 23 sean muy parecidos, se decide que el valor **óptimo** para el porcentaje del tamaño de los torneos es 15 para evitar un posible elitismo.

En cuanto a los tiempos de ejecución por cada generación, se han obtenido los siguientes resultados:

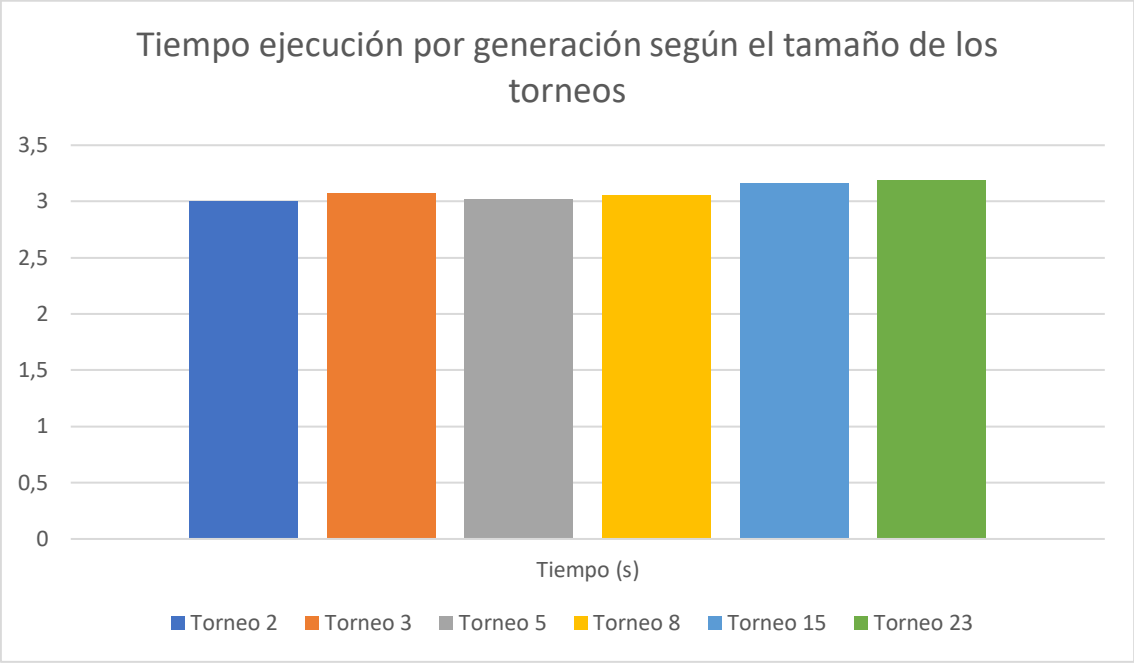

Tabla 2. Tiempo ejecución por generación según el tamaño de los torneos

La diferencia no es muy apreciable en cuanto al tiempo de ejecución por lo que concluimos que el tamaño de los torneos no afecta en los tiempos de ejecución. Esto se puede deber a que las llamadas al servidor de la función fitness es lo que marca la diferencia en el tiempo de ejecución además como la población tiene un tamaño de 100 individuos y un total de 150 ejecuciones, se puede calcular que habrá **15000 evaluaciones**. Como no se cuenta con una cache con los fitness, el número de evaluaciones no va a ser menor del calculado y el tiempo de ejecución por generación no se va a reducir.

Porcentaje de mutación

El segundo parámetro para comprobar es la **tasa de mutación**. Gracias a ella se evita que la solución se estanque en una solución local en fases avanzadas de ejecución, donde la población es menos variada. Entonces, esta tasa de mutación tendría que ser mayor a 0 para evitar que la solución se estanque, pero, por otra parte, tampoco puede ser muy alta porque entonces el algoritmo perdería el aprendizaje que lleva.

Para la experimentación se van a probar porcentajes pequeños. En la próxima tabla se muestra la evolución de las distintas generaciones dependiendo de la tasa de mutación.

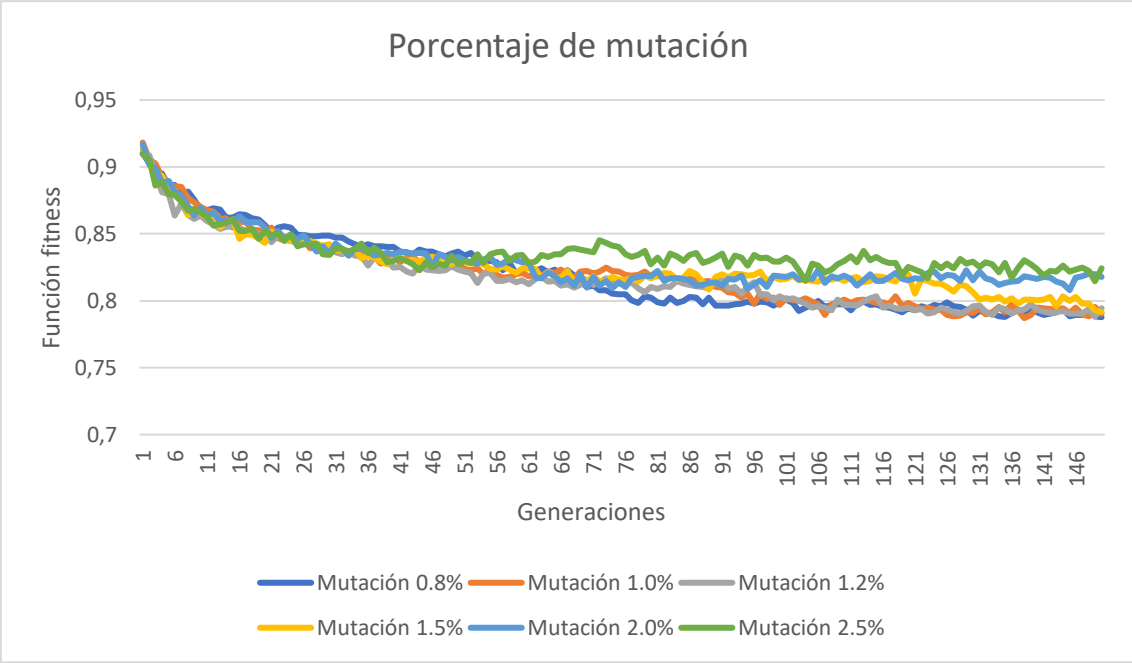

Tabla 3. Porcentaje de mutación

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

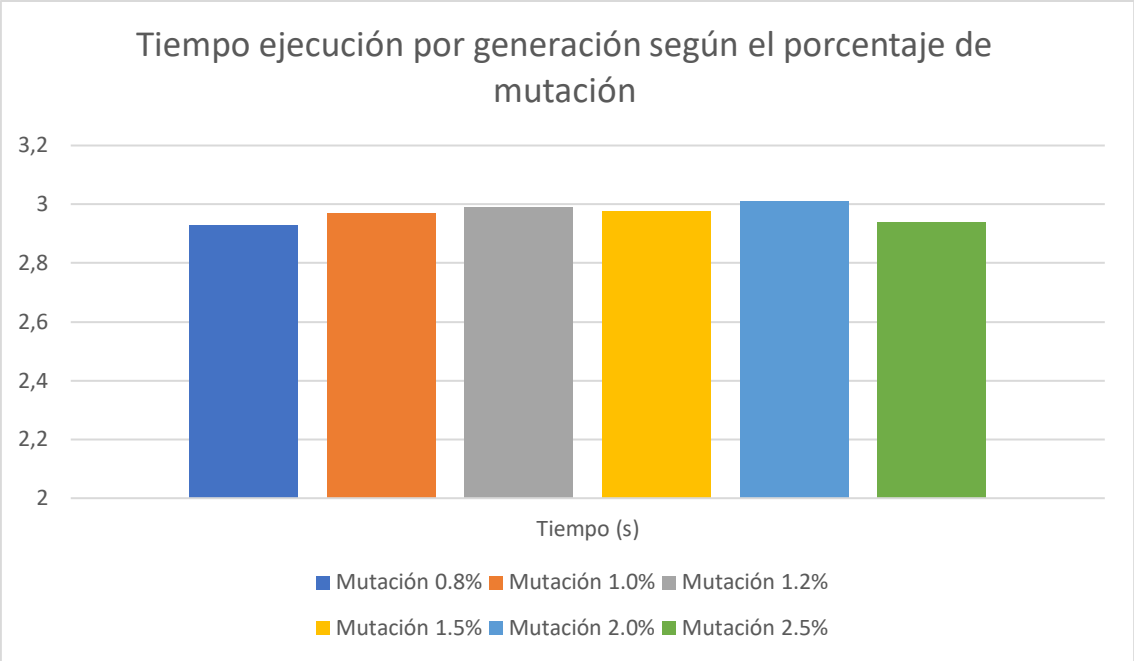

Tabla 4. Tiempo ejecución por generación según el porcentaje de mutación

Se puede apreciar que los resultados no son significativos ya que el número de **15000 evaluaciones** se mantiene, igual que en las pruebas anteriores.

Tamaño de la población

El tamaño de la población es determinante a la hora de ejecutar un algoritmo genético evolutivo 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 se cubriría el suficiente espacio de las regiones de soluciones. Para ello, habría que elegir un tamaño de población que permita seguir aprendiendo al algoritmo y al mismo tiempo no se vea lastrado por los largos tiempos de ejecución.

Los resultados de esta prueba son los siguientes:

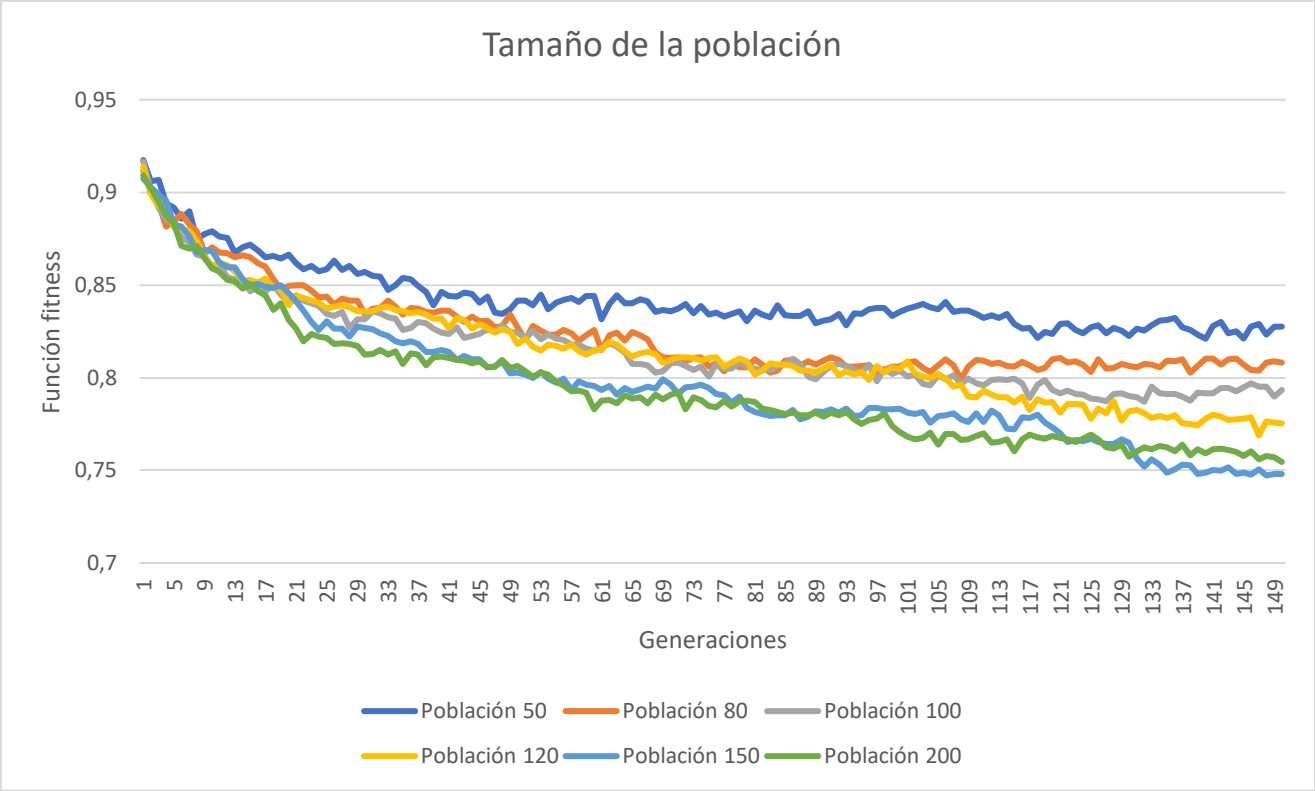

Tabla 5. Tamaño de la población

Como los mejores resultados obtenidos corresponden a los tamaños de población más altos, se elegirá preferiblemente la población de 150 individuos ya que si se eligiera una mayor, el numero de evaluaciones seria también mayor.

Respecto a los tiempos de ejecución se han obtenido los siguientes resultados que se muestra en la gráfica:

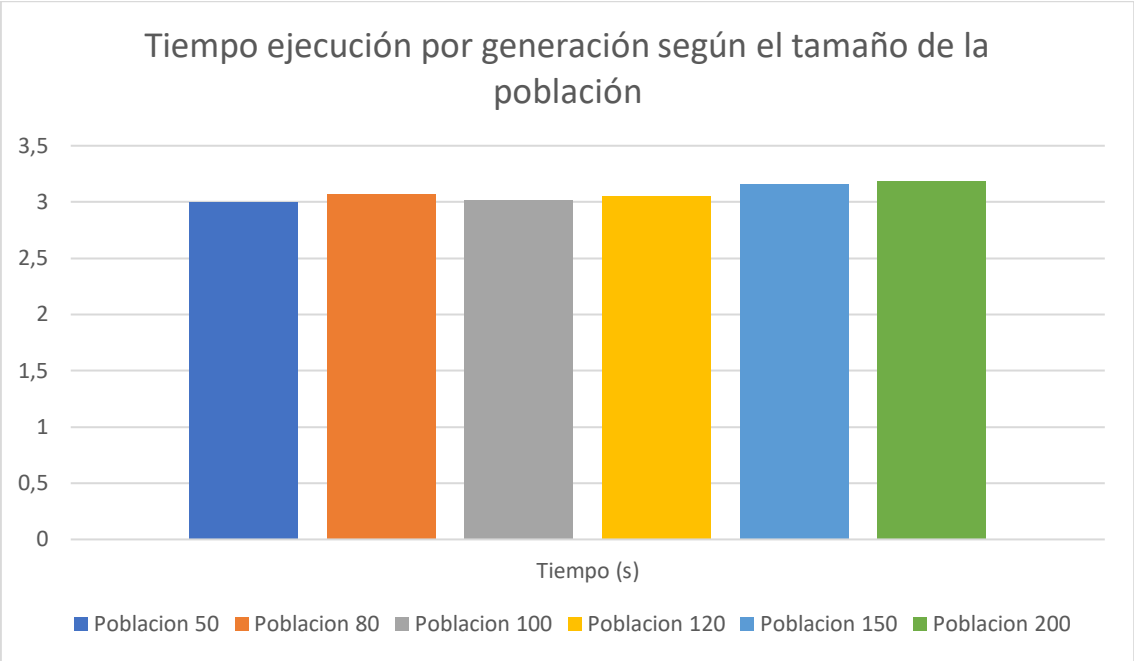

Tabla 6. Tiempo ejecución por generación según el tamaño de la población

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 población:

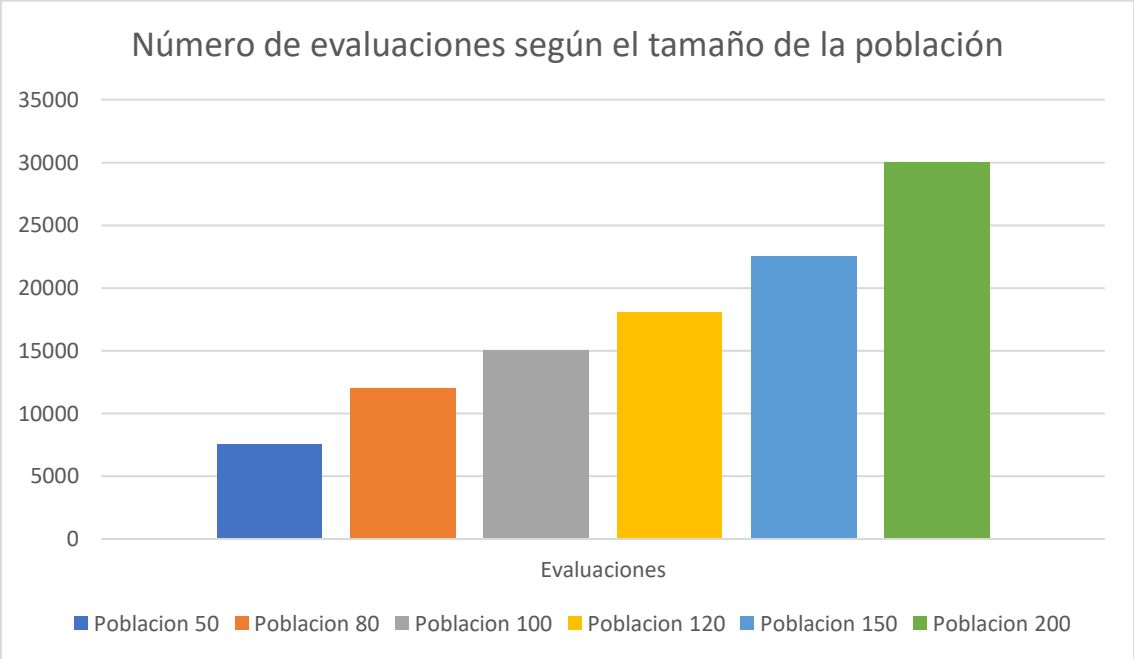

Tabla 7. Número de evaluaciones según el tamaño de la población

Prueba con la mejor parametrización

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

Alfa

El mejor fitness obtenido de esta prueba es **0.8375**, a continuación se va a mostrar una gráfica con la evolución de los mejores valores de la función fitness de cada generación.

Tabla 8. Mejores valores de la función fitness Alfa

Madriz

El mejor fitness obtenido de esta prueba es **0.7406**, a continuación se va a mostrar una gráfica con la evolución de los mejores valores de la función fitness de cada generación.

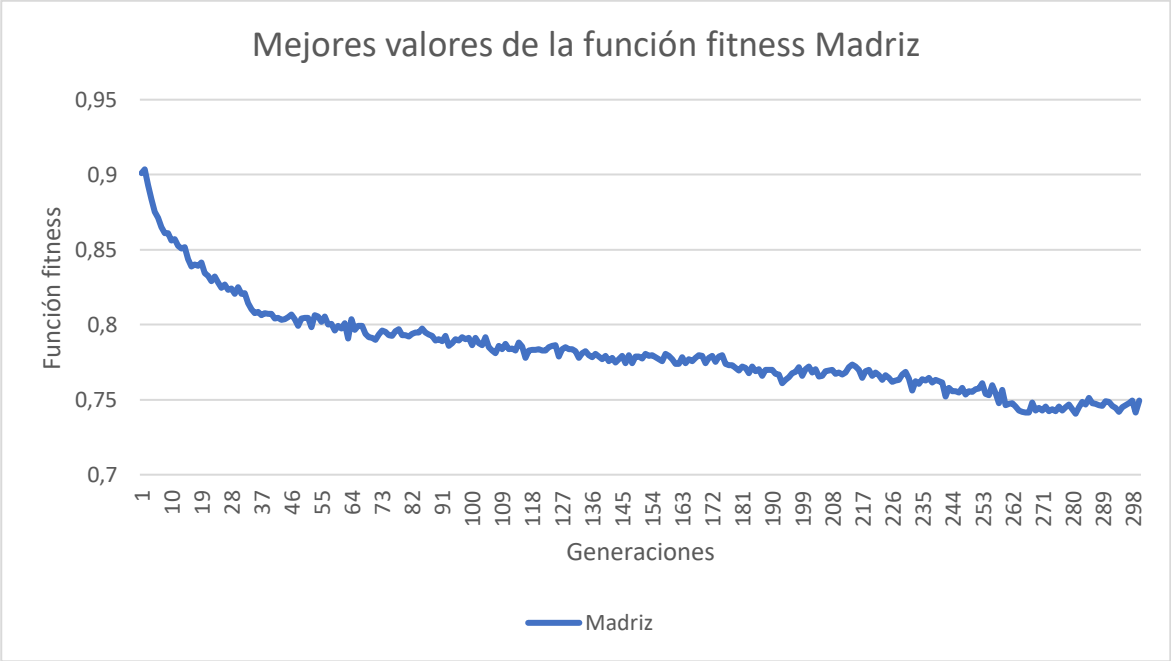

Tabla 9. Mejores valores de la función fitness Madriz

Isidro

El mejor fitness obtenido de esta prueba es **344.59**, a continuación se va a mostrar una gráfica con la evolución de los mejores valores de la función fitness de cada generación.

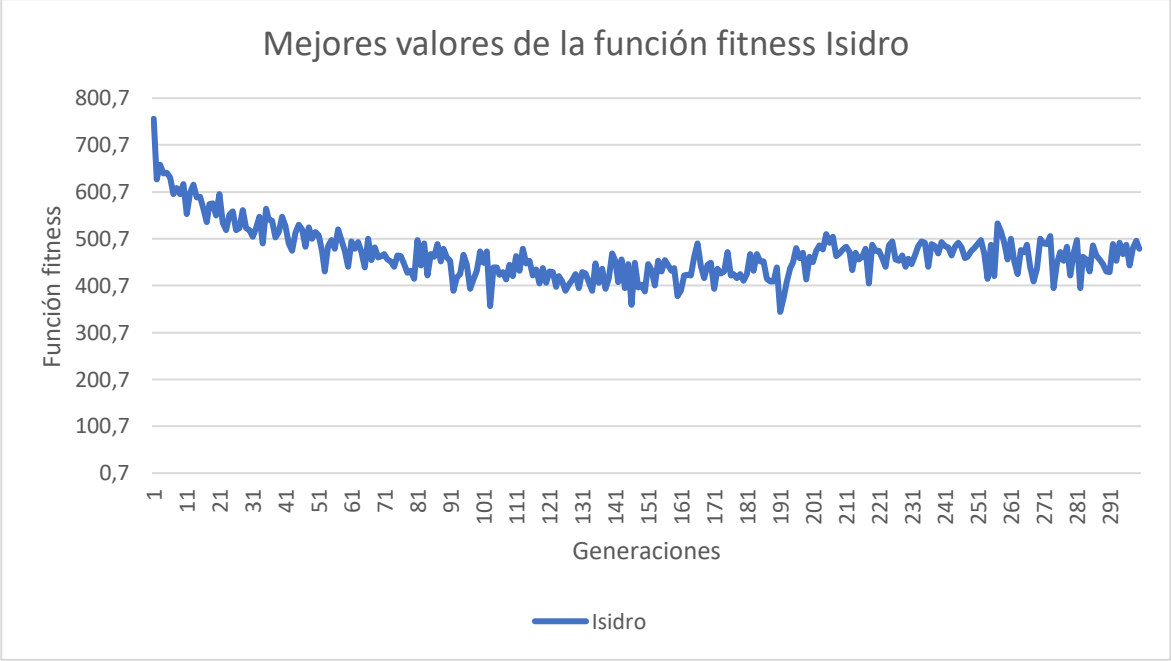

Tabla 10. Mejores valores de la función fitness Isidro

Conclusiones

Con la práctica se ha podido observar cómo los algoritmos genéticos evolutivos son bastante buenos cuando tienen ruido, ya que la función fitness tenía un comportamiento **estocástico** y en cambio el algoritmo continuaba con su proceso de aprendizaje.

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 mucho mayor.