[Fecha]

Luis Buceta Ojeda -- 100303573 | Adrián Rodríguez Grillo -- 100316457

Grupo 83

PRáCTICA 1

Algoritmos genéticos y evolutivos

# INTRODUCCIÓN

En esta práctica nuestro objetivo es la realización de un algoritmo genético que estudie el problema propuesto, que aborda el siguiente objetivo: la colocación de un conjunto de sensores medioambientales en un número de estaciones de forma que el coste de instalación sea el mínimo con el mínimo sacrificio en precisión. Habrá un total de 24 estaciones con una posible instalación de hasta 16 sensores por cada una.

# PRIMERA PARTE

## Codificación

En esta primera parte de la práctica tenemos la situación en la que el Ayuntamiento solo permite la instalación total o no instalación de los sensores medioambientales. Para ello, utilizamos una codificación binaria, que representa como ‘0’ la no instalación de un sensor concreto y como ‘F’ la instalación del mismo.

Cada estación estará compuesta por 16 caracteres o genes que representan a cada uno de los tipos de sensores (definidos por las moléculas que son capaces de detectar). Por tanto, al disponer de 24 estaciones trabajaremos sobre cromosomas de 384 genes.

## Resolución

Para la obtención del resultado óptimo utilizaremos un algoritmo genético, siguiendo los procedimientos para el correcto funcionamiento de este, explicados en clase. Estos son la creación de una población inicial, que será un conjunto de soluciones aleatorias; el estudio de esta población inicial, utilizando la función de evaluación aportada en el enunciado; el método de selección de individuos para la siguiente generación y funciones de cruce y mutación para evolucionar el conjunto de soluciones y poder así encontrar la solución óptima.

### Inicialización

El primer paso del algoritmo genético es la creación de la población inicial, que en nuestro caso se compondrá de 100 individuos (cada uno con 384 genes) generados aleatoriamente según la codificación descrita. Esta población contiene a 100 posibles soluciones que, según avancen las iteraciones, irán adaptándose generación a generación a la función de evaluación (o *fitness*).

**NOTA:** Aunque el número de individuos de la población es de 100 durante el desarrollo y explicación de la práctica, hemos desarrollado la posibilidad de pasar como argumento al programa la cantidad de población a generar, lo que nos permitirá probar el algoritmo con poblaciones de distintos tamaños.

### Evaluación

Para evaluar a la población creada, disponemos de una función de *fitness* dada en el enunciado, por lo que no conocemos el criterio de evaluación que sigue para determinar la eficiencia de la solución. En cada iteración del algoritmo, en esta fase, obtendremos al mejor individuo de la población estudiada y será el estandarte en el que nos fijaremos para comprobar el rendimiento del algoritmo. Para obtener el mejor cromosoma de cada población, pasamos a la web de la función de *fitness* cada cromosoma, uno a uno, y con la función evaluator vamos eligiendo al individuo con un *fitness* menor. Al final de la evaluación, obtendremos el mejor cromosoma junto con su *fitness*.

### Selección

Para la selección de cromosomas para siguientes generaciones, debemos utilizar algunos de los métodos canónicos que hemos estudiado en clase (como la ruleta, los torneos, etc.). En concreto, hemos decidido utilizar los torneos, puesto que su funcionamiento otorga muchas posibilidades a los mejores cromosomas para pasar a la siguiente generación sin dejar de lado el posible sobreajuste que pueda provocar la elección directa de los mejores cromosomas para futuras generaciones. Nuestros torneos funcionan con cuatro participantes, elegidos aleatoriamente de la población generada, y con nuestra función de evaluación escogeremos al mejor de estos cuatro. Queremos mantener el mismo tamaño de población durante toda la ejecución del algoritmo (es decir, tendremos 100 individuos en la siguiente generación).

Ya que por temas del azar es posible que los mejores cromosomas no sean escogidos para participar en los torneos (aunque es una posibilidad bastante baja), hemos decidido añadir elitismo a nuestro proceso de selección. En concreto, introducimos a la siguiente generación directamente (sin pasar por torneos) a los dos mejores individuos de nuestra población. Con esto, nos aseguramos que, como mínimo, la población no empeore su rendimiento según pasen las iteraciones del algoritmo. Al introducir siempre a los dos mejores de la generación anterior, solo debemos realizar un número de torneos equivalente a la longitud de la población menos dos.

Al final de esta fase, tendremos a una población optimizada respecto a la generación anterior, pero la falta de un cambio real entre los cromosomas entre una generación y otro solo conseguirá que el algoritmo se estanque. Para la formación de una nueva generación real, debemos cruzar a los individuos de la población para crear a “hijos” que conserven la información de los “padres” y a la vez contemple nuevas configuraciones para evaluar. Además, añadiremos un elemento de mutación para, de nuevo, dar variedad genética en la población.

**NOTA:** al igual que la población, en la práctica nuestro algoritmo permite elegir al número de participantes para que compitan en los torneos, útil para probar distintas configuraciones del algoritmo. Sin embargo, mantenemos el número de cromosomas afectados por el elitismo (dos), para no sobreajustar el resultado del algoritmo.

### Cruce

La función de cruce es la encargada de generar la nueva población real que será utilizada en la siguiente iteración del algoritmo. El proceso de selección anterior escoge a los mejores padres posibles que, cruzándolos, generarán hijos que mantendrán la información ya evaluada de los padres. Además, este cruce generará individuos con configuraciones genéticas distintas a lo obtenido únicamente por los torneos, y es la clave para hacer progresar el funcionamiento del algoritmo. En caso de no cruzar a los cromosomas padre, el algoritmo solo podría determinar que el mejor *fitness* fuese el mejor *fitness* encontrado en la población inicial (que, generada aleatoriamente, y con 384 bits, hace altamente improbable que pueda ser una solución realmente óptima del problema).

Nuestra función de cruce se basa en el cruce uniforme estudiado en clase. Este es un tipo de cruce donde se escoge a dos padres (nosotros lo cogemos en orden, ya que la población ya fue seleccionada aleatoriamente), y gen a gen determinamos, aleatoriamente, de cuál de los dos padres obtiene el valor del gen el hijo que se está formando. Los hijos, sin embargo, se forman a pares, por lo que lo que no se lleva un hijo se lo lleva el otro (por ejemplo, si el hijo 1 obtiene el gen6 del padre 1, el hijo 2 obtendrá el gen 6 del padre 2). Así generaremos a toda la población de la siguiente iteración, que mantendrá a los dos cromosomas que cogimos por elitismo en la fase anterior (a estos no los cruzamos, los mantenemos igual).

Ya tenemos a la siguiente generación, pero antes de volver a iterar sobre ella, hemos decidido que pueda haber una pequeña mutación en algunos de los genes de los individuos de la población, que nos dará un poco más de diversidad en la población y nos ayuda a prevenir estancamientos.

### Mutación

La mutación de la población se basa en el recorrido de toda la población, y por cada gen de cada cromosoma, calcular una pequeña probabilidad aleatoria que determinará si el gen debe cambiar o no. Esta probabilidad aleatoria es de un 0.01 (un 1%), y en el caso de que se dé, el gen cambiará su valor a otro dentro de la codificación permitida (en el caso de la práctica, o ‘0’ o ‘F’).

Tras terminar esta fase, tendremos una población totalmente distinta a la anterior, pero optimizando los cromosomas para que cada vez estén más cerca del *fitness* óptimo. Sin embargo, todo lo descrito es solo una iteración del algoritmo. En realidad, el algoritmo se ejecuta durante muchas iteraciones, que van usando las poblaciones generadas por la iteración anterior para ir optimizando las posibles soluciones. El paso final para completar el funcionamiento del algoritmo es establecer el bucle de ejecución que realiza cada iteración, y determinar cuándo debe parar el bucle.

**NOTA:** al igual que con la generación de la población inicial y los participantes de los torneos, hemos programado la posibilidad de ejecutar el algoritmo con una probabilidad de mutación elegida por el usuario. Nos servirá para la realización de pruebas con distintas configuraciones del algoritmo.

### Función principal (main)

Nuestra función principal de la función es la encargada de llevar el bucle de ejecución, que realiza las fases anteriores y se encarga también de llevar un registro de cada iteración, con el tiempo de ejecución que tarda en realizarse y el *fitness* obtenido por su mejor cromosoma. Únicamente en la primera iteración, generaremos la población inicial, mientras que en esta y en el resto de iteraciones realizaremos la evaluación, selección, cruce y mutación de la población.