Estudio del Tiempo Asintótico de Algoritmos Mediante MVC

Alejandro Rodríguez, Rubén Palmer y Sergi Mayol

Resumen— Esta aplicación presenta una gráfica de los costes computacionales asintóticos de tres algoritmos variando el tamaño *N* del vector de datos. Los dos primeros encontrarán la moda con un coste asintótico entre $O(n)$ y $O(n \log n)$. El tercero implementará el producto vectorial del mismo vector por él mismo, representando así un coste $O(n^2)$.

Vídeo - ver vídeo

1. Introducción

En este artículo explicaremos el funcionamiento y la implementación de nuestra práctica estructurando y dividiendo sus partes en los siguientes apartados:

Inicialmente, se citarán las librerías usadas en este proyecto. Para aquellas que se hayan desarrollado específicamente, se expondrá una breve guía de su funcionamiento y uso.

Seguidamente, se explicará cuál es la arquitectura, interfaces, funcionamiento e implementación de nuestro **M**odelo **V**ista **C**ontrolador (MVC) así como los métodos de mayor importancia. Para facilitar la legibilidad, se separará en cada uno de los grandes bloques de la estructura; en este caso Modelo, Vista, Controlador y Hub

Finalmente, se mostrará una breve guía de como usar nuestra aplicación.

2. Librerías

En esta sección se explicarán las librerías implementadas y usadas para llevar a cabo el desarrollo de las prácticas y permitir la reutilización de las mismas durante el transcurso de la asignatura.

El principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo "overhead" al rendimiento del programa.

2.1. Better swing

Better swing es una librería de derivada de java swing que permite un desarrollo más amigable y sencillo de interfaces gráficas, al estar inspirado en el desarrollo web (HTML y CSS), concretamente, en el framework de "Bootstrap". Se basa en "JfreeChart" para pintar las gráficas de líneas creando "wrappers" para facilitar su uso y manejabilidad.

2.1.1. Funcionamiento

El funcionamiento del paquete es muy sencillo, básicamente, la idea es crear una o varias ventanas con una configuración deseada y acto seguido ir creando y añadiendo secciones (componentes), donde posteriormente se pueden ir borrando y actualizando los componentes individual o grupalmente. Por ello, este paquete proporciona una serie de métodos para la ventana y las secciones.

¿Cómo funciona la ventana?

Para hacer que la ventana funcione es necesario crear una instancia del objeto `Window`,

inicializar la configuración de la misma empleando el método `initConfig` recibiendo por argumento la ruta donde se encuentra el archivo. En caso contrario se empleará la predeterminada. Finalmente, para que se visualice la ventana se realizará con el método `start` que visualizará una ventana con la configuración indicada anteriormente.

En el caso de querer añadir componentes como una barra de progreso, un botón o derivados, es tan sencillo como crear una sección y emplear el método `addSection`, el cual recibe por parámetro la sección a añadir, el nombre de la sección y la posición de este en la ventana.

¿Cómo funciona la sección?

Las secciones son instancias de la clase `Section` que permiten formar los diferentes componentes de la ventana.

El funcionamiento de una sección es muy sencilla, se trata de instanciar un objeto de la clase `Section` y llamar a algún método de esta clase, pasando los parámetros adecuados, y finalmente añadir la sección a la ventana.

Otros

Adicionalmente, se dispone de una clase llamada `DirectionAndPosition`, que constituye las posibles orientaciones y direcciones que una sección puede tener.

2.1.2. Implementación

Para el desarrollo de esta librería se ha centrado en la simplicidad hacia el usuario final y la eficiencia de código mediante funciones sencillas de emplear y optimizadas para asegurar un mayor rendimiento de la interfaz de usuario. Este paquete se divide en dos principales partes:

Window: Consiste en un conjunto de funciones para la creación y configuración de la ventana.

Section: Consiste en un conjunto de funciones base para la creación de secciones en la ventana.

La implementación de la `Window` consiste en diversas partes: gestión de teclas, configuración, creación y actualización de la ventana y creación de las secciones.

La gestión de las teclas se realiza mediante una clase llamada `KeyActionManager`, que implementa la interfaz `KeyListener`. Esta clase permite gestionar los eventos de teclado de la ventana y se utiliza para la depuración y el desarrollo del programa.

En cuanto, la configuración, creación y actualización de la ventana y la creación de las secciones, se han desarrollado una serie de métodos que envuelven a un conjunto de funciones propias de java swing. Por tanto, con este conjunto de métodos y funciones se obtiene la clase `Window`.

A continuación se explicarán los principales métodos de `Better swing`:

`initConfig` es un método que permite cargar la configuración de la vista y crea el marco de la vista, incluyendo apariencia, posición, tamaño, icono, color de fondo y manejo de eventos de teclado.

`start` permite la visualización de la ventana, haciendo que la ventana sea visible para el usuario.

`stop` actua como wrapper de `JFrame::dispose`.

`addSection` permite añadir una sección a un objeto `Window` indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`updateSection` permite actualizar una sección indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`deleteComponent` permite borrar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintComponent` permite repintar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintAllComponents` repinta todos los componentes de la ventana.

La implementación de la `Section` consiste en diversas partes, las cuales son las siguientes: Los métodos que permiten crear las secciones ya configuradas y las clases en las que se basan los métodos anteriores.

La propia librería contiene muchas más clases y métodos que si el lector desea explorar puede acceder a su documentación a través del código fuente proporcionado en la entrega de la práctica.

2.1.3. Manual de uso

En este apartado se describe como emplear la librería para su correcto funcionamiento.

Para emplear la librería es tan sencillo como crear una instancia de la clase `Window`, llamar al método `initConfig` indicando el fichero de configuración de la ventana, si se desea, en caso contrario se deberá pasar un `null` y se emplearán la configuración por defecto, y finalmente llamar a la función `start` cuando se desee inicializar la ventana.

Es importante inicializar la configuración antes de ejecutar la función `start`, ya que en caso contrario se producirá una excepción. A continuación se muestra un sencillo ejemplo:

```
Window view = new Window();
view.initConfig("config.txt");
view.start();
```

Como se observa, la librería permite cargar la configuración e iniciar la ventana independientemente, permitiendo una mayor flexibilidad.

Además, en el caso de querer reiniciar la configuración, cambiar la visibilidad de la ventana o guardar el contenido de esta sin tener que volver a compilar el código, se puede realizar a través de unos atajos de teclado, que son los siguientes:

Q: Cerrar programa.
R: Reiniciar configuración.
V: Cambiar visibilidad.
G: Guardar contenido.

Por ejemplo, al cambiar la configuración y reiniciar la ventana se aplicarán los cambios automáticamente sin compilar de nuevo el código, es decir, se permite el conocido “Hot Reloading”.

En el caso de querer crear una sección y añadirla, se realizaría de la siguiente forma:

```
Window view = new Window();
Section section = new Section();
// Los datos pueden ser de longitudes irregulares
long[][] data = { ... };
Color chartColors[] = { Color.RED, Color.BLACK };
String chartColumnLabels[] = { "Linea 1",
                               "Linea 2" };
section.createLineChart(labels,
                        data,
                        chartColors,
                        chartColumnLabels,
                        "Ejemplo Lineas");
view.addSection(section,
                DirectionAndPosition.POSITION_TOP,
                "Chart");
```

En el ejemplo anterior se crearía una gráfica de líneas con dos líneas, una de color rojo y otra negra, con los nombres “Linea 1” y “Linea 2”, con el título “Ejemplos Lineas” y con los puntos del array data.

Finalmente, comentar que hay ilimitadas posibilidades de creación y personalización de componentes con los ya configurados y las posibilidades de crear libremente los propios componentes.

2.2. Time profiler

TimeProfiler es un paquete Java que simplifica el cronometraje de la ejecución de funciones. Con TimeProfiler se puede medir fácilmente el tiempo que se tarda en ejecutar una sola función o varias funciones, una o varias veces, permitiendo fácilmente hacer medias de ejecución.

2.2.1. Funcionamiento

El paquete proporciona cuatro métodos: `timeIt`, `batchTimeIt`, `timeIt` para una matriz de funciones y `batchTimeIt` para una matriz de funciones. Basta con introducir la(s) función(es) y el tamaño de lote deseado, y TimeProfiler devolverá la(s) duración(es) de la ejecución. El paquete utiliza las clases Instant y Duration de Java para medir el tiempo con precisión y facilitar la conversión de los datos a las diferentes representaciones.

Adicionalmente, el paquete usa la estructura de datos TimeResult, que actúa como wrapper a un array de Duration con un conjunto de métodos que ofrecen un conjunto de cálculos hiperoptimizados sobre los datos, como podría ser la media, la moda, la suma de sus valores, etc.

2.2.2. Implementación

Antes de explicar brevemente como se ha implementado, se debe comentar el estudio realizado para calcular el posible overhead que ofrece esta librería al usar `Runnable` en vez de una llamada nativa. Para evaluar si tenemos overhead, se calculó el tiempo medio de ejecución de un algoritmo N^2 usando la librería y llamando al método nativamente. El resultado nos dio que la implementación con `Runnable` es, de media, 0.8 % más lento. Debido a este resultado, uno puede usar esta librería sin tener que preocuparse que el tiempo resultante no sea fiel al real o tenga demasiado overhead.

Para el desarrollo de esta librería se ha centrado en el uso de paralelismo mediante programación funcional, de esta manera se aprovecha al máximo la optimización por parte del compilador y aumentamos la velocidad al poder operar paralelamente los streams de datos.

La librería se basa en la utilización del método privado `timeFunction` que dado una función devuelve cuanto tiempo ha tardado en ejecutarse. Todos y cada uno de los métodos disponibles al usuario generan streams de datos y aplican mediante `maps` o derivados la función `timeFunction` y devuelven un `TimeResult`. Esta última estructura de datos es un wrapper de un array de `Duration` con un conjunto de métodos que aprovechan el paralelismo de la programación funcional para obtener datos de interés sobre ese array; como podría ser la media, la moda, la mediana, etc.

2.3. Manual de uso

A continuación se mostrarán un conjunto de casos de uno.

Supongamos que tenemos una función que tiene como argumento un integer x (`fn(int: x)`. Para saber cuanto tarda una ejecución podemos usar `timeIt`. Añadir que, debido a que se usa la interfaz `Runnable`, se debe pasar la función como una función lambda:

```
TimeResult time = TimeProfiler.timeIt(() -> fn(5));
```

Si uno quisiera hacer la media entre cinco ejecuciones, se puede usar `batchTimeIt` en conjunto a la función `mean` de `TimeResult` especificando en que unidad queremos la media; en este caso Nanosegundos:

```
double meanNanos = TimeProfiler.batchTimeIt(() ->
↪  fn(5), 5).mean(Duration::toNanos);
```

Como se ha comentado previamente, también existe la posibilidad de pasar un conjunto de funciones tanto a `timeIt` como a `batchTimeIt`. Supongamos que tenemos otra función que toma como argumento un integer x (`fn2(int: x)`. El código modificado sería el siguiente:

```
Runnable[] functions = new Runnable[] {
    () -> fn(5),
    () -> fn2(5)};
TimeResult times = TimeProfiler.timeIt(functions);
double[] meanNanos = Arrays
    .stream(TimeProfiler.batchTimeIt(functions, 5))
    .mapToDouble((x) -> x.mean(Duration::toNanos))
    .toArray();
```

Comentar finalmente, que esta es una posible implementación, pero se puede tratar desde un diseño más imperativo. Si uno quiere explorar la documentación completa, puede acceder a ella a través de este link.

2.4. Conceptos MVC

El Modelo Vista Controlador (MVC) es un patrón arquitectónico de software que divide una aplicación en tres elementos interconectados, separando la representación interna de la información, como se muestra al usuario y donde se hacen cómputos con esa información respectivamente.

Para esta práctica, se ha decidido modificar parcialmente este patrón al implementar la comunicación entre los módulos mediante un sistema de peticiones. Por ende, encontramos 4 módulos en nuestro MVC: Modelo, Vista, Controlador y Hub. Este último se encargará de gestionar toda la comunicación.

Esta decisión de modificación del MVC ha sido respaldada por la facilidad que obtenemos de escalar el patrón, permitiendo crear tantos módulos como queramos. Esto permitiría que, si en un futuro se quisiera añadir otro controlador en otro lenguaje de programación o encapsular controladores por funcionalidad, el único cambio a efectuar ocurriría en el hub.

A continuación se explicarán cada uno de los módulos.

2.4.1. Hub

El flujo de datos empleado en este patrón de diseño ha sido inspirado en como se comunican los diferentes elementos en un servidor. Es decir, cualquier módulo del MVC deberá realizar una petición al servidor (Hub) y este se encargará de solventar y dar servicio a dicha petición. Se puede ver con mayor claridad como funciona y su implementación en la siguiente figura:

El Servidor/Hub se encarga de administrar los diferentes tipos de peticiones que llegan de los diferentes componentes. En la figura 1 se observan los diferentes tipos de peticiones posibles que se pueden realizar, en concreto: GET, POST y PUT. En un entorno web, GET sirve para obtener datos, POST para enviar y PUT para actualizar. En el programa los diferentes tipos de peticiones se han traducido en:

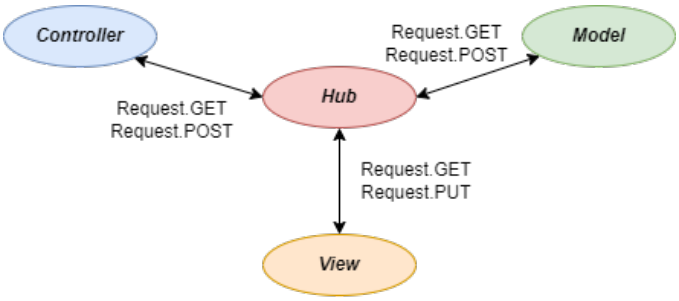

Figura 1. Patrón MVC usado en la práctica.

GET Realiza un retorno del dato solicitado.
POST Añade un el dato enviado.
PUT Realiza un set del dato enviado.

Para permitir esta implementación, se ha creado la interfaz `Notify` con el método `notifyRequest` el cual deberá estar implementado en todos los módulos del MVC, ya que será el responsable de gestionar las peticiones y asegurar que cada uno de ellos puede realizar sus tareas designadas de manera eficiente y precisa.

2.4.2. Modelo

El modelo es la representación de los datos que maneja el software. Contiene los mecanismos y la lógica necesaria para acceder a la información y para actualizar el estado del modelo.

Esta clase contiene diferentes tipos de estructuras de datos específicas a esta práctica, incluyendo, pero no solamente, la iteración actual del programa, el tamaño del lote, los tiempos de ejecución de todos los algoritmos implementados y el “timeout”.

Adicionalmente de los datos mencionados previamente, provee unos métodos o funciones auxiliares que facilitan la modificación o adquisición de los datos:

`resetData` y `resetIterations` permiten poner los valores por defecto de los principales datos del modelo.

`nextIteration` actualiza el tamaño del vector de datos para la siguiente iteración y aumenta el contador de iteraciones.

`collectData` obtiene los resultados de la última iteración presente en el controlador de manera "inteligente", al tratar los datos de diferente manera dependiendo del estado del controlador.

`getData` devuelve todos los tiempos de ejecución hasta ese instante convertidos mediante la función de representación seleccionada en la vista.

Finalmente, al ser un módulo de nuestro MVC, implementa la interfaz `Notify` y su método `notifyRequest` que le permite recibir notificaciones de los otros módulos del MVC.

2.4.3. Vista

La vista contiene los componentes para representar la interfaz del usuario (IU) del programa y las herramientas con las cuales el usuario puede interactuar con los datos de la aplicación. Adicionalmente, la vista se encarga de recibir e interpretar adecuadamente los datos obtenidos del modelo.

`loadContent` es el encargado de cargar el contenido en la ventana. Para esta práctica, carga los siguientes elementos semánticos:

`header` del programa mediante `createButtons`, generando el conjunto de botones que permiten al usuario del programa lanzar cada uno de los algoritmos e incluso pausar su ejecución en cualquier momento. Para ser más específicos, los botones generados son: *Escalar*, *Moda n Log n*, *Moda n*, *Todos* y *Reanudar/Pausar*.

`main` del programa mediante `updateChart`, cargando el gráfico con respecto a las variables por defecto encontradas en el modelo.

`footer` del programa que ocupará la zona inferior de la interfaz, conteniendo, las barras de progreso para alcanzar el tiempo límite de cada algoritmo, el tamaño del lote, la representación temporal, además de la iteración actual y la ponderación por iteración.

Adicionalmente, la clase contiene un conjunto de métodos o funciones que facilitan la generación de los elementos en la interfaz de usuario. A continuación se explicarán brevemente los más importantes:

`createProgressBarToTimeOut` crea una barra de progreso que, para esta práctica, estará asociada al progreso con respecto al timeout especificado en el modelo de un algoritmo. Siendo la barra del algoritmo Escalar de color rojo, el algoritmo ModeNLogN de color azul y el algoritmo Moda N de color verde, siendo así consistentes con la leyenda de la gráfica.

`createButtons` crea todos los botones que aparecen en el header de la interfaz, es decir, los botones: *Escalar*, *Moda n Log n*, *Moda n*, *Todos* y *Pausar/Reanudar*. Adicionalmente, por cada botón se le asigna un "listener" que generará la petición correspondiente a la etiqueta del botón.

`footer` crea un menú en la parte inferior de la interfaz con las siguientes opciones:

- Una opción para seleccionar la representación del tiempo permitiendo seleccionar entre Nanosegundos, Milisegundos, Segundos, Minutos, Horas o Días.
- Una opción para seleccionar el tamaño del lote. Cuando se cambia el tamaño del lote se reinician todos los datos.
- Una opción para seleccionar el número de iteraciones. Cuando se cambia el número de iteraciones se reinician todos los datos.

2.4.4. Controlador

El controlador en un MVC es el responsable de recibir y procesar la entrada del usuario y actualizar el modelo. En esta práctica, dicha responsabilidad es la de calcular el tiempo de ejecución de los siguientes algoritmos:

La moda de un vector numérico en $O(n)$
La moda de un vector numérico en $O(nlogn)$
El producto escalar entre dos vectores numéricos en $O(n^2)$

Para la implementación de estos algoritmos se ha decidido seguir una convención de programación declarativa, aunque se ha implementado la versión imperativa en el caso de que el desarrollador los prefiera usar. Adicionalmente, el propio controlador produce los vectores de datos usados como argumentos en los algoritmos mediante un generador aleatorio, haciendo su ejecución lo más transparente posible.

La ejecución del cálculo se realiza en un thread virtual, obteniendo una mejora en el tiempo de respuesta de la aplicación; permitiendo que otros eventos puedan tomar el thread principal, aprovechando los cores de la CPU y reduciendo el bloqueo de la aplicación; permitiendo que el thread principal sea menos probable de quedarse bloqueado debido a cálculos de larga duración.

Una vez acabado los cálculos, crea una petición al hub para que el modelo pueda obtener los resultados. Esto es posible debido a que, al ser un módulo del MVC modificado, implementa la interfaz `Notify` y el método `notifyRequest` que le permite comunicarse con los otros módulos.

2.5. Manual de usuario

Para el correcto uso de la aplicación, el conjunto de acciones que se pueden realizar en dicho programa serán definidas a continuación.

Cuando se ejecute el programa por primera vez en la pantalla se debe mostrar la siguiente interfaz de usuario:

2.5.1. Header

En el "header" de la aplicación podemos encontrar un conjunto de botones que nos permiten controlar su ejecución.

Los cuatro primeros botones (*Escalar*, *Moda NLogN*, *Moda N*, *Todos*) ejecutan y muestra el algoritmo (o algoritmos) respectivos. Finalmente, el último botón nos permite pausar o reanudar la ejecución actual.

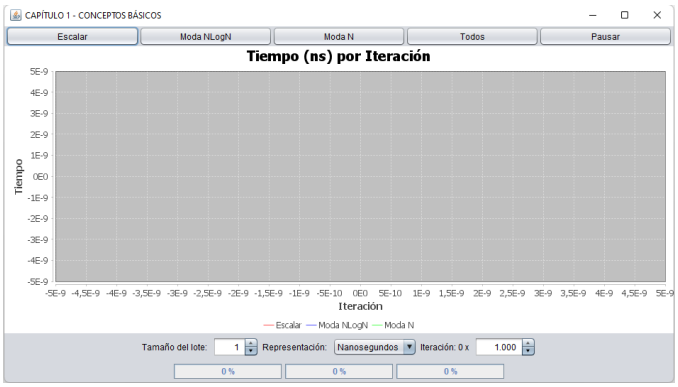

Figura 2. Interfaz de usuario

2.5.2. Main

El principal bloque de la aplicación se encuentra el gráfico donde se representarán los tiempos de ejecución de los posibles algoritmos a ejecutar. Adicionalmente, se puede ampliar la imagen, guardarla y modificar los axis entre otros.

Añadir que en la parte inferior del gráfico siempre se podrá consultar la leyenda, que indica que color del gráfico se relaciona a que algoritmo.

2.5.3. Footer

En el "footer" de la aplicación se pueden encontrar un conjunto de "gadgets" que nos permiten controlar la ejecución de los algoritmos y su visualización, además de proporcionar algo de "feedback". A continuación se explicará cada uno de ellos de izquierda a derecha:

Primeramente, la sección definida por la etiqueta "Tamaño de lote" permite modificar la cantidad de llamadas a los algoritmos se harán por iteración. Si el número indicado es mayor a 1, se representará la media entre todos los resultados.

Seguidamente, la sección definida por la etiqueta "Representación" permite elegir la representación temporal de los datos. Por defecto, la representación de los datos será en Nanosegundos (ns). Cuando este cambie, se actualizará tanto el nombre del gráfico como la representación de los datos, cambiando el eje x si fuese necesario.

A su derecha, la sección definida por la etiqueta “Iteración” visualiza el número de iteración actual de la ejecución y su actual “step”. Este último aspecto permite ajustar por cuanto se quiere ponderar el número de iteración para los algoritmos. Así pues, si se tiene `5 X 500`, los algoritmos verán que se encuentran en la iteración 2500.

Finalmente, en la parte inferior se encuentran tres barras de progreso que indican cual es el porcentaje de progreso de la ejecución actual con respecto un `timeout` definido en el código fuente. Cada una de ellas tiene asociado un color que permite identificar a que algoritmo hace referencia al ser el mismo que en la leyenda. Una vez ha llegado al 100 % la función dejará de evaluarse y se podrá tomar como finalizada.

A continuación se muestra un ejemplo de ejecución del programa con todos los algoritmos, tamaño de lote = 1, representación en nanosegundos y 1000 de ponderación por iteración:

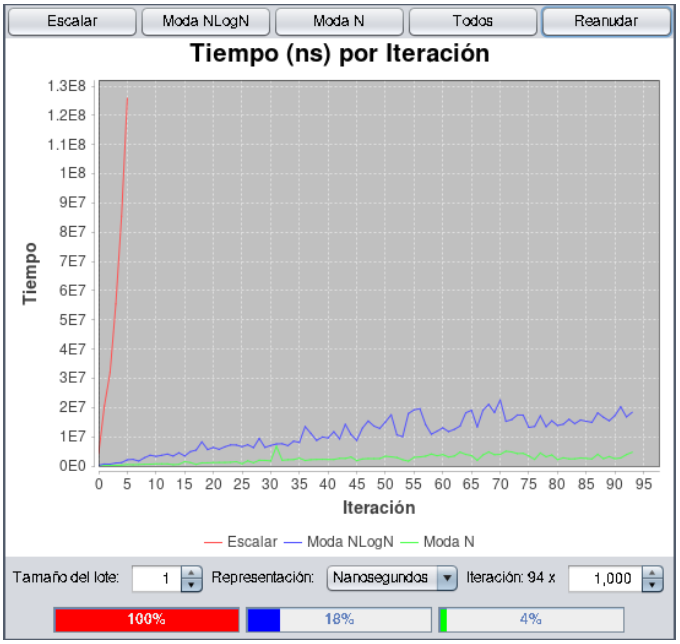

Figura 3. Ejemplo de ejecución

3. Conclusión

Teniendo en cuenta el desarrollo expuesto, la aplicación permite generar un punto de referencia de un conjunto de algoritmos de diferente complejidad asintótica mediante una representación gráfica e interactiva de la media de los tiempos de ejecución a través de unos parámetros de entrada definidos por el usuario. Adicionalmente, se ha implementado mediante una versión modificada del patrón de diseño de software **M**odelo **V**ista **C**ontrolador (MVC) añadiendo un cuatro módulos que permite la comunicación entre los diferentes elementos de este mediante peticiones. Esta modificación ha sido fuertemente inspirada en el diseño de un servidor, lo que permite gran flexibilidad y escalabilidad de desarrollo, al poder interceptar y gestionar las peticiones a gusto del desarrollador.

Además, se ha hecho énfasis en el uso de las librerías creadas por los miembros del grupo para facilitar el trabajo y la reutilización de código en futuras prácticas. Reiterar que el principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo “overhead” al rendimiento del programa.

Concluir que, con respecto al ámbito académico, este proyecto nos ha permitido consolidar el concepto de patrón de diseño MVC gracias a un primer proceso de investigación y discusión del diseño a implementar entre los integrantes del grupo.

4. Distribución del trabajo realizado

El trabajo realizado por cada miembro de la práctica ha sido el siguiente:

- Alejandro Rodríguez:
  - Documentación: explicación/implementación View y reconocimientos.
  - Práctica: sección header (implementación de los botones), sección footer (implementación de las opciones y sus ac-

ciones), parte del View y parte del BetterSwing.

- Rubén Palmer:
  - Documentación
  - Desarrollador de la librería `TimeProfiler`
  - Principal desarrollador del "backend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "frontend"
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto
  - Desarrollador de la plantilla de la documentación
- Sergi Mayol:
  - Documentación
  - Desarrollador de la librería `BetterSwing`
  - Principal desarrollador del "frontend" de la aplicación
  - Desarrollador general de la aplicación
  - Principal desarrollador del "backend" de la aplicación
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto

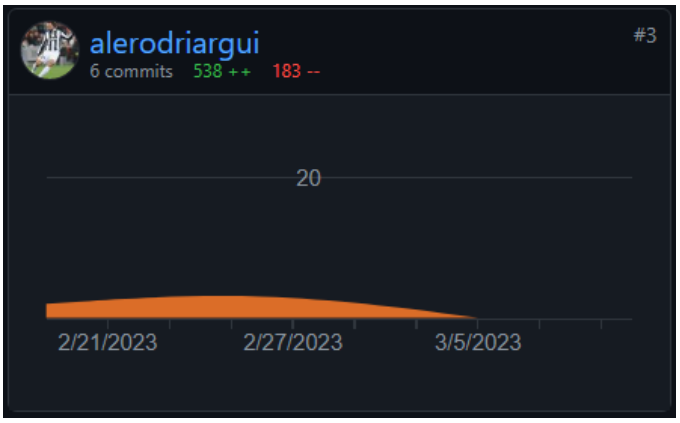

Figura 4. Contribución en el proyecto de Alejandro Rodríguez

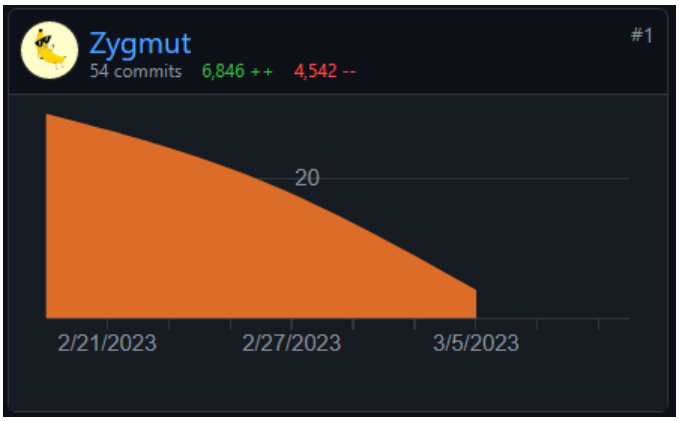

Figura 5. Contribución en el proyecto de Rubén Palmer

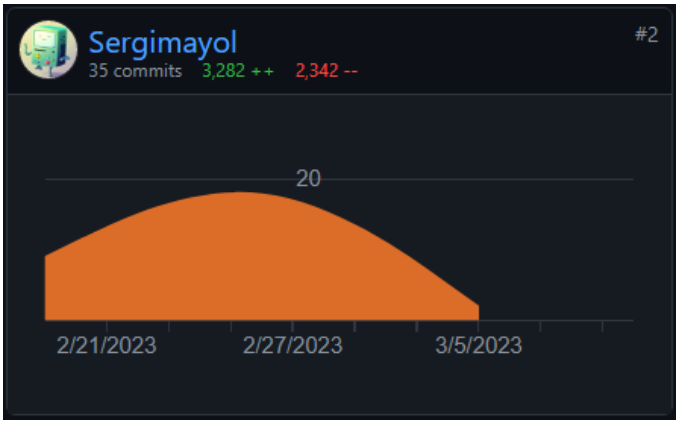

Figura 6. Contribución en el proyecto de Sergi Mayol

RECONOCIMIENTOS

- Se agradece la colaboración del Dr. Miquel Mascaró Portells en la resolución de dudas y supervisión del proyecto.
- Agradecer a los desarrolladores y contribuidores de la librería JFreeChart por permitir el uso de su librería de manera gratuita.