Universidad de Murcia

Material didáctico para la asignatura Sistemas Inteligentes de 3º de Grado en Informática

AUTORES:

- José Manuel Cadenas Figueredo
- María del Carmen Garrido Carrera
- Raquel Martínez España
- Santiago Paredes Moreno

Capítulo 5

Introducción a la librería GAlib

5.1 Introducción

GAlib [1] es una librería de objetos implementados en el lenguaje C++ que tiene como objetivo diseñar y construir algoritmos genéticos. La librería incluye un conjunto de herramientas para poder utilizar algoritmos genéticos y resolver diferentes problemas de optimización utilizando diferentes representaciones y operadores genéticos. Las principales características que nos ofrece esta librería de algoritmos genéticos son:

- Es una librería multiplataforma (Unix, Linux, MacOS, Windows).
- Utiliza una máquina virtual paralela (PVM) para poder trabajar con implementaciones paralelas y distribuidas.
- Permite la parametrización tanto por línea de comandos, como por archivo o en el propio código.
- Permite obtener diferentes estadísticas mediante archivos estructurados tanto de forma "on-line" como "off-line".

De forma general para utilizar la librería se trabaja con dos clases principalmente, la clase `GAGenome` que define al cromosoma, individuo o genoma (términos que usaremos indistintamente a lo largo del documento), y la clase `GAGeneticAlgorithm` que define los diferentes elementos del algoritmo genético. Así, para definir un algoritmo genético utilizando la librería GAlib se deben de seguir estos tres pasos:

- Definir la representación del cromosoma.
- Definir los operadores genéticos más adecuados.
- Definir la función objetivo, también conocida como función "fitness".

Para los dos primeros puntos, la librería GAlib dispone de varias implementaciones, sin embargo la función objetivo debe de ser definida por el usuario. A lo largo de los siguientes apartados se irán desarrollando y detallando los diferentes tipos de representación así como los diferentes tipos de algoritmos genéticos disponibles. Pero antes de describir la funcionalidad de esta librería, hay que comentar cómo se instala y cómo se comienza a utilizar la librería GAlib.

5.2 Creación de un proyecto que use GAlib

En este apartado nos vamos a centrar en detallar cómo configurar e iniciar un nuevo proyecto para poder utilizar la librería GAlib. Aunque existen muchos entornos de programación para el lenguaje C++ y en todos ellos es posible configurar la librería, nos vamos a centrar en configurar la librería en el entorno de programación Dev-C++. También hay que tener en cuenta que si el programador prefiere no utilizar ningún entorno de programación, deberá compilar desde la línea de comandos o mediante un archivo Makefile. En este caso deberá de realizar los includes necesarios según las partes de la librería que desee utilizar.

5.2.1 Instalación de la librería GAlib en el entorno Dev-C++

Antes de comenzar a configurar la instalación de la librería GAlib en el entorno Dev-C++ debemos de proceder a descargar el código fuente de la misma desde su página oficial "http://lancet.mit.edu/ga/". Una vez que tenemos descargado el código, debemos descomprimirlo en una carpeta que en este caso llamaremos "galib247". Al descomprimir el código dentro de la carpeta "galib247" queda el fichero "galib.a" y una carpeta "galib247" (Figura 5.1). Dentro de esta última carpeta "galib247" es donde podemos encontrar el resto de fuentes de la librería, en concreto dentro del la carpeta llamada "ga". Una vez que la librería está descargada y descomprimida, vamos a ver los pasos que hay que seguir para configurarla dentro de un proyecto en el entorno Dev-C++.

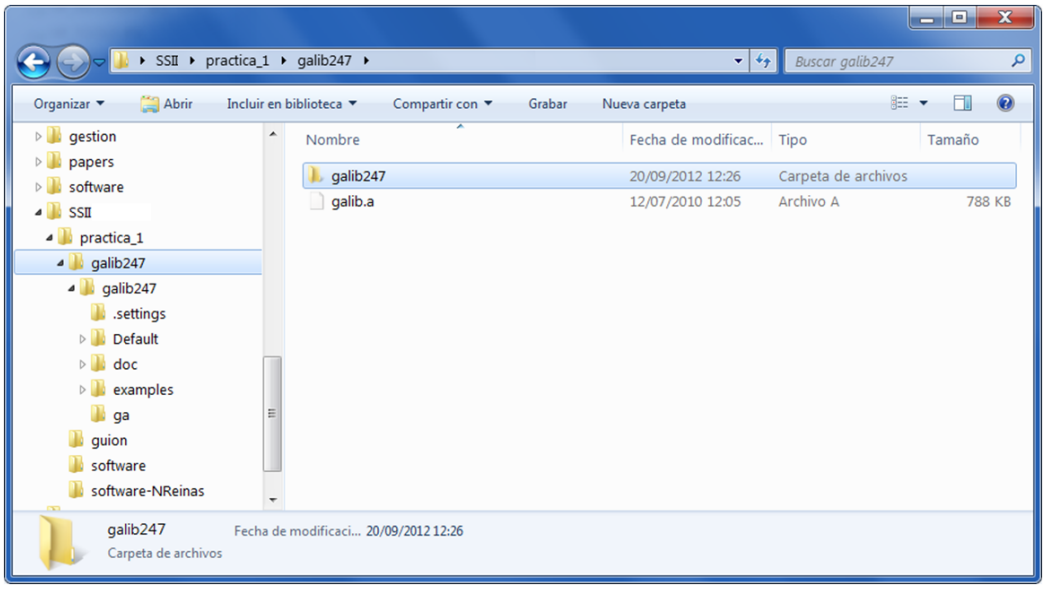

Figura 5.1: Descarga de la librería GAlib

El primer paso es crear un proyecto vacío en el entorno Dev-C++. Una vez que tenemos el proyecto vacío creado, vamos a configurarlo para poder utilizar la librería GAlib. Dentro de las opciones del proyecto, en la parte superior podemos encontrar diversas pestañas (archivos, principal, compilador, parámetros, directorios, etc.). Primeramente debemos de ir a la pestaña de parámetros y añadir en el cuadro de "Linker" la librería estática "galib.a" a las librerías estáticas. Para realizar esta inclusión debemos de pulsar en añadir biblioteca, ir donde se ha descomprimido la librería y seleccionar el archivo "galib.a". Tras incluir el archivo debe de quedar la librería incluida tal como se muestra en la Figura 5.2.

Una vez incluida la librería, debemos de indicarle el path donde se encuentran los ficheros ".h" de la librería. Para indicarle el path de los includes, hay que seleccionar la pestaña de "Directorios" y situarse en la segunda pestaña "directorio de includes". En este último paso debe quedar una configuración similar a la que aparece en la Figura 5.3.

Tras configurar todas las opciones indicadas sólo queda aceptar las modificaciones. En este punto ya podemos comenzar a crear un algoritmo genético utilizando todas las herramientas que nos ofrece esta librería.

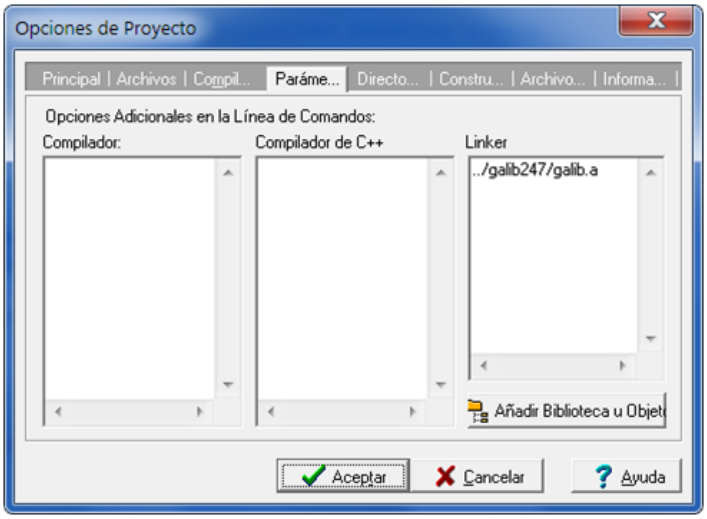

Figura 5.2: Incluyendo la librería "galib.a"

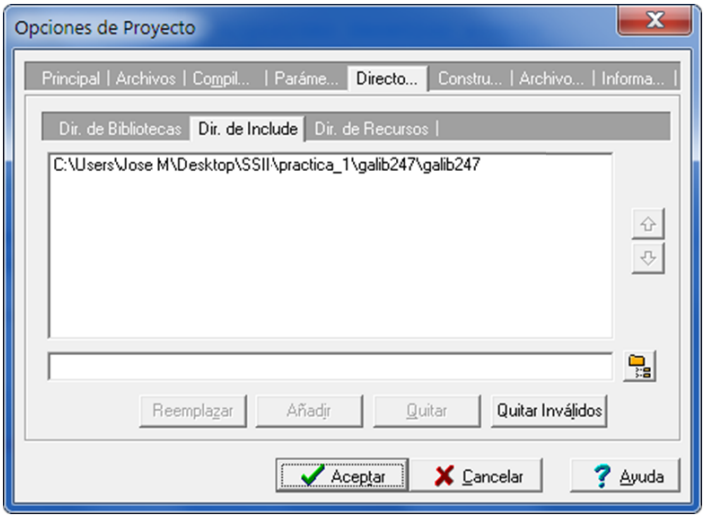

Figura 5.3: Incluyendo el path de los includes de GAlib

5.2.2 Esquema básico de un algoritmo genético utilizando GAlib

Cómo hemos comentado, el objetivo principal de la librería GAlib es ofrecer un conjunto de herramientas para poder implementar de forma sencilla un algoritmo genético. Para resolver un problema usando un algoritmo genético implementado usando la librería GAlib, debemos de elegir cómo va a ser representada la solución de nuestro problema en una estructura de datos, es decir, debemos de escoger la estructura del cromosoma que

en la librería se conoce como `GAGenome`. Tras decidir la representación a utilizar dependiendo del problema, hay que crear un algoritmo genético que trabaje con una población de cromosomas o individuos con la estructura previamente definida. Después mediante los diversos operadores genéticos y una función objetivo, que debe de ser definida por el usuario para evaluar cada uno de los individuos, el algoritmo quedará definido para obtener una solución al problema planteado. Por lo tanto, un proyecto debe de tener como mínimo la siguiente estructura básica:

```
1  //ESTRUCTURA BÁSICA DE UN PROYECTO UTILIZANDO GAlib
2
3  float Objective(GAGenome &);
4  int main(){
5
6      //Definición del cromosoma que en este ejemplo es de tipo string
           binario
7
8    GA1DBinaryStringGenome genome(length, Objective);
9
10     //Definición del algoritmo genético que en este ejemplo es un algoritmo
           genético simple
11
12     GASimpleGA ga(genome);
13
14     //Definición de los parámetros del algoritmo
15
16     // popsize = Tamaño de la población
17     ga.populationSize(popsize);
18     // ngen = Número de generaciones
19     ga.nGenerations(ngen);
20     // pmut = Probabilidad de mutación
21     ga.pMutation(pmut);
22     // pcross = Probabilidad de cruce
23     ga.pCrossover(pcross);
24
25     //Definición del tipo de cruce a aplicar
26
27     ga.crossover(GA1DBinaryStringGenome::UniformCrossover);
28
29     //Definición de tipo de selección
30
31     GATournamentSelector sel;
32     ga.selector(sel);
```

```
33
34      //Llamada al método para que el algoritmo comience a evolucionar
35
36      ga.evolve();
37
38      //Impresión de las estadísticas una vez terminado el algoritmo genético
39
40      cout << ga.statistics() << endl;
41  }
42
43  //Definición de la función objetivo que debe de ser siempre definida por el
        usuario
44
45  float Objective(GAGenome &){
46      ...
47  }
```

Como se puede apreciar, tras definir el tipo de representación del cromosoma y crear el objeto que define el algoritmo genético, solamente hay que definir los parámetros configurables de todo algoritmo genético, como el tamaño de la población, la probabilidad de cruce, de mutación y el número de generaciones. Después se definen los operadores genéticos que queremos que se apliquen y por último se llama al método `evolve()` para que se ejecute el algoritmo.

En los siguientes apartados se comentará más en detalle las posibles representaciones de los cromosomas, los diferentes tipos de operadores genéticos y los tipos de algoritmos genéticos que esta librería nos ofrece.

5.3 Descripción de los problemas utilizados

Dado que a lo largo de esta parte vamos a presentar los distintos elementos que la librería GAlib nos proporciona para implementar un algoritmo genético, usaremos distintos ejemplos para ilustrar dichos elementos. En esta sección definimos estos ejemplos para posteriormente poder usarlos directamente en las distintas secciones hasta completar su resolución mediante algoritmos genéticos apropiados a cada uno de ellos.

5.3.1 Optimización de una función

Como primer ejemplo, veremos un problema que consiste en minimizar la siguiente función sujeta a ciertas restricciones:

$$\begin{array}{ll} \text{Min} & y = x_1^2 - 2 \cdot x_1 + x_2^2 + 2 \\ \text{s.a:} & \\ & 0 \leq x_1 \leq 5 \\ & 0 \leq x_2 \leq 5 \end{array}$$

5.3.2 Problema de las N reinas

Otro ejemplo que resolveremos será el problema de las N reinas que consiste en situar N reinas en un tablero de ajedrez de $N \times N$ sin que se amenacen entre ellas. Una reina amenaza a otra si está en la misma fila, columna o diagonal.

Los posibles movimientos de una reina son los mostrados en la Figura 5.4.

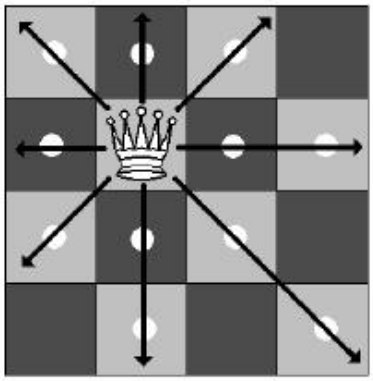

Figura 5.4: Movimientos de una reina

5.3.3 Problema de la mochila

El último ejemplo con el que trabajaremos será el problema de la mochila.

Supongamos que tenemos n tipos distintos de ítems, que van del 1 al n. De cada tipo de ítem se tienen q_i ítems disponibles, donde q_i es un entero positivo que cumple $1 \leq q_i \leq \infty$.

Cada tipo de ítem i tiene un beneficio asociado dado por v_i y un peso (o volumen) w_i. Usualmente se asume que el beneficio y el peso no son negativos.

Por otro lado se tiene una mochila, donde se pueden introducir los ítems, que soporta un peso máximo (o volumen máximo) W.

El problema consiste en meter en la mochila ítems de tal forma que se maximice el beneficio total de los ítems que contiene siempre que no se supere el peso máximo que puede soportar la misma. La solución al problema vendrá dada por la secuencia de variables $x_1, x_2, ..., x_n$ donde el valor de x_i indica cuántas copias se meterán en la mochila del tipo de ítem i.

El problema se puede expresar matemáticamente por medio del siguiente programa lineal:

$$\begin{array}{ll} \text{Max} & \sum_{i=1}^{n} v_i x_i \\ \text{s.a:} & \\ & \sum_{i=1}^{n} w_i x_i \leq W \\ & 1 \leq q_i \leq \infty \end{array}$$

Si $q_i = 1$ para $i = 1, 2, ..., n$ se dice que se trata del problema de la mochila 0-1. Si uno o más q_i es infinito entonces se dice que se trata del problema de la mochila no acotado también llamado a veces problema de la mochila entera. En otro caso se dice que se trata del problema de la mochila acotado. En adelante, siempre trataremos con el problema de la mochila 0-1.

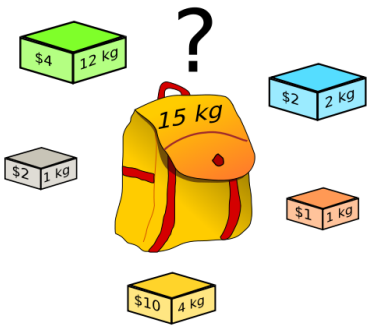

Figura 5.5: Mochila 0-1

5.4 Generadores aleatorios y semillas

La librería GAlib incluye diversos métodos para generar números aleatorios. Antes de utilizar cualquier método para generar números aleatorios tenemos la posibilidad de

establecer una semilla para el generador. El método para establecer la semilla es `void GARandomSeed(unsigned s = 0)`. Si este método se llama sin parámetros o con valor 0, la semilla que se establece es la hora actual multiplicada por el ID que el sistema haya asignado al proceso. Cuando la librería se utiliza en sistemas donde no se asigna un ID a los procesos, solamente se utiliza como semilla el tiempo. Para establecer una semilla definida por el usuario, simplemente hay que pasar tal valor como parámetro al método. Si durante la ejecución del proceso se quiere cambiar de semilla, hay que llamar otra vez al método con un nuevo valor de la semilla (si se llama con el mismo valor la llamada no tiene efecto) y a partir de ese momento la semilla se verá reinicializada. Si queremos reestablecer el generador con la misma semilla, debemos realizar una llamada al método `void GAResetRNG(unsigned int s)` con un valor distinto de 0.

Los métodos disponibles en la librería para obtener números aleatorios son los siguientes:

- `int GARandomInt()`
- `int GARandomInt(int min, int max)`
- `double GARandomDouble()`
- `double GARandomDouble(int min, int max)`
- `float GARandomFloat()`
- `float GARandomFloat(int min, int max)`
- `int GARandomBit()`
- `GABoolean GAFlipCoin(float p)`
- `int GAGaussianInt(int stddev)`
- `float GAGaussianFloat(float stddev)`
- `double GAGaussianDouble(double stddev)`
- `double GAUnitGaussian()`

Los métodos `GARandomInt, GARandomDouble, GARandomFloat` que no reciben parámetros devuelven un número aleatorio en el intervalo 0 y 1. Por el contrario, los métodos `GARandomInt`, `GARandomDouble`, `GARandomFloat` que reciben los parámetros `min` y `max`, devuelven un número aleatorio entre el valor mínimo y máximo establecido, incluyendo ambos valores.

El método `GAFlipCoin` devuelve un valor booleano basado en un lanzamiento de moneda con sesgo p. Por ejemplo, si `p` es 1, el método siempre devuelve 1, si `p` es 0.75 el método devuelve 1 con una probabilidad del 75 %. Si el usuario quiere utilizar este tipo de valores aleatorios, el método más eficiente es `GARandomBit`. Este método utiliza el método numérico descrito en [5].

El resto de métodos toman como base una función de Gauss y devuelven un número aleatorio a partir de una distribución gaussiana tomando como desviación típica la que se indica cómo parámetro en dichos métodos. Dependiendo de la precisión necesaria en el número aleatorio, el método elegido será diferente, siendo el método `GAGaussianDouble` la más precisa. Por último, el método `GAUnitGaussian` devuelve un número aleatorio a partir de una distribución de Gauss de media 0 y desviación 1.

Sin tener en cuenta el método utilizado para obtener números aleatorios, es importante recordar que se debe siempre establecer una semilla distinta de 0 antes de comenzar con la ejecución del algoritmo para así poder reproducir una ejecución del mismo, bien para obtener de nuevo la solución o para ir ajustando los parámetros del mismo.