Pareja de Puntos Más Cercana Mediante MVC

Alejandro Rodríguez, Rubén Palmer y Sergi Mayol

Resumen— Esta aplicación presenta una interfaz gráfica de usuario que le permite interactivamente encontrar la solución a la "Pareja de puntos más cercana", para un conjunto de puntos dado y con una distribución determinada. A partir de la IU, el usuario podrá seleccionar el tipo de distribución (Gaussiana, Normal, Exponencial, ...), el número de puntos a generar, la semilla para la generación de los puntos y el tipo de algoritmo a aplicar. En esta práctica, se han implementado dos algoritmos para encontrar la solución. Uno de ellos, tiene una complejidad de $O(n^2)$ y el otro algoritmo tiene una complejidad de $O(n * logn)$.

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 y un ejemplo de esta.

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 [1] es una librería 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.json");
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.

3. 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 y como han sido adaptados y empleados en la práctica.

3.1. 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:

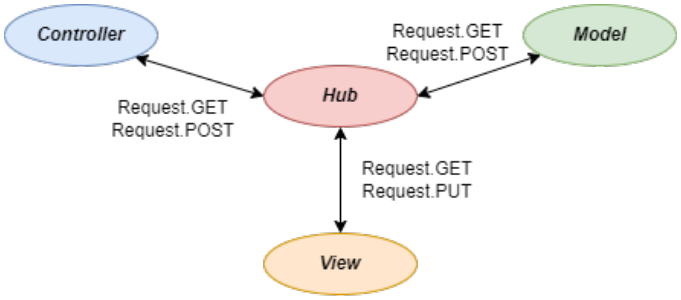

Figura 1. Patrón MVC usado en la práctica.

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:

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.

3.2. 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:

hub Permite la comunicación con el controlador y la vista.
seed Representa el número de la semilla.
pointAmount Representa el número de puntos.
data Array de puntos para guardar los datos.
solutionsForMaxNN Array de soluciones para el algoritmo N*N buscando la distancia máxima.
solutionsForMinNN Array de soluciones para el algoritmo N*N buscando la distancia mínima.
solutionsForMinNLogN Array de soluciones para el algoritmo N*logN buscando la distancia mínima.
nSolutions Número de soluciones a representar

useNLogNAlgorithm Booleano para seleccionar el tipo de algoritmo a aplicar.

useMaxOnAuto Booleano para iterar el algoritmo en caso de que sea true.

Lambda Almacena el valor de lambda.

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.

3.2.1. Estructuras de datos

Para esta práctica se han generado tres estructuras de datos de lectura para facilitar la encapsulación de información. Para implementarlas se ha aprovechado de la estructura "record" de Java. A continuación se mencionan y explican las estructuras:

Point Representa un punto en la nube de datos.

PairPoint Permite guardar una pareja de "Point's".

Solution Permite guardar un "PairPoint" junto a su distancia y su tiempo para encontrarlos como solución, es decir, esta estructura, básicamente, contiene una solución del algoritmo.

3.3. 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. Cabe mencionar, que al igual que el resto de componentes del MVC, la clase View implementa la interfaz "Notify", la cual permite la comunicación con el resto de elementos.

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

`menu`, función que crea y configura el menu de utilidades de la ventana principal. En esta, se incluyen las siguientes opciones: Abrir ventanas de estadísticas, borrado y reseteo de datos, selección de algoritmos, y selección del modo auto.

`body`, función que crea, configura y actualiza la gráfica de la ventana principal. En concreto, se trata de un gráfico de dispersión que contiene el conjunto de puntos de la práctica.

`sidebar`, función que crea y configura la barra de opciones de la derecha de la interfaz principal. Esta permite al usuario interactuar y configurar el entorno de ejecución de los algoritmos.

`footer`, función que crea y configura las opciones de inicio de los algoritmos en la interfaz. Concretamente, este elemento contiene el modo del algoritmo a ejecutar (distancia máxima o distancia mínima) y la opción auto. La opción auto ejecuta los algoritmos seleccionados hasta que no se pueda encontrar ninguna solución más.

Adicionalmente, a la vista principal, esta permite desplegar dos ventanas más. Una ventana muestra a tiempo real, el uso y consumo de la memoria de la Java virtual machine (JVM) y la otra ventana muestra las estadísticas de la ejecución de los algoritmos además de su comparación. A continuación, se muestran dos imagenes (2 y 3) de como se verían las ventanas al iniciarlas junto a una breve explicación de la misma.

3.3.1. Estadísticas JVM

Este apartado de la vista principal, es el encargado de enseñar a tiempo real las estadísticas de la máquina virutal de java. Concretamente, se actualiza cada 0.5s a apartir de los datos obtenidos de la clase de java "Runtime" y muestra la memoria libre, la memoria total y el uso de esta en una gráfica de lineas, donde el eje x es instante en el tiempo que se han obtenido los datos y el eje y su valor. Todos los datos de la memoría obtenidos están en MB.

3.3.2. Estadísticas Algoritmos

Este apartado de la vista principal, es el encargado de enseñar las estadísticas de la ejecución de los algoritmos, tanto N^2 como $NlogN$. Estas estadísticas de cada algoritmo incluyen la media de las distancias, la máxima y mínima distancia y el tiempo medio de ejecución,

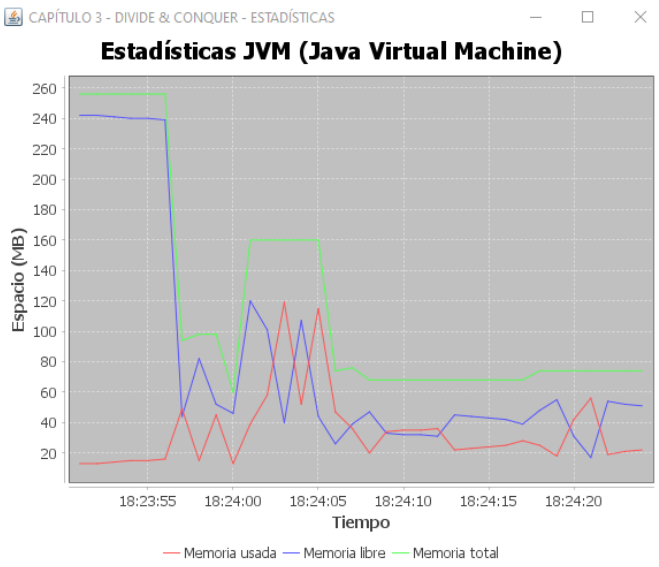

Figura 2. Interfaz estadísticas JVM

además de una compración directa entre los algoritmos.

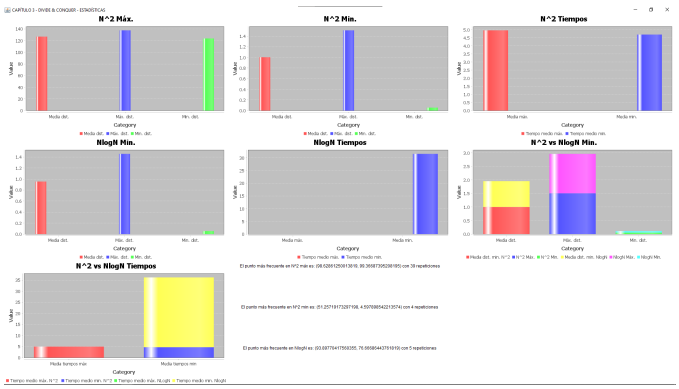

Figura 3. Interfaz estadísticas algortimos

Como se ha podido ver anteriormente en la imagen (3), los datos de las estadísticas están representados con diagramas de barras, donde el eje x es la categoría a la que pertenecen y el eje y el valor correspondiente.

3.4. 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 generar un conjunto de puntos de una distribución en concreto, la de encontrar *n* parejas que maximicen o minimicen su distancia y generar estadísticas basadas en los datos del modelo.

3.4.1. Diseño

El controlador se separa en tres grandes secciones como ya se ha mencionado previamente.

Para la generación de datos, se definen un conjunto de funciones que, mediante el objeto interno que permite generar datos aleatorios, retorna el siguiente valor de la distribución que caiga dentro de unas "boundaries" específicas. De esta manera, nos aseguramos que todos los datos son fácilmente visibles a la hora de visualizaros; Facilitando así las futuras visualizaciones de las soluciones. Actualmente, el software permite la visualización de la distribución Uniforme, Gausiana, Exponencial desplazada y Bernoulli. Adicionalmente, esta generación de datos utiliza una semilla, por lo que el usuario fácilmente puede replicar resultados.

Para la generación de estadísticas, se recogen los datos del modelo y se efectúan medias, máximos y mínimos para visualizarlos claramente en la ventada de estadísticas (3). De esta manera, ofrecemos al usuario un feedback sobre la ejecución del algoritmo que puede ser usada para contrastar los resultados en un estudio formal.

Finalmente, el cálculo de distancias mínimas y máximas se efectúan bajo los datos presentes en el modelo aplicando el respectivo algoritmo dependiendo de la petición obtenida del Hub. A continuación se entrará en más detalle la implementación de los algoritmos y su proceso de diseño.

3.4.2. Algoritmos

Para esta práctica, se han implementado la búsqueda de distancias mínimas y máximas con un coste asintótico $O(n^2)$ y la búsqueda de distancias mínimas en $O(nlogn)$.

Para las implementaciones $O(n^2)$, los algoritmos iteran por todos puntos de datos y calculan todas las posibles permutaciones entre ellos, guardando dinámicamente las

mejores soluciones en el respectivo "array". Asintóticamente hablando, debido a que el software está generalizado para m soluciones, presenta un coste de $O(n^2mlogm)$ ya que se efectúa una ordenación de los datos para asegurar que siempre se guardan las mejores soluciones. Adicionalmente, debido a que se ha considerado que el número de soluciones a encontrar no sería especialmente grande, se ha decidido no optimizar el algoritmo a $O(n^2m)$ para mantener una mejor legibilidad y calidad del código, ya que se podría implementar una modificación del las actuales soluciones en coste lineal y guardar localmente la mejor/peor solución para poder compararlo con la solución actual.

Para la implementación $O(nlogn)$ de las distancias mínimas, como el título del documento indica, se ha decidido usar "Divide & Conquer" para ir dividiendo el espacio de puntos logarítmicamente. Este se divide en dos partes, un inicial estado donde se preparan los datos y una segunda, y más computacionalmente pesada, donde se computan todas las distancias. Inicialmente, se recogen los datos del modelo y se ordenan por su componente x, de esta manera facilitamos la separación de los datos en los diferentes sectores en el siguiente apartado. El segundo apartado es un algoritmo recursivo cuyos parámetros permiten identificar el "slice" de datos que se está tratando actualmente y los datos a tomar. La condición de salida de este algoritmo se cumple cuando el "slice" de datos solo presenta dos puntos. En tal caso, se genera un "array" de soluciones de m posiciones donde en la primera posición se encuentra la única solución y las restantes presentan distancias imposibles. En este caso, debido a que se están trabajando con "arrays", el tiempo de composición de las soluciones es de $O(nlogn)$ al tener que obtener ambas soluciones, ordenarlas y producir un "slice" con las menores distancias. Finalmente, se debe revisar los puntos intermedios entre los dos sectores, ya que podrían existir mejores soluciones. Para ello, se toma la menor distancia y se crea un "slice" de datos desde el punto intermedio con $2 * menor\_distancia$ como el diámetro. De esta manera, aseguramos que solo se tomen aquellos puntos que puedan ofrecer una mejor solución. Finalmente, se generan todas las posibles permutaciones entre ellos mediante un algoritmo $O(n^2)$ similar al expuesto previamente. Como se acaba de comentar, uno podría pensar que, debido a que por cada ejecución del algoritmo recursivo se efectúa un "snippet" de código con complejidad asintótica $O(n^2)$ el algoritmo presentaría dicha complejidad. Esto es cierto en distribuciones cuya densidad de puntos en los valores centrales es significativamente mayor, como podría ser la Gausiana. Sin embargo, para distribuciones equiprobables, como la Uniforme, sabemos que la cantidad de puntos de cada conjunto que están dentro de un sector tiende a la raíz cuadrada de los puntos en total. Debido a esto, se podría considerar que este algoritmo es $O(nlognmlogm)$ en distribuciones equiprobables estadísticamente hablando, ya que para otras distribuciones puede presentar una ejecución más cercana a $O(n^2)$. Por ejemplo, la distribución de Bernoulli presenta una excepcional densidad de datos en un reducido espacio. Según esta teoría, la ejecución $O(nlogn)$ debería acercarse a la de $O(n^2)$. Suponiendo que está utilizando este software con cinco mil puntos y cinco parejas, se obtiene que ambas ejecuciones tardan, en nuestra máquina de testing, aproximadamente 56 milisegundos. Esto afirma nuestra teoría.

Finalmente, una vez obtenidas las mejores soluciones, se envía una petición al Hub para que actualice los datos en el modelo. 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.

3.4.3. Optimizaciones

Para optimizar el algoritmo se ha hecho un estudio para saber a partir de que punto un algoritmo con coste asintótico $O(n^2)$ es más eficiente que un $O(nlogn)$.

Según el estudio realizado, hemos obtenido que, ya que la distancia media entre dos pares de puntos en una distribución uniforme es de siete, se obtiene una mayor eficiencia realizando el algoritmo en $O(n^2)$ que el $O(nlogn)$.

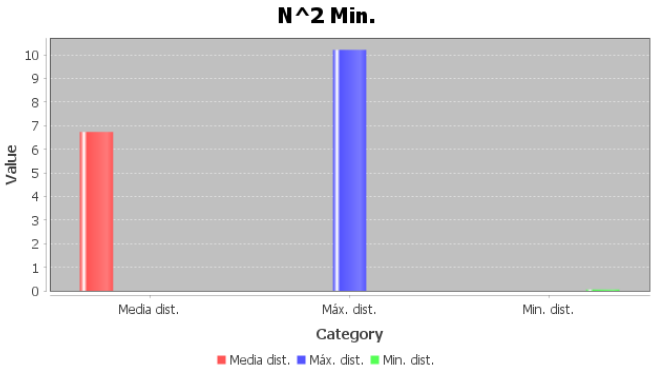

Figura 4. Estudio sobre la distancia media de una distribución uniforme

La razón por la que se limita el número de puntos verificados es para reducir el tiempo de ejecución del algoritmo. Al limitar el número de puntos verificados, podemos reducir el tiempo de ejecución del algoritmo y aun así encontrar el par (o pares) de puntos más cercanos en la matriz de la franja con una precisión razonable.

Por ende, se modifica el caso base para que, una vez la franja de la sección es menor a siete, se lance un algoritmo de coste asintótico $O(n^2)$ para encontrar más rápidamente la solución.

4. 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. También, la distribución de las secciones de la interfaz junto a su explicación y funcionalidad.

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

4.1. Menu

El menu de la aplicación se trata de la barra que se situa debajo del marco superior de la

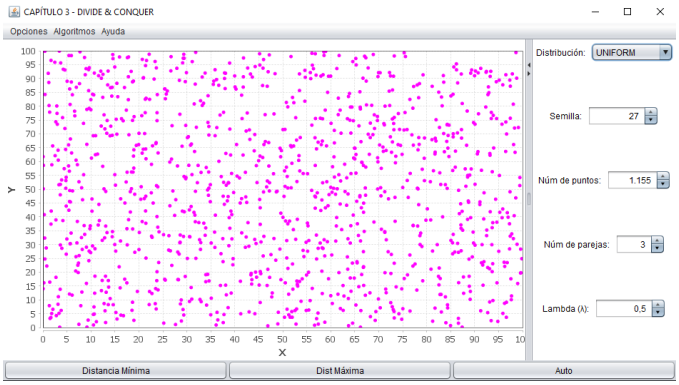

Figura 5. Interfaz de usuario

ventana. En esta se puede encontrar un conjunto de opciones para que el usuario pueda iteractuar con la aplicación, modificar y analizar el comportamiento de esta. En concreto, se encuentran las opciones de "Opciones", "Algoritmos" y "Ayuda".

En la primera se situán todas las posibles acciones generales relacionadas con la aplicación, ya sea como borrar soluciones, borrar y/o resetear datos e inciar las ventanas de estadísticas explicadas anteriormente en los apartados 3.3.1 y 3.3.2.

En la segunda se sitúan todas las posibles acciones relacionadas directamente con la ejecución de los algoritmos, especificamente, que tipo de algoritmo ejecutar (n^2 y $nlogn$) y el modo de la opción auto, explicado en el apartado 4.4, (n^2, $nlogn$ y "BenchMark").

Y finalmente, en la última opción se encuentra un menu desplegable con un manual de usuario con la explicación del funcionamiento de la aplicación.

4.2. Main

El "Main" es el bloque principal de la vista, donde se representará gráficamente, la distribución, el número de puntos seleccionado, la semilla y la solución a los algoritmos, además de poder interactuar con el gráfico.

4.3. Sidebar

El "Sidebar" contiene un conjunto de opciones para modificar los datos y el modo de ejecución de la aplicación. A continuación, se explicará cada una de estas opciones y su función.

En primer lugar, se encuentra la opción para seleccionar la "Distribución". En esta, se puede escoger el tipo de distribución que queremos aplicar a la generación de puntos. En esta práctica, se han implementado las siguientes distribuciones:

- Uniform
- Gaussian
- Exponential
- Bernoulli

En segundo lugar, se halla la opción para elegir la "Semilla". En esta opción, se tiene la opción cambiar el valor de la semilla para que la generación de punto se pueda replicar. Es decir, los puntos se generan aleatoriamente a partir de esta semilla.

En tercer lugar, existe la opción para seleccionar la cantidad de puntos que se desean representar, "Núm. puntos". Se podrán generar los puntos que se deseen introduciendo la cantidad en esta opción y pulsando enter.

En cuarto lugar, se encuentra la opción para elegir el número de parejas, "Num. Parejas". Esta opción permite seleccionar el número de parejas de puntos que se desea que el algoritmo encuentre. Por ejemplo, si seleccionamos 3 parejas, se calcularán 3 parejas de soluciones no repetidas.

Por último, la etiqueta "Lambda", esta solo será útil para el tipo de distribución que se haya aplicado a los puntos. La única distribución implementada la cual lambda (λ) puede contribuir a la generación de datos es la distribución Exponencial. Las demás distribuciones no se verán afectadas por lambda.

4.4. Footer

En la sección "Footer", se hallan tres botones para seleccionar el tipo de algoritmo a ejecutar. En primer lugar, empezando por la izquierda de la interfaz, tenemos el botón Distancia mínima. Una vez el botón ha sido pulsado, se calculará la distancia mínima entre dos puntos dependiendo de los parámetros que el usuario haya introducido.

En segundo lugar, tenemos el botón "Distancia máxima". Una vez el botón ha sido pulsado, se calculará la distancia máxima entre dos puntos dependiendo de los parámetros que el usuario haya introducido.

Por último, el botón "Auto" calculará de manera repetitiva la distancia mínima o máxima entre dos puntos, a partir de los parámetros introducidos por el usuario. Se ejecutará el algoritmo seleccionado de manera iterativa hasta que se acabe el algoritmo o se pulse el botón de "Stop", el cual aparece después de haber pulsado el botón "Auto". Adicionalmente, esta opción permite realizar al usuario de la aplicación un "Benchmark" de los algoritmos de forma automática.

4.5. Ejemplo ejecución

Al iniciar la aplicación se mostrará una interfaz como la expuesta en la imagen 5. Para poder iniciar la ejecución del algoritmo, es necesario que se pulse el botón del algoritmo que se desee ejecutar, encontrar parejas a una máxima distancia o a una mínima distancia, además de seleccionar la complejidad del algoritmo que por defecto se ejecuta el n^2. Y, opcionalmente, modificar los datos como se ha mencionado anteriormente en la sección 4.3.

A continuación, la siguiente imagen, muestra un ejemplo de la interfaz en mitad de la ejecución:

A partir de aqui, el usuario puede reiniciar la ejecución para ejecutar nuevamente el algoritmo, borrar las soluciones obtenidas, ejecutar otros algoritmos y ver estadísticas de la ejecución, ver apartados 3.3.1 y 3.3.2.

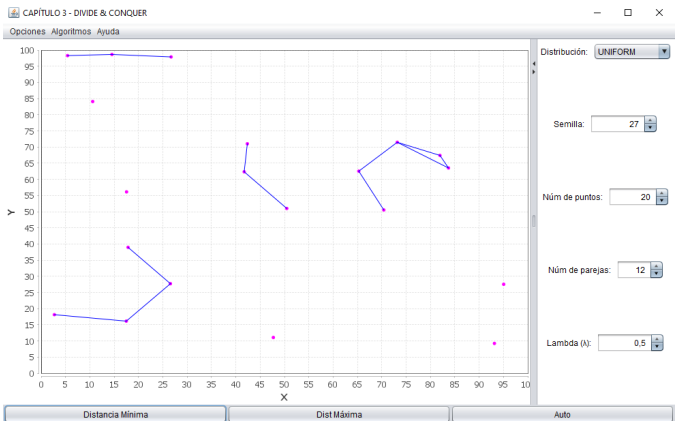

Figura 6. Interfaz de usuario

5. 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.

6. Distribución del trabajo realizado

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

- Alejandro Rodríguez:
  - Desarrollador de funcionalidades extra de la IU.
- Rubén Palmer:
  - Documentación
  - 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 `Better Swing`
  - Principal desarrollador del "frontend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "backend" de la aplicación
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto

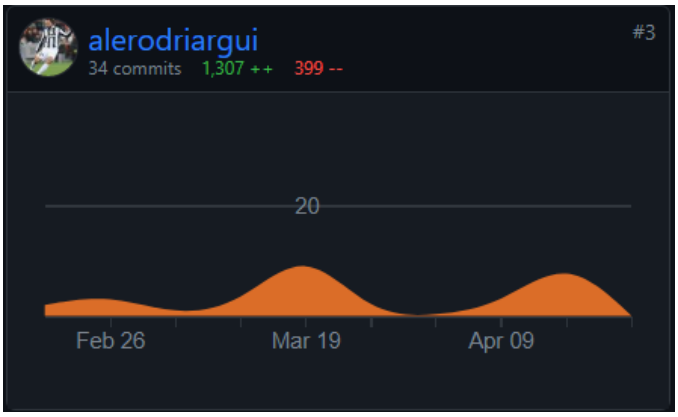

Figura 7. Contribución en el proyecto de Alejandro Rodríguez

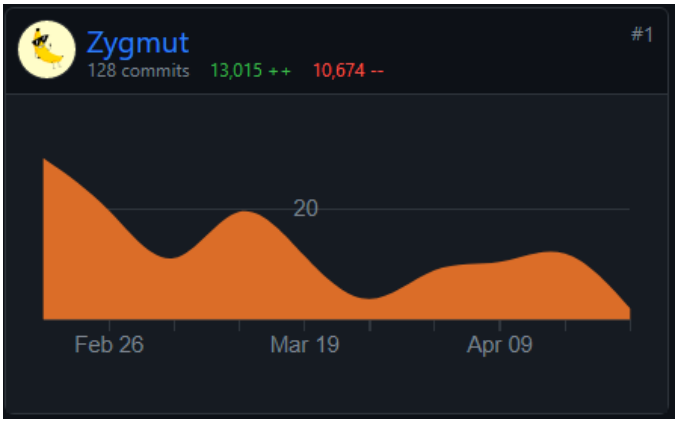

Figura 8. Contribución en el proyecto de Rubén Palmer

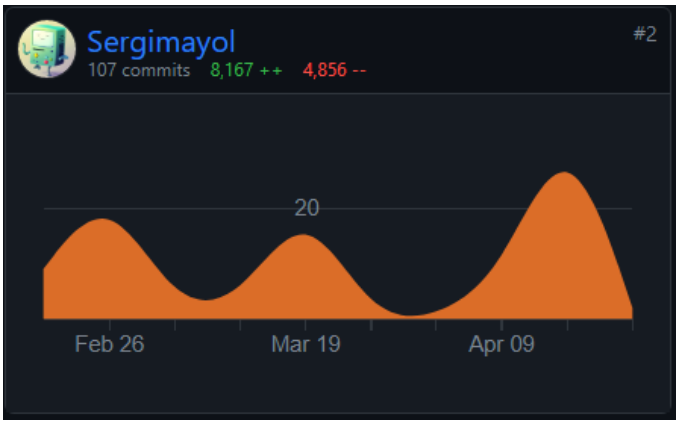

Figura 9. Contribución en el proyecto de Sergi Mayol

Reconocimientos

- Se agradece la colaboración del Dr. Miquel Mascaró Portells en la resolución de dudas y supervisión del proyecto.

Referencias

[1] Better Swing, *An easy way to develop java GUI apps*, V0.0.3. Ver documentación completa aquí.
[2] Graphical engine, *How to develop a graphical engine from scratch*. Ver enlace aquí.
[3] What is the exponential distribution, what are its formulas and how does it work. Ver documentación aquí.
[4] What is the Bernoulli distribution, what are its formulas and how does it work. Ver documentación aquí.
[5] What is the exponential distribution, what are its formulas and how does it work. Ver documentación aquí.
[6] Closest pair of points problem. Ver más en aquí.