VNIVERSITAT DE VALÈNCIA

MGVIZ: DESARROLLO E INTEGRACIÓN DE MÉTODOS DE ANÁLISIS Y VISUALIZACIÓN DE DATOS GENÓMICOS EN BIOMEDICINA

Tesis doctoral de

Jose Miguel Juanes Tébar

Dirigida por

Dr. Felipe Javier Chaves Martínez
Dr. Vicente Arnau Llombart
Dr. Pablo Marín García

Programa de Doctorado en Tecnologías de la Información, Comunicaciones y Computación

Noviembre 2020

Este estudio ha sido financiado por el Instituto de Salud Carlos III a través del proyecto “IFI15 / 00138”, cofinanciado por el Fondo Social Europeo. “*El FSE invierte en tu futuro*”.

AGRADECIMIENTOS

En estas líneas me gustaría agradecer a todas aquellas personas que me han ayudado a continuar.

Agradecer enormemente a mis directores, Javier Chaves, Vicente Arnau y Pablo Marín por toda la ayuda que desde el primer momento me han ofrecido. En especial me gustaría agradecer a Pablo por haber ejercido no solo de director, sino de mentor, compañero y amigo. Gracias por haberme ayudado a mejorar no solo como investigador sino como persona. Y sobretodo gracias por todo el tiempo que me has dedicado durante estos años de tesis.

Agradecer también a Ana Bárbara, que aunque no hayas sido directora oficialmente siempre te he considerado como tal. Gracias por toda la ayuda que me has prestado desde el primer momento, por tus mensajes de ánimo y por tu paciencia por todas las horas que he acaparado a Pablo durante estos años.

Agradecer a mis compañeros de la *UGD*, *Seqplexing*, *ETSE*, *i2sysbio* y *Kanteron*. También agradecer enormemente a mis compañeros del MGviz: Anastasiya, Dani, Jaime, Jordi, María, Miguel, Pablo y Rodiel. En especial me gustaría agradecer a David y a Pedro, por haber sido

no solo compañeros sino también amigos. Gracias a todos vosotros por el apoyo y la ayuda que me habéis brindado para que esto llegase a ser una realidad.

Agradecer a Toni y a Eli por su ayuda, su apoyo y los valiosísimos consejos que me han dado, tanto durante mis estancias en Cambridge como durante las noches por videollamada que hemos estado en estos últimos meses.

Agradecer a mis compañeros y amigos de matemáticas: Fer, Marta, Diego, Diana y Silvia. En especial agradecer a Pablo y a Joaquín, por que siempre habéis estado ahí cuando más os he necesitado.

Y por último, y no por ello los menos importantes, a toda mi familia. En especial a mis padres Porfidia y Jose, por todo el sacrificio que habéis tenido que hacer para que yo haya llegado hasta este punto, y a mi hermano Sergio. Esto nunca habría salido adelante sin vuestro apoyo diario.

A todos vosotros, gracias de corazón.

RESUMEN

En estos últimos años se ha producido un crecimiento exponencial del área de la visualización de datos, potenciado en gran medida gracias al auge de la ciencia de datos y el *Big Data*. En los análisis bioinformáticos, la visualización de los datos representa un componente muy importante, pues permite descubrir patrones que ayuden a generar hipótesis que se puedan testear con nuevos datos.

La creación de software, destinado a la realización de visualizaciones interactivas para exploración de datos, no solo requiere una alta capacitación técnica, sino también un conocimiento de las necesidades de los investigadores y la comprensión de la naturaleza de los datos; tres cualidades que no siempre se encuentran juntas en los equipos que desarrollan aplicaciones bioinformáticas. Aunque existen aplicaciones específicas para la creación de este tipo de visualizaciones, el problema es que están basadas en bibliotecas de alto nivel, lo que las limita a visualizar sólo los tipos de gráficas para las que han sido predefinidas. Esto hace que la adaptación de nuevos tipos de gráficas a las necesidades de los investigadores requiera largos procesos de desarrollo. Una forma de solventar este problema sería el uso de sistemas genéricos y modulares de visualización, que

sean fácilmente interoperables e integrables en aplicaciones según la necesidad de los análisis.

El presente trabajo se centra en el estudio y desarrollo de estos sistemas genéricos y modulares. Con ellos, se busca permitir la realización de visualizaciones de cualquier análisis, favoreciendo la adaptabilidad mediante la combinación de diferentes componentes y visualizaciones agnósticas del problema biológico. En esta tesis se abordará también el estudio de la creación de aplicaciones a base de la combinación de estos componentes interoperables, y se propondrá que sean estas aplicaciones las que contengan la lógica y las propiedades del problema biológico a explorar.

ABREVIATURAS

Sigla	Inglés	Castellano
API	*Application Programming Interfaces*	Interfaz de Programación de Aplicaciones
BAM (formato)	*Binary Alignment Map*	Mapa de alineamiento binario
BED (formato)	*Browser Extensible Data*	—
BWA	*Burrows-Wheeler Aligner*	Alineador Burrows-Wheeler
CNV	*Copy Number Variation*	Variación en el Número de Copias
COSMIC	*Catalogue of Somatic Mutations in Cancer*	—
DSL	*Domain-Specific Language*	Lenguaje Específico de Dominio

DSS	*Decision Support System*	Sistema de apoyo a la toma de decisiones
EDA	*Exploratory Data Analysis*	Análisis exploratorio de datos
FASTQ (formato)	*FASTA with Quality*	—
FTP	*File Transfer Protocol*	Protocolo de transferencia de archivos
GFF (formato)	*General Feature Format*	—
HIS	*Hospital Information System*	Sistema de Información Hospitalario
HL7	*Health Level Seven*	—
ID	*Identifier*	Identificador
IDE	*Integrated Development Environment*	Entorno de desarrollo integrado
IS	*Insertion Site*	Lugar de inserción
JWT	*JSON Web Tokens*	—
LTR	*Long Terminal Repeat*	Repetición Terminal Larga

MPA	*Multi-Page Application*	Aplicación de múltiples páginas
MVC	*Model View Controller*	Modelo Vista Controlador
NGS	*Next generation sequencing*	Secuenciación de Segunda Generación
NLP	*Natural Language Processing*	Procesado de Lenguaje Natural
ORF	*Open Reading Frame*	Pauta de Lectura Abierta
PCR	*Polymerase Chain Reaction*	Reacción en Cadena de la Polimerasa
REST	*Representational State Transfer*	Transferencia de Estado Representacional
ROI	*Region Of Interest*	Región de interés
SAM (formato)	*Sequence Alignment Map*	Mapa de Alineación de Secuencias
SPA	*Single-Page Application*	Aplicación de una única página
SVG	*Scalable Vector Graphics*	Gráficos Vectoriales Escalables

TAR	*Transcriptionally Active Regions*	Regiones Activas Transcripcionalmente
TAR (formato)	*Tape ARchive*	—
UI	*User Interface*	Interfaz de Usuario
UX	*User eXperience*	Experiencia de Usuario
VAF	*Variant Allele Fraction*	Fracción alélica de una variante
VCF	*Variant Call Format*	—
W3C	*World Wide Web Consortium*	Consorcio de la World Wide Web

Índice de contenidos

Índice de figuras

1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

En estos últimos años se ha producido un crecimiento exponencial del área de la visualización de datos, potenciado en gran medida gracias al auge de la ciencia de datos y el *Big Data*, que requiere de técnicas interactivas y visualizaciones elaboradas y muy complejas.

De la misma forma, en los análisis bioinformáticos la visualización de los datos representa un componente muy importante. Muchas de las visualizaciones están orientadas a la exploración de datos, con el fin de descubrir relaciones nuevas o propiedades no descritas anteriormente de los datos. En bioinformática, al igual que en Big Data, también se necesitan componentes interactivos y flexibles que permitan por ejemplo cambiar parámetros, crear agrupaciones o ajustar series temporales. Esto es esencial para poder descubrir patrones que ayuden a generar hipótesis que se puedan testear con nuevos datos.

Por otro lado, en el campo de la bioinformática, para poder reproducir la exploración y visualización de datos interactivos se está

implantando el modelo de notebooks (como los *Jupyter Notebooks*[1] o *Rmarkdown*[2]), el uso de gráficos interactivos (como *Plotly*[3]) e incluso software que permite la creación de pequeñas aplicaciones orientadas a la exploración de datos (como *Shiny*[4]). Esta facilidad para crear interfaces ha permitido a los biólogos y bioinformáticos poder ir más allá de usar Excel (donde están los datos) para, también, visualizarlos. Ahora son capaces de crear los prototipos de sus propias aplicaciones y desarrollar bocetos de interacción entre el dato y el usuario final. Esto ha producido una nueva demanda de aplicaciones más profesionales y que sean realmente interactivas, y no simplemente gráficas estáticas de los datos.

Como matemático, diseñador gráfico y bioinformático con una gran motivación hacia las interfaces de usuario (UI) y dar una buena experiencia de usuario (UX) basada en visualizaciones, mis principales intereses son: (1) la formalización de elementos gráficos; (2) la modularización y abstracción de las aplicaciones interactivas para análisis de datos; y (3) la creación de algoritmos de análisis o visualización de datos eficientes que ayuden a una buena experiencia del usuario.

---

[1] Proyecto ***Jupyter***: https://jupyter.org

[2] Documentación de ***RMarkdown***: https://rmarkdown.rstudio.com

[3] Proyecto ***Plotly***: https://plotly.com

[4] Documentación de ***Shiny*** https://shiny.rstudio.com

Es común en muchas aplicaciones web realizar bien la parte técnica, pero mostrar el contenido de las bases de datos sin una transformación que les den sentido para el usuario. Esto normalmente ocurre porque los programadores que las han generado son buenos ingenieros, pero son ajenos al dominio de conocimiento de los datos a mostrar o explorar. Este trabajo intenta evitar esta dicotomía explorando los caminos de la visualización interactiva en genómica, uniendo la investigación técnica con la capa de conocimiento biológico a la que ha de servir.

La creación de software, destinado a la realización de visualizaciones interactivas para exploración de datos, no solo requiere una alta capacitación técnica, sino también un conocimiento de las necesidades de los investigadores y la comprensión de la naturaleza de los datos; tres cualidades que no siempre se encuentran juntas en los equipos que desarrollan aplicaciones bioinformáticas. Esto es importante porque el propósito inmediato de la visualización de los datos es la mejora de la comprensión. Y el propósito último de la visualización de los datos, más allá de la comprensión, es permitir la mejora en las decisiones o acciones derivadas del análisis de esos datos.

Esta comprensión debe estar orientada a la mejora de la usabilidad y la utilidad de la aplicación. La usabilidad es una característica de calidad que mide cómo de fácil de usar es una interfaz de usuario. La utilidad se refiere a si la herramienta cumple con su principal objetivo: hacer lo que los usuarios necesitan. La unión de ambos conceptos permite determinar si una herramienta es realmente útil: importa poco

que sea fácil de usar si no cumple con lo que se necesita, al igual que tampoco es bueno si puede hacer (hipotéticamente) todo lo que el usuario quiere, pero no puede ser usada porque la interfaz es compleja.

Finalmente, la primera ley del análisis exploratorio de datos es que, si los investigadores no pueden visualizar los datos, tampoco pueden analizarlos. Ejemplos clásicos de la importancia de la visualización de datos para comprender los datos son el cuarteto de *Anscombe* o «la docena del *datasuario*[5]».

Aunque cuando se comenzó esta tesis ya existían aplicaciones específicas para la creación de visualizaciones interactivas y exploración de datos, el problema es que estaban basadas en bibliotecas de alto nivel con un número limitado de gráficas. Esto implicaba que la generación de nuevas visualizaciones, adecuadas a las necesidades de los investigadores, requiriesen largos procesos de desarrollo. Esto se podría solventar con sistemas genéricos y modulares de visualización que fueran fácilmente interoperables e integrables en aplicaciones según la necesidad de análisis.

El presente trabajo se centra en el estudio y desarrollo de estos sistemas genéricos y modulares. Con ellos, se busca permitir la realización de visualizaciones de cualquier análisis, favoreciendo su adaptabilidad mediante la combinación de diferentes componentes y

---

[5] *The datasaur dozen dataset*: https://blog.revolutionanalytics.com/2017/05/the-datasaurus-dozen.html

visualizaciones agnósticas del problema biológico. En esta tesis se abordará también el estudio de la creación de aplicaciones a base de la combinación de estos componentes interoperables, y se propondrá que sean estas aplicaciones las que contengan la lógica y las propiedades del problema biológico a explorar.

1.2 Objetivos

El objetivo general de esta tesis doctoral es el estudio y desarrollo de metodologías de visualización y su integración en aplicaciones bioinformáticas que permitan una exploración interactiva y reactiva de datos genómicos en biomedicina. Los resultados de esta investigación deben tener una capa translacional y deben poder ser integrados tanto en entornos de investigación como en biotecnológicos, sin descartar, además, los sistemas sanitarios orientados al diagnóstico.

Una parte fundamental de este trabajo se ha centrado sobre todo en la formalización e implementación de una gramática de visualización reactiva y unos modelos de arquitectura de aplicaciones biomédicas, basados en componentes modulares, que permitan la rápida creación de aplicaciones multifuncionales sin grandes costos de integración. Finalmente, dentro de los objetivos de este trabajo se plantea la creación de las pruebas de concepto (**MGvizApps**) que validen los resultados de esta investigación, para su uso en la industria relacionada con la genómica en los sistemas de salud.

En definitiva, modernizar la forma en cómo el analista visualiza, explora e interactúa con los datos para la generación de informes o anotación e interpretación de los resultados genómicos.

Este objetivo principal se puede dividir en los siguientes sub-objetivos:

1. Estudio e implementación de métodos de visualización, manejo y análisis de datos usados en los flujos de trabajo de análisis genómico con nuevos métodos de secuenciación (NGS).
2. Desarrollo de nuevos métodos y aplicaciones bioinformáticas para la detección de CNVs tanto en muestras germinales como en cáncer.
3. Creación de un sistema de visualización para dotar a las aplicaciones de análisis genómicas de una adecuada capa de reactividad e interactividad (análisis a tiempo real). Formalizar las especificaciones de este sistema, que llamaremos **Jviz** y su implementación mediante una biblioteca de *JavaScript* para su uso en aplicaciones web biomédicas.
4. Creación de un sistema de componentes que facilite la usabilidad y permita la interoperabilidad para la creación de aplicaciones web modulares.
5. Explorar métodos que faciliten la creación de aplicaciones bioinformáticas para la generación de informes biomédicos basados en análisis de NGS. Se priorizará que las aplicaciones se ejecuten en el lado cliente pero que sean fácilmente extensibles a sistemas cliente-servidor.

1.3 Estructura de este documento

Este documento está dividido en 5 capítulos, con una sección adicional para referencias bibliográficas y un apéndice con la copia de los artículos publicados como resultado de esta tesis.

1. Introducción, objetivos y estructura

En este primer capítulo, se muestra una breve introducción a la importancia de la visualización en bioinformática para la realización de análisis exploratorio de datos y búsqueda de evidencias. A continuación, se presenta el objetivo principal de la tesis y la motivación que ha llevado a la creación de las contribuciones aportadas en este trabajo y por último la estructura del documento de tesis.

2. Estado del arte

En este apartado se introducen los conceptos necesarios para entender el funcionamiento general de los sistemas de visualización interactivos y se describe el estado actual de los principales sistemas

y *frameworks* para desarrollar aplicaciones web que permitan la interacción del usuario en tiempo real para la exploración de datos.

3. Resultados

Los resultados de esta tesis se agrupan en 5 subcapítulos. Los tres primeros representan aplicaciones web siguiendo las tecnologías establecidas y bibliotecas de código estándar. En cada una de ellas, además de los retos de visualización y técnicas de usabilidad, se desarrollaron algoritmos de análisis bioinformático para responder a preguntas biológicas concretas por parte de los usuarios de los laboratorios.

El apartado 3.1 presenta la aplicación **TilingScan**, diseñada para el análisis de expresión diferencial en levaduras a partir de datos de GeneChip Tiling Arrays.

El apartado 3.2 presenta la aplicación **VISMapper**, que permite la detección y visualización de lugares de inserción de virus en experimentos de terapia génica.

El apartado 3.3 presenta la aplicación **SAP** (*Sequence Analysis Platform*), una aplicación de análisis de datos de NGS automatizada con una aplicación web para seguimiento del estado del análisis y priorización e informe de variantes clínicamente relevantes.

En el apartado 3.4 se desarrolla el cuerpo principal y la aportación más novedosa de este trabajo, el sistema de visualización **jviz**, que incluye una gramática para la descripción de gráficos reactivos, un intérprete en JavaScript para poder generar dichos gráficos y un editor online.

Finalmente, el apartado 3.5 representa la culminación de todo este trabajo donde se integra toda la tecnología y algoritmos desarrollados como parte de este trabajo en la plataforma **MGvizApps**, que ofrece un marco para la integración de aplicaciones genómicas. En este apartado se describen además 3 aplicaciones: (1) una reimplementación de **VISMapper** visto en el capítulo 3.2; (2) una herramienta llamada **SeqMask** que facilita y automatiza la tarea de preparar secuencias para los programas de diseño de primers; y (3) una aplicación denominada **CNVReporter**, que permite la exploración de variaciones de número de copias en datos de NGS y la generación de los consiguientes informes.

4. Discusión

En este apartado se analiza la importancia de los resultados obtenidos y se ponen en contexto con otras tecnologías actuales, evaluando además si estos resultados validan la importancia de los desarrollos modulares como hipótesis válida para la mejora de UI/UX.

5. Conclusiones

En este último capítulo se presentan las conclusiones de este trabajo, junto con los posibles trabajos futuros que se podrían derivar de este trabajo.

2

ESTADO DEL ARTE

Como se ha visto en el apartado 1.2, el objetivo de esta tesis es el estudio y desarrollo de los elementos que permitan la creación de aplicaciones web reactivas para el análisis de datos biomédicos. En este caso, está orientado principalmente hacia datos genómicos en investigación y entornos clínicos, con un alto contenido en análisis exploratorio de datos tanto para la generación de hipótesis científicas como para una interpretación médica.

Para cumplir este objetivo se han estudiado las metodologías usadas para la creación de aplicaciones web como son, la creación de

interfaces de usuario (UI) y la mejora de la experiencia del usuario (UX), incluido la visualización reactiva.

En este apartado se introducen los conceptos necesarios para entender el funcionamiento general de los sistemas de visualización interactivos. Para esto, se describirá el estado actual de los principales sistemas y frameworks para desarrollar aplicaciones web, los cuales permitan la interacción del usuario en tiempo real para la exploración de datos.

Estructura de las aplicaciones web modernas

Una aplicación web es una compleja pieza de software formada por un conjunto de diversos componentes, como por ejemplo la interfaz de usuario, una pantalla de inicio de sesión, una base de datos, servicios de compras, etc. Para administrar estos componentes, los desarrolladores han de diseñar una arquitectura de aplicaciones web para definir las relaciones lógicas y las interacciones entre todos estos componentes en una aplicación web.

En general, el desarrollo de aplicaciones web se puede dividir en dos partes: el *front end* y el *back end (***Figura 2.1***)*. El ***front end*** (o lado cliente) es la parte de la aplicación que los usuarios ven y con la que interactúan. Por otro lado, el ***back end*** (o lado servidor) es la parte a la que el usuario no tiene acceso; esto incluye por ejemplo el servidor donde se encuentran los archivos de la aplicación y la base de datos donde se conservan los datos del usuario.

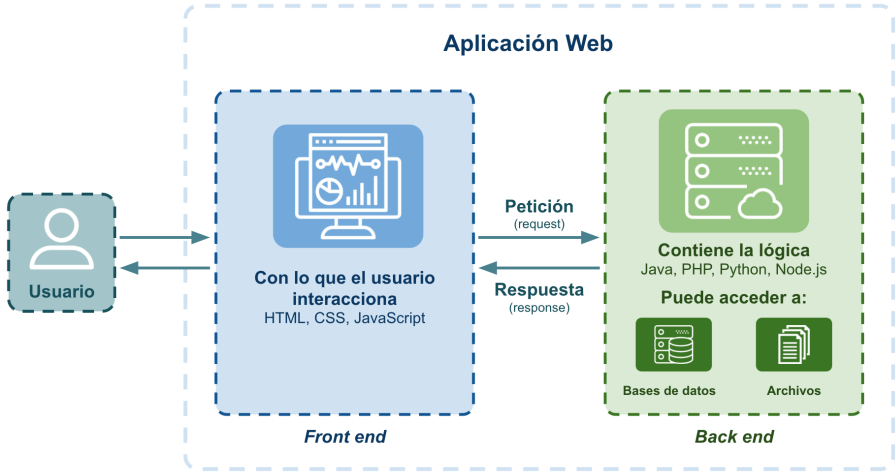


Figura 2.1. Estructura general de una aplicación web.

Front end

El ***front end***, también conocido como *cliente* o «lado del usuario», incluye todo lo que se muestra al usuario por medio del navegador web (texto, imágenes, animaciones, etc.).

Cuando un usuario entra en una página web, el navegador realiza una petición al servidor en la que está alojada dicha página. Como resultado a esta petición, el servidor envía una serie de archivos y datos, que son procesados y mostrados por medio del navegador web. En términos generales, esta respuesta está compuesta por archivos HTML, CSS, *JavaScript*, imágenes y otros archivos que sean también necesarios para la página que se esté solicitando. El navegador, una vez ha recibido todos estos archivos, los procesa y muestra la página web al usuario.

Las tres tecnologías que permiten el desarrollo tanto de páginas como de aplicaciones en la web son las siguientes:

- **HTML** (*Hypertext Markup Language*), es un lenguaje de etiquetas principalmente utilizado para organizar y dar formato a las páginas y aplicaciones web.
- **CSS** (*Cascading Style Sheets*), es un lenguaje de hojas de estilo que se utiliza para describir la presentación de un documento escrito en HTML. Mediante CSS se describe cómo se deben representar los elementos de la página o aplicación web.
- **JavaScript**, es el lenguaje de programación flexible que proporciona la capa de interactividad a las aplicaciones y páginas web.

Back end

El ***back end*** se conoce como la parte de las aplicaciones o páginas web que vive dentro del servidor, también conocido como «lado del servidor». El *back end* de una aplicación o página web suele constar de varios elementos, como por ejemplo el servidor (que almacena el contenido y contiene la lógica) y las bases de datos (donde se almacena por ejemplo la información de los usuarios).

Para el desarrollo del *back end* de una página o aplicación web se necesita un lenguaje de programación específico (Java, PHP, Ruby, Node.js, etc.), diferente al que se utiliza para el desarrollo del *front end*.

Arquitectura de las aplicaciones web

Actualmente existen dos tipos de arquitecturas básicas para el diseño de las aplicaciones web: las aplicaciones de varias páginas (MPA) y las aplicaciones de una sola página (SPA) (**Figura 2.2**).

Las aplicaciones que se basan en la metodología de ***Multi-Page Applications*** (MPA) siguen el modelo clásico de desarrollo web, en la que la navegación a través de la aplicación se realiza por medio de varias páginas conectadas entre sí. Cualquier cambio o acción que se realice en la aplicación, como por ejemplo mostrar nuevos datos o enviar un formulario, requiere que el servidor genere una nueva página que debe ser recargada en el cliente. Algunos ejemplos de aplicaciones web que siguen la metodología MPA son *Amazon*[6] y *Wikipedia*[7].

Por otro lado, la metodología ***Single-Page Applications*** (SPA) sigue un modelo diferente al clásico, en el que la aplicación está formada por una única página donde la lógica, contenido y navegación se gestiona mediante *JavaScript*. Esto elimina la necesidad de tener que cargar páginas adicionales, haciendo que la navegación entre las diferentes vistas de la aplicación sea más fluida y dinámica. Las aplicaciones que se desarrollan con este modelo utilizan peticiones

[6] Sitio web de **Amazon**: https://www.amazon.com

[7] Sitio web de **Wikipedia**: https://www.wikipedia.org

AJAX[8] o *Fetch*[9] para obtener los datos a mostrar en la aplicación o para realizar el envío de formularios mediante métodos *POST*. *Google Maps*[10] y la nueva versión de la red social *Facebook*[11] son ejemplos de aplicaciones que siguen este enfoque.

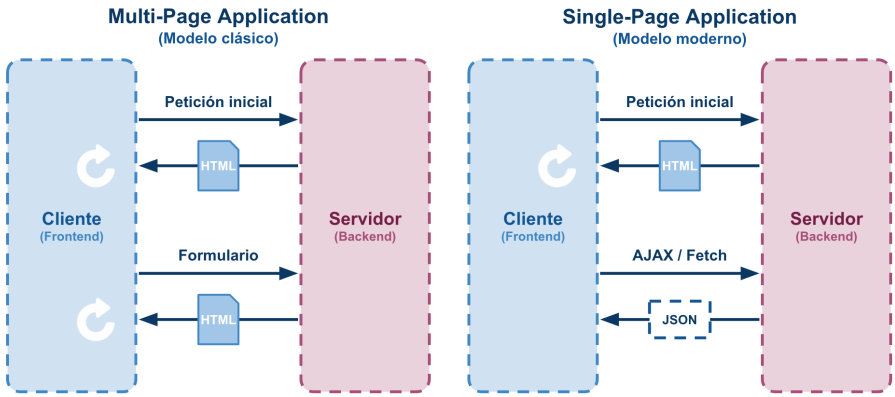


Figura 2.2. *Esquema del funcionamiento de los modelos MPA y SPA. En ambos, tras realizar la petición inicial se recibe una nueva página HTML que se carga en el cliente. La diferencia radica en el punto en el que tienen que enviar datos al servidor: en el modelo MPA esto se realiza enviando un formulario, lo que hace que el servidor genere una nueva página HTML que debe ser cargada en el cliente; por el contrario, en el modelo SPA el envío de*

---

[8] Especificación de la tecnología **AJAX**: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[9] Especificación de la tecnología **Fetch**: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[10] Aplicación **Google Maps**: https://www.google.com/maps

[11] Red social **Facebook**: https://www.facebook.com

datos se realiza utilizando tecnologías AJAX, cuyo resultado es un JSON (o XML) que se utiliza en la página actual sin tener que actualizarse.

Dependiendo del tipo de proyecto, habría que valorar la elección de un modelo basado en una SPA o en una MPA. Las SPA están más orientadas para aplicaciones pequeñas, mientras que en las MPA suelen utilizarse en aplicaciones más grandes con una gran cantidad de vistas. En la actualidad, muchas aplicaciones están apostando por un **modelo híbrido**, en el que para la parte importante de la aplicación se utiliza un modelo basado en una SPA, mientras que para el resto de páginas de la aplicación se utiliza el modelo clásico MPA. *GitHub*[12] es un ejemplo de aplicación con una arquitectura híbrida.

Introducción a la interfaz de usuario (UI)

La UI y UX en las aplicaciones bioinformáticas

El estudio de las interfaces de la experiencia de usuario, pese a haber sido un tema muy estudiado en la última década, no se ha producido grandes avances en la modernización de las principales aplicaciones bioinformáticas.

La mayoría de herramientas actuales siguen un modelo semi-estático de visualización como *Ensembl*[13] y UCSC[14], usadas por miles de

[12] Sitio web de **GitHub**: https://github.com

[13] Sitio web de **Ensembl**: https://www.ensembl.org/index.html

[14] Sitio web de **UCSC**: https://genome.ucsc.edu

investigadores en genómica y que siguen teniendo básicamente la misma interfaz que hace 10 años (**Figura 2.3**).

Con respecto a las aplicaciones más dinámicas, al inicio del desarrollo de esta tesis el número de aplicaciones reactivas para biomedicina eran escasas, principalmente prototipos o aplicaciones con tecnología reactiva reusadas de otros ámbitos de conocimiento por gente muy competente técnicamente, pero sin conocimiento del problema biomédico a resolver. El resultado de esto eran aplicaciones con un agradable aspecto, pero de poca utilidad en el día a día como IOBIO[15] (**Figura 2.4**). Este tipo de aplicaciones han seguido caminos interesantes y atractivos para la modernización de las UI y su reactividad, pero se han distanciado de la capa «biológica» de los datos como elemento de análisis y han perdido la «interactividad biológica» para la exploración de los datos.

[15] Sitio web de **IOBIO**: https://iobio.io

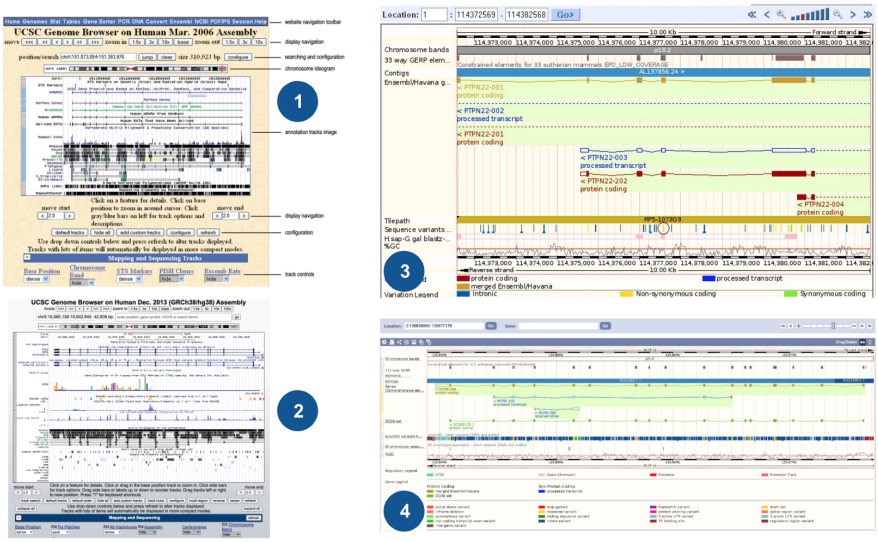


Figura 2.3. *UCSC en 2008(1) y en 2020(2). Ensembl en 2010(3) y en 2020 (4).*

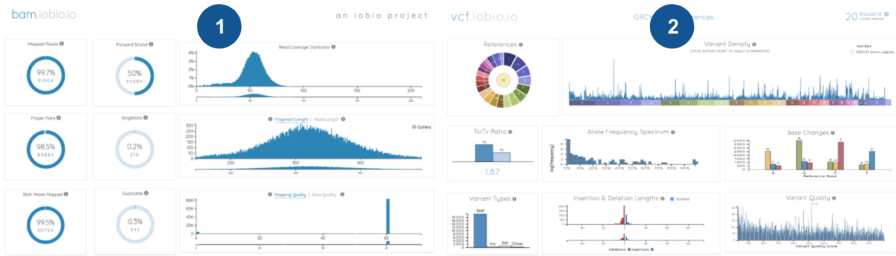


Figura 2.4. *Herramienta **IOBIO** para el análisis de **BAM** (1) y **VCF** (2). Ejemplo de aplicación tecnológicamente moderna, pero que no utiliza las posibilidades que brinda dicha tecnología para la mejora de la exploración de los datos biológicos.*

En 2019, Gehlenborg presentó una revisión y clasificación taxonómica de los gráficos en visualización e interfaces de datos genómicos. Para el estudio se centró en aquellos gráficos e interfaces con una gran

aceptación en los ámbitos de visualización biológica. La orientación y criterio de este estudio coincide bastante con lo presentado en este trabajo (**Figura 2.5**).

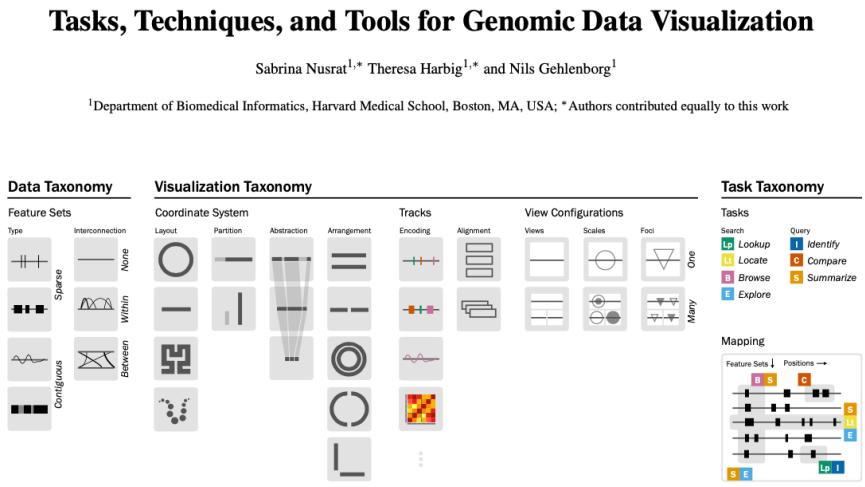


Figura 2.5. *Revisión de visualización en datos genómicos por Gehlenborg (2019) y su propuesta de una taxonomía visual.*

Herramientas para desarrollos de UI y UX modernos

Antes de presentar las diferentes herramientas que ayudan en el desarrollo de las interfaces de usuario, es necesario introducir la diferencia entre los conceptos ***framework*** y **biblioteca**. En líneas generales, ambos son códigos que están escritos en un lenguaje de programación, cuya utilidad es ayudar a resolver problemas comunes. Muchos desarrolladores suelen usar estos dos términos de manera intercambiable, pese a que existen grandes diferencias entre ambos como se muestra a continuación (**Figura 2.6**).

Una **biblioteca** es una colección de métodos o funciones, que se construyen con la idea de la reutilización del código. En una biblioteca, el control lo tiene el desarrollador, pues es él que decide cuándo debe utilizar las funciones de la biblioteca y cuándo no. La biblioteca se utiliza de apoyo al desarrollador en la creación de la aplicación, y no dicta ni el flujo ni el diseño que debe seguir. El desarrollador es quien tiene el control.

Por el otro lado, en un ***framework*** todo el flujo de control ya está predefinido, y el desarrollador únicamente debe completar una serie de huecos definidos por el *framework* con el código específico de la aplicación. De esta forma, es el framework el que utilizará el código implementado por el desarrollador cuando sea necesario, todo lo contrario de lo que ocurre en una biblioteca. Cuando se usa un framework el desarrollador NO tiene el control. En inglés se definen como *opinionated frameworks*.

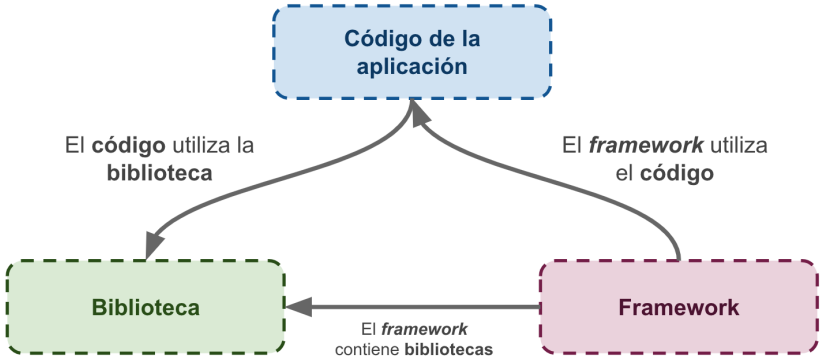


Figura 2.6. *Esquema de las relaciones entre el código de la aplicación, una biblioteca y un «framewok».*

El beneficio de utilizar un *framework* frente a una biblioteca es que los desarrolladores no necesitan preocuparse por elegir entre un diseño u otro: el desarrollador únicamente se debe preocupar en añadir las funciones específicas del dominio de la aplicación. Por otro lado, una biblioteca proporciona más libertad y control al desarrollador que un *framework*, de forma que en cualquier momento se puede cambiar de una biblioteca a otra sin que esto afecte a la aplicación.

La diferencia entre estos dos modelos se puede ilustrar mediante una sencilla metáfora. Una biblioteca es como usar el planificador de *Ikea* una vez ya tienes una casa, pero necesitas ayuda con los muebles. No tienes el tiempo o el conocimiento para construir tu cocina desde cero, de forma que *Ikea* te permite elegir diferentes componentes predefinidos pero modulares y combinables. En este caso, tú decides cómo combinarlos o incluso si usarlos para la misma función con la que *Ikea* los ha definido en su catálogo. En este caso, tú tienes el control.

Por otro lado, un *framework* es como encargar la construcción de una casa prefabricada. El contratista dispone de un conjunto de planos y algunas opciones limitadas en lo referente a la arquitectura y el diseño. En última instancia, es la constructora quien tiene el control, y te informará sobre cuándo y dónde puedes dar tu opinión.

A continuación, se va a realizar una comparación de las principales bibliotecas y frameworks que se tuvieron en cuenta para el desarrollo de esta tesis. En la **Figura 2.7** se muestra la evolución temporal de la popularidad de diferentes softwares para la construcción de interfaces de usuario.

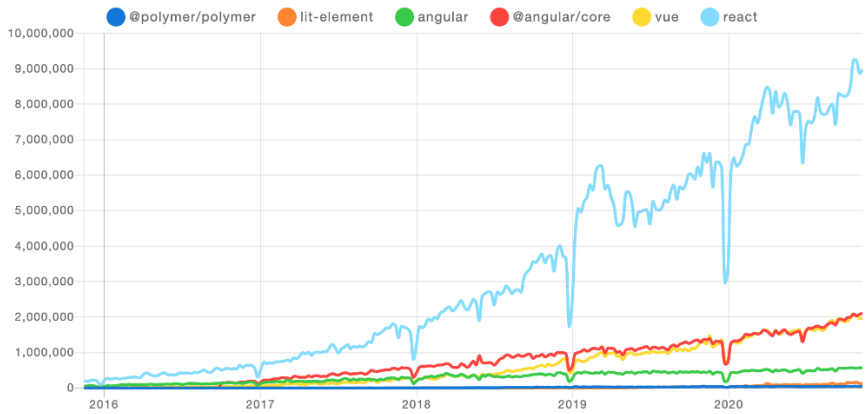


Figura 2.7. *Evolución del número de descargas en los últimos 5 años de las librerías comentadas en esta sección:* ***Polymer*** *(azul),* ***LitElement*** *(naranja),* ***AngularJS*** *(verde),* ***Angular*** *(rojo),* ***Vue****.js (amarillo) y* ***React*** *(azul claro). Fuente: https://www.npmtrends.com/@polymer/polymer-vs-lit-element-vs-angular-vs-@angular/core-vs-vue-vs-react*

Web Components, Polymer y LitElements

Los ***Web Components*** [16] son una colección de características estandarizadas por la **W3C** [17] que permiten la creación de componentes aislados, y que pueden ser integrados en cualquier aplicación o página web sin necesitar de ninguna librería o framework adicional. Estas características incluyen los ***custom elements*** (que permiten encapsular en una etiqueta HTML todo un componente funcional), las **plantillas de HTML**, el **Shadow DOM**, y los ***HTML***

[16] Introducción a los **Web Components**: https://www.webcomponents.org

[17] Especificación por la **W3C**: https://www.w3.org/TR/components-intro

Imports. Sin embargo, desde abril de 2019 esta característica ya no está soportada[18], quedando reemplazada por los ***ES Modules*** de *JavaScript*.

El principal problema de los *Web Components* es el soporte por parte de los navegadores web. No todos los navegadores soportan esta especificación, por lo que una aplicación que utilice alguna de estas características puede funcionar perfectamente en un navegador, pero no en otro. Para dar solución a este problema, existen librerías específicas, denominadas como *polyfills*, que simulan este tipo de funcionalidades que pueden no estar disponible de forma nativa.

Polymer[19] es una biblioteca desarrollada por Google que facilita la creación y el uso de los *Web Components* en las aplicaciones o páginas web. La biblioteca *Polymer* proporciona una serie de *polyfills* que permiten que todas las características que ofrecen los *Web Components* puedan ser utilizadas en cualquier navegador.

La primera versión de *Polymer* fue liberada en 2015, y trabajaba con la primera versión (v0) de los *Web Components*. Esta versión obligaba a que los *Custom Elements* debían estar definidos en archivos HTML independientes, de forma que las aplicaciones que querían utilizar esos componentes debían importar estos archivos HTML utilizando la característica de los *HTML Imports*. En la segunda versión de

---

[18] Estado de los ***HTML Imports***: https://www.chromestatus.com/feature/5144752345317376

[19] Proyecto ***Polymer***: http://polymer-project.org

Polymer, liberada a principios del 2017, se dio el salto a la versión estable de los *Web Components* (v1), aunque la forma de trabajar con ella no difería mucho de la primera versión.

Sin embargo, debido a que la característica de los *HTML Imports* fue descontinuada, obligó a sacar la versión 3 de la biblioteca *Polymer* que pasaba de definir los componentes en archivos HTML a definirlos en archivos *JavaScript*. Esta versión fue presentada a mediados del 2017, pero no fue hasta un año después que se hiciera la liberación de la versión estable.

Por otro lado, ***LitElement*** es la biblioteca sucesora de *Polymer*. Sigue la misma filosofía que la última versión de *Polymer*, pero delega muchas de sus funcionalidades en el navegador y se apoya más en las nuevas características del *ES6* de *JavaScript*. *LitElement* se apoya en otra biblioteca llamada ***LitHTML*** que permite crear las plantillas de los componentes utilizando *JavaScript*. *Polymer* 3 se encuentra ahora mismo en mantenimiento y no se esperan nuevas versiones. Los mismos desarrolladores recomiendan utilizar *LitElement* en lugar de *Polymer* 3 para nuevas aplicaciones o componentes.

Angular

Angular[20] es un *framework* de *TypeScript* orientado específicamente al desarrollo de aplicaciones SPA, siguiendo el paradigma Modelo-

[20] Documentación de **Angular**: https://angular.io

Vista-Controlador (MVC). *Angular* proporciona un entorno estructurado para poder desarrollar este tipo de aplicaciones.

Actualmente existen dos versiones de este *framework*: la primera versión se conoce como ***AngularJS***, mientras que a la segunda versión (y siguientes) se conoce como ***Angular***. Este cambio de nombre es debido a una reescritura total del *framework* al pasar de la primera a la segunda versión, haciendo que sean totalmente incompatibles.

Sin embargo, el desarrollo por medio de *Angular* tiene una serie de desventajas. La primera gran desventaja es su pronunciada curva de aprendizaje. Aprender *Angular* requiere aprender también *TypeScript*, que es un lenguaje que extiende *JavaScript* proporcionando algunas características inexistentes. Por otro lado, el hecho de que *Angular* proporcione una estructura específica hace que sea más complicado migrar un proyecto ya existente a *Angular*.

Vue.js

Vue.js[21] es una biblioteca relativamente nueva que ha ganado gran popularidad por su facilidad de uso y baja curva de aprendizaje para obtener resultados básicos. La primera versión fue liberada en 2014. La idea original de esta biblioteca es que fuera una versión ligera de *Angular*, pero sin dictar cómo debe estar estructurada la aplicación.

[21] Documentación de **Vue.js**: https://vuejs.org

Vue.js se basa en el uso de plantillas HTML, lo que hace que sean fáciles de leer y escribir para gente que venga de trabajar con otras librerías como *Polymer*. Esto también hace que la curva de aprendizaje sea más suave comparada con la de otras bibliotecas.

React

React[22] es una biblioteca de *JavaScript*, desarrollada por la compañía *Facebook* y cuya primera versión fue liberada en 2013. El principal usuario de *React* es la propia compañía *Facebook* (que lo utiliza para su tan conocida red social), aunque otras empresas importantes como *Airbnb*, *PayPal* y *Walmart* también lo utilizan.

Se puede considerar *React* más como una biblioteca de *JavaScript* que como un *framework*, ofreciendo así al desarrollador mucha flexibilidad a la hora de cómo estructurar el proyecto. *React* puede ser utilizado tanto para crear nuevos proyectos como para ser integrada únicamente en ciertas partes de un proyecto.

Una de las características importantes de *React* es el uso que hace del ***Virtual DOM***. El *Virtual DOM* es una representación en memoria del DOM, mucho más rápida de manipular que el DOM real del navegador. Al producirse cualquier cambio en los componentes definidos con *React*, este utiliza ese *Virtual DOM* para obtener los cambios mínimos que se deben hacer sobre el DOM real. De esta forma, el acceso y manipulación del DOM se limita únicamente a lo

[22] Documentación de **React**: https://reactjs.org

estrictamente necesario, reduciendo el tiempo que tarda en aplicar los cambios.

La gran ventaja que proporciona *React* frente a otras bibliotecas reside en la capacidad que proporciona a los desarrolladores para crear aplicaciones complejas, de forma que permite separar las piezas que la componen y que puedan reutilizarse en otras aplicaciones o proyectos.

React define un lenguaje propio denominado **JSX**[23] como extensión de *JavaScript*. *React* utiliza este lenguaje para definir los componentes como si se tratase de HTML. Debido a que el lenguaje *JavaScript* no es capaz de entender JSX, se necesita de un *transpiler* que convierta el código escrito con JSX en código que un navegador pueda entender. Ejemplos de *transpilers* son *Babel*[24] o *Sucrase*[25].

Como extensión de *React* existe el proyecto ***React Native***[26], que permite desarrollar aplicaciones móviles (tanto para *Android* como para *iOS*) de forma nativa utilizando componentes de *React*. Así, estos componentes pueden ser compartidos tanto en aplicaciones web como en aplicaciones móviles.

---

[23] Documentación de **JSX**: https://reactjs.org/docs/introducing-jsx.html

[24] Documentación de **Babel**: https://babeljs.io

[25] Documentación de **Sucrase**: https://sucrase.io

[26] Documentación de **React Native**: https://reactnative.dev

Formatos de gráficos interactivos en web

A la hora de trabajar con gráficos interactivos en web, actualmente existen dos estándares consolidados: ***Canvas*** para trabajar con mapas de bits y **SVG** para trabajar con imágenes vectoriales.

En ambos casos se utiliza un elemento HTML como contenedor del gráfico, en el cual se define las propiedades básicas del mismo (altura, anchura, etc..). La diferencia es que en *Canvas* las instrucciones para generar el gráfico se definen fuera de ese contenedor, utilizando *JavaScript* y referenciando al contenedor (**Código 2.1**), mientras que, por el contrario, en SVG los elementos que definen el gráfico están dentro del contenedor, siguiendo la misma sintaxis que HTML (**Código 2.2**).

```
<!-- Elemento canvas de HTML -->
<canvas id="draw" width="120" height="120"></canvas>

<!-- Código JavaScript para dibujar en el canvas -->
<script type="text/javascript">
    let canvas = document.getElementById("draw");
    .....
</script>
```

Código 2.1*. Creación de un elemento Canvas de HTML y manipulación de este mediante JavaScript.*

```
<!-- Elemento SVG de HTML -->
<svg width="50" height="50"
xmlns="http://www.w3.org/2000/svg">
```

```
  <g fill="none" stroke="black" stroke-width="3" >
    <line x1="0" y1="1.5" x2="100" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="100" />
  </g>
   .....
</svg>
```

***Código 2.2**. Creación de un elemento SVG en HTML.*

Canvas

El elemento ***Canvas***[27] de HTML proporciona un lienzo donde se puede crear y manipular imágenes representadas píxel a píxel, utilizando una API en *JavaScript*. Utilizando esta API se puede dibujar una gran variedad de elementos, como por ejemplo arcos, rectángulos, texto, etc. También permite la manipulación de imágenes píxel a píxel e incluso la realización de animaciones.

El elemento *Canvas* es ampliamente utilizado en las aplicaciones web para la representación de gráficas, así como para el desarrollo de videojuegos o para la edición de imágenes.

Por desgracia, el rendimiento de las aplicaciones desarrolladas con *Canvas* desciende a medida que aumentan la resolución del cliente (ya que aumenta el número de píxeles a dibujar en cada iteración). Debido a ello, se desaconseja su uso en aplicaciones que puedan ser

[27] Especificación de **Canvas**: https://www.w3.org/TR/2dcontext

utilizadas en el modo de pantalla completa o en monitores con resolución 4K.

SVG: Scalable Vector Graphics

El formato **SVG** [28] es un formato de gráficos vectoriales en dos dimensiones que está basado en XML. Las imágenes vectoriales se basan en elementos geométricos, a los cuales se les asigna unas coordenadas matemáticas (con lo que se define su posición y dimensión en el gráfico) junto con unos atributos de estilo (color, bordes, etc.).

Las imágenes vectoriales tienen la ventaja de ser más adaptables y manejables que las imágenes de mapa de bits. Gracias a que están basadas en fórmulas matemáticas, tienen una resolución infinita, de forma que se pueden escalar sin que su calidad se vea afectada. Además, gracias a que se basan en fórmulas y no en píxeles, consiguen almacenar información compleja en menor espacio.

El formato SVG fue desarrollado por la W3C, cuya primera especificación estable fue presentada en 2001. Inicialmente el uso de SVG en la web fue muy limitado debido a la falta de soporte para las imágenes vectoriales en algunos navegadores, como por ejemplo *Internet Explorer*. *Konqueror*[29] fue el primer navegador que admitió imágenes SVG en el año 2004. Después de eso, aunque lentamente,

---

[28] Especificación de **SVG**: https://www.w3.org/TR/SVG2

[29] Proyecto ***Konqueror***: https://www.konqueror.org

el resto de navegadores fueron añadiendo soporte para las imágenes vectoriales. En la actualidad, las imágenes SVG están soportadas de forma nativa tanto en los navegadores de escritorio como en los navegadores móviles.

Comparación SVG y Canvas

Tanto SVG como *Canvas* son tecnologías que permiten dibujar de forma interactiva en los navegadores web. Pese a que en ambas se pueden llevar a cabo casi las mismas acciones, estas dos tecnologías funcionan de forma totalmente diferente, y tienen rendimientos diferentes (**Figura 2.8**).

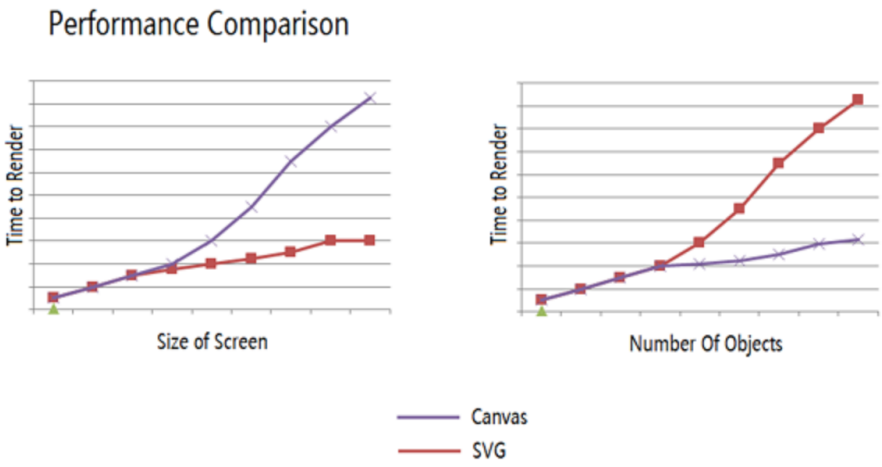


Figura 2.8. *Comparación del rendimiento entre* ***SVG*** *y* ***Canvas****. En la gráfica de la izquierda se compara el tamaño de la pantalla contra el tiempo que tarda en dibujarse. En la gráfica de la derecha, se compara el número de objetos dibujados contra el tiempo que tarda en dibujarse. Fuente: https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/samples/gg193983(v=vs.85).*

Para poder entender la diferencia en el funcionamiento de estas dos tecnologías, es fundamental distinguir entre los dos modos de mostrar contenido en el navegador: el **modo inmediato** y el **modo retenido**.

En el **modo inmediato**, el desarrollador utiliza una serie de comandos de la API para manipular el gráfico, los cuales son transmitidos al navegador que directamente los ejecuta sobre el gráfico. La API de *Canvas* es un ejemplo de modo inmediato.

Por otro lado, en el **modo retenido**, el desarrollador utiliza también los comandos de la API para manipular el gráfico, pero es la API la que genera un modelo del resultado final en memoria, tras lo cual lo traduce en comandos de dibujo al navegador. La API de SVG es un ejemplo de modelo retenido.

Dicho de otra forma, con *Canvas* se dibujan los píxeles de la imagen y el sistema se olvida de ellos, lo que reduce la memoria que necesita para mantener el modelo interno del dibujo. La ventaja de este modelo es la gran cantidad de elementos que se pueden dibujar sin que se reduzca el rendimiento (**Figura 2.8**, derecha). Por el contrario, en SVG al ser un modelo retenido, cada elemento que es dibujado se agrega al modelo interno del DOM del navegador, lo que afecta al rendimiento cuando se trabaja con imágenes que están formadas por muchos elementos (**Figura 2.8**, derecha).

Por otro lado, el hecho de que *Canvas* no mantenga un modelo interno del dibujo obliga a que por cualquier pequeña modificación que se necesite hacer en el gráfico, este se tenga que volver a dibujar por completo, lo cual puede resultar muy ineficiente para pequeños cambios cuando el dibujo contiene una alta densidad de píxeles

(**Figura 2.8**, izquierda). Esto no ocurre en SVG, donde el modelo retenido hace que cualquier manipulación posterior de cualquier elemento de la imagen sea más sencilla y no requiera volver a dibujar toda la imagen (**Figura 2.8**, izquierda).

Unido a esto encontramos también la gestión de los eventos e interacciones del usuario en los dos sistemas. Por un lado, debido a que en *Canvas* el resultado final es una imagen de mapa de bits, la gestión de la interacción del usuario ha de llevarse a cabo por medio de técnicas muy avanzadas, como por ejemplo utilizando un *quadtree* (Finkel & Bentley, 1974). Por el contrario, la gestión de eventos en el caso de gráficos en SVG se realiza de forma nativa por parte del propio navegador, por el hecho de que tanto el SVG como los elementos que lo componen son también elementos del DOM.

Grammar of Graphics

Una vez revisada la tecnología para la creación de aplicaciones interactivas, ahora introduciremos los conceptos relativos a la visualización y exploración de datos.

La visualización siempre ha sido una de las fases más importantes en cualquier proceso de análisis exploratorio de datos. Esta fase permite la extracción de la información significativa y la obtención de conclusiones a partir de los datos, independientemente de la complejidad de los mismos.

Sobre la visualización, nos centramos en revisar las especificaciones y tecnologías que ayuden a controlar el aspecto y reactividad en cada uno de los elementos de un gráfico. Uno de los textos más influyentes en este aspecto fue el libro de Wilkinson (2005), *The Grammar Of Graphics*, donde se formalizó la anatomía de los elementos que componen un gráfico como una gramática.

La gramática de los gráficos es una forma de describir y expresar la construcción de cualquier gráfico para la visualización exploratoria, siguiendo un modelo lógico, estructurado y centrado en los datos. Esta manera de formalizar un gráfico permite ir más allá de poder usar solo un pequeño número de tipos de gráficos predefinidos (como gráficos de barras, de líneas, diagramas de dispersión, etc.), y abre la oportunidad de crear combinaciones infinitas manteniendo las reglas gramaticales.

Haciendo un símil con la gramática de un idioma, la gramática de los gráficos permite expresar los datos con diferentes orientaciones o modalidades, al igual que la gramática de un idioma permite expresar un concepto u oración dándole diferentes significados, permitiendo construir oraciones declarativas (declaraciones de hecho), oraciones imperativas (declaración de solicitud) u oraciones exclamativas (declaraciones de emoción). Para expresar esta variabilidad, la gramática de los gráficos ofrece siete capas (**Figura 2.9**), de tal forma que podemos utilizar un conjunto reducido de estas capas para construir un gráfico más sencillo (al igual que una oración mínima puede tener un solo sustantivo y un verbo), o hacer gráficos más complejos mediante el uso de múltiples capas (al igual que una oración puede contener por ejemplo varios sustantivos).

Las siete capas (**Figura 2.9**) que componen la gramática de los gráficos son las siguientes:

- **Datos**: se trata de la capa más importante en cualquier gráfico exploratorio, y está formado por todos aquellos datos que se desean explorar y representar de forma gráfica para una mayor comprensión en interpretación de los mismos.
- **Estética**: dentro de esta capa se incluyen todos los elementos de un gráfico que permiten una mejor interpretación del mismo, como por ejemplo los ejes, las leyendas, colores, tamaños, etc.
- **Escalas**: permiten aplicar transformaciones a los datos, como por ejemplo transformaciones de unidades o conversión de los datos en tamaños y colores.
- **Figuras geométricas**: engloba a todas aquellas figuras y elementos geométricos que se pueden utilizar para la representación de los datos en el gráfico, como por ejemplo líneas, áreas y puntos.
- **Estadística:** permite aplicar transformaciones para poder realizar resúmenes de los datos o distribuciones. Esta capa es vital para poder realizar algunos tipos de gráficos como por ejemplo histogramas o boxplots.
- ***Facets:*** permite la división del gráfico en varios subgráficos, útil para poder dividir los datos en varios gráficos y facilitar la comparación entre ellos para buscar diferencias.
- **Coordenadas**: esta capa es la responsable de computar las posiciones de los datos en el gráfico. Normalmente se utiliza el sistema de coordenadas cartesiano, aunque también se

podrían utilizar otros sistemas de coordenadas como el sistema de coordenadas polares.

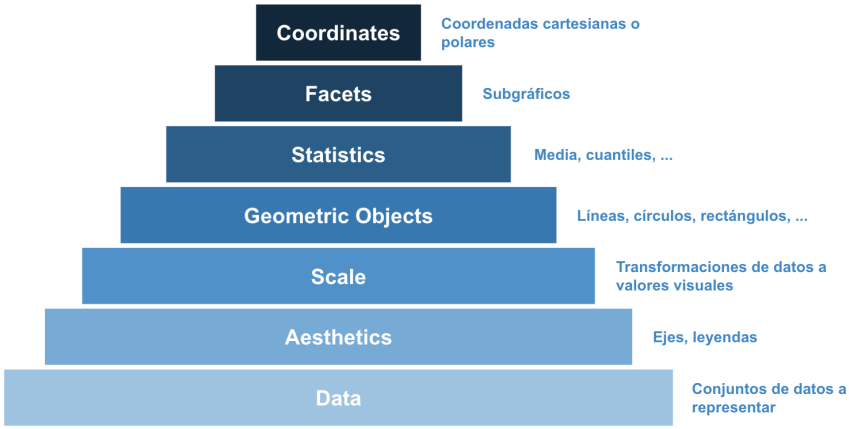


Figura 2.9. *Capas de un gráfico según la Gramática de Gráficos.*

Varias bibliotecas e implementaciones basadas en la gramática de gráficos estaban en sus inicios de desarrollo al tiempo que se comenzó este trabajo. Dos de esas bibliotecas son **ggplot2** y **Vega**, las cuales indicaron la dirección a seguir a la hora de realizar gráficos modulares y reactivos.

ggplot2

La biblioteca ***ggplot2*** (Wickham, 2016) permite la generación de gráficos en *R* con un enfoque por capas, el cual fue descrito en su trabajo *A layered grammar of graphics* (Whickham, 2010), que a su vez está basado en la gramática de gráficos de Wilkinson.

Vega

Vega se basa en la gramática de gráficos de Wilkinson para poder implementar gráficos modulares, que es la pieza base para poder hacer los gráficos reactivos. Además, *Vega* va un paso más allá e introduce la definición de las capas gramaticales como un esquema en lugar de utilizar llamadas a funciones por medio de un lenguaje de programación.

Como complemento a *Vega* existe también ***Vega-Lite***, que es una gramática de alto nivel enfocada en la generación de gráficos comúnmente utilizados en entornos de análisis exploratorio de datos (histogramas, gráficos de líneas y áreas, *boxplots*, etc.).

Reactividad

La reactividad, en las aplicaciones informáticas, es una propiedad por la cual todos los elementos del sistema se actualizan instantáneamente en respuesta a cambios externos o internos del sistema, consiguiendo que los datos, los gráficos, tablas y otros elementos visuales estén sincronizados, pudiendo construir una narrativa en tiempo real ante las interacciones de cualquier estímulo aplicado al sistema.

Un gráfico reactivo es aquel que responde a los cambios realizados por el usuario propagando dichos cambios a los elementos que corresponda. Es importante resaltar la diferencia técnica entre el concepto de reactividad e interactividad. La interactividad consiste en que el sistema acepte y responda a la intervención del usuario con el

sistema, es decir, que capte una entrada por parte del usuario y devuelva una salida adecuada a dicha entrada. La reactividad se centra en conectar internamente los procesos lógicos de estas entradas y salidas. Para la interactividad, lo que pasa entre la entrada y la salida es una «caja negra», mientras que la reactividad se encarga principalmente de lo que ocurre dentro de la «caja negra».

3

RESULTADOS

3.1 TilingScan

Bioinformatics, 31(19), 2015, 3228–3230
doi: 10.1093/bioinformatics/btv343
Advance Access Publication Date: 2 June 2015
Applications Note

OXFORD

Gene expression

A web application for the unspecific detection of differentially expressed DNA regions in strand-specific expression data

José M. Juanes[1,†], Ana Miguel[2,3,†], Lucas J. Morales[2], José E. Pérez-Ortín[2,3] and Vicente Arnau[1,*]

[1]Departamento de Informática, Escola Tècnica Superior d'Enginyeria, [2]Departamento de Bioquímica y Biología Molecular, Facultad de Biología and [3]E.R.I. Biotecmed, Universitat de València, Burjassot, Spain

*To whom correspondence should be addressed.
[†]The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors
Associate Editor: Ziv Bar-Joseph

Received on January 30, 2015; revised on May 20, 2015; accepted on May 29, 2015

Abstract

Genomic technologies allow laboratories to produce large-scale data sets, either through the use of next-generation sequencing or microarray platforms. To explore these data sets and obtain maximum value from the data, researchers view their results alongside all the known features of a given reference genome. To study transcriptional changes that occur under a given condition, researchers search for regions of the genome that are differentially expressed between different experimental conditions. In order to identify these regions several algorithms have been developed over the years, along with some bioinformatic platforms that enable their use. However, currently available applications for comparative microarray analysis exclusively focus on changes in gene expression within known transcribed regions of predicted protein-coding genes, the changes that occur in non-predictable genetic elements, such as non-coding RNAs. Here, we present a web application for the visualization of strand-specific tiling microarray or next-generation sequencing data that allows customized detection of differentially expressed regions all along the genome in an unspecific manner, that allows identification of all RNA sequences, predictable or not.

Availability and implementation: The web application is freely accessible at http://tilingscan.uv.es/. TilingScan is implemented in PHP and JavaScript.

Contact: vicente.arnau@uv.es

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Figura 3.1. *Artículo de* ***TilingScan****, publicado en 2015 en la revista* ***Bioinformatics*** *(Q1): https://pubmed.ncbi.nlm.nih.gov/26040457.*

3.1.1 **Introducción y objetivos**

Necesidad biomédica

La primera necesidad biomédica que se abordó, relacionado con el primer objetivo de esta tesis, fue la necesidad de detectar, describir y anotar todas las regiones con expresión en el genoma de levadura del género *Candida*.

Clásicamente, las secuencias de ADN de los genomas de eucariotas se han clasificado como codificantes y no codificantes. La caracterización de este tipo de secuencias se puede hacer mediante la predicción *in silico* de zonas codificantes, buscando *Open Reading Frames* (ORF) (Brent, 2007). Esta vía para buscar regiones expresadas del genoma da frutos rápidamente en regiones fácilmente identificables computacionalmente: genes codificantes, ARN ribosómico altamente conservado, ARN de transferencia y algunos ARN no codificante (ncRNA) con patrones predecibles. Pero para el resto de regiones genómicas no detectables por estos procedimientos, la mejor forma de terminar de caracterizarlos es la vía experimental, con tal de descubrir todas las zonas del genoma expresadas constitutivamente o de forma diferencial según diferentes condiciones. Uno de los métodos con más éxito para este fin son los ***tiling arrays*** (Yazaki *et al*., 2007) (**Figura 3.2**), donde se escogen sondas de forma que sean solapantes y que cubran la totalidad del genoma; sin embargo, en la actualidad se han impuesto los experimentos de secuenciación de transcriptomas mediante secuenciación masiva (NGS) (Depledge *et al.*, 2018) (**Figura 3.3**). Ambos métodos permiten

detectar los patrones de expresión a lo largo de todo el genoma, tanto de los exones conocidos como los de las zonas con expresión no documentadas. Esto posibilita su caracterización, además de la búsqueda de regiones diferencialmente expresadas que puedan ayudar a descubrir nuevos transcritos o elementos de regulación o regiones activamente transcritas (Klevebring *et al*., 2010).

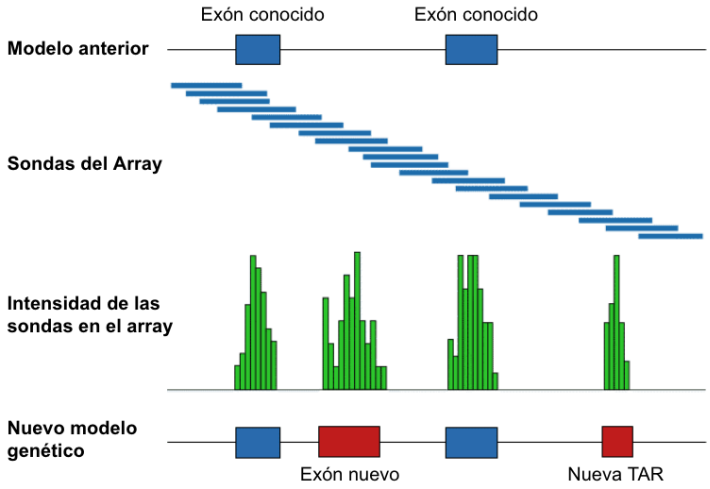


Figura 3.2. *Esquema del funcionamiento de un experimento de tiling array.*

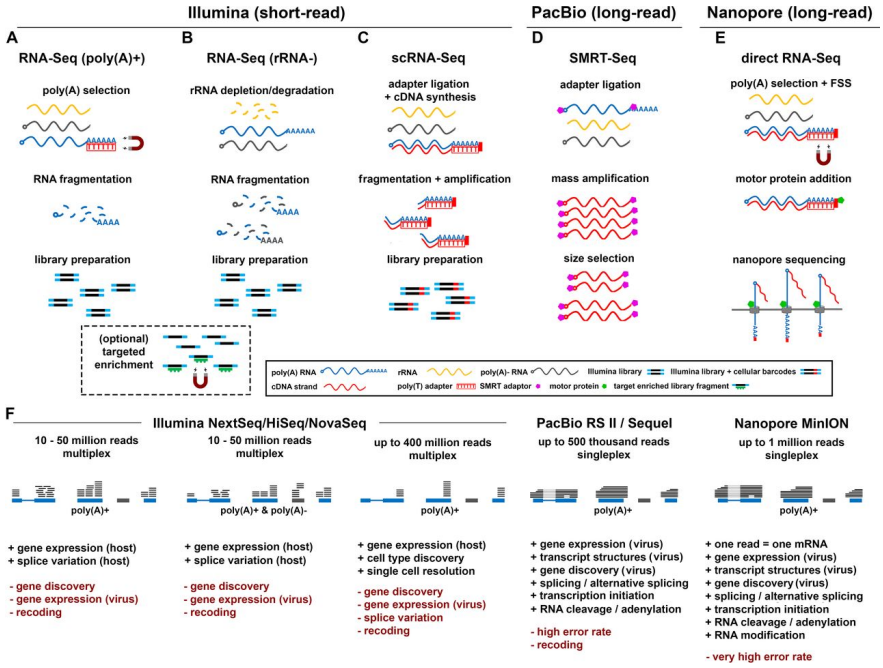


Figura 3.3. *Secuenciación de transcriptomas por NGS. Figura extraída de Depledge et al., 2018.*

Necesidad Bioinformática

A partir de datos experimentales procedentes de *GeneChip Tiling Arrays* de Affymetrix para expresión específica de hebra en genomas de levadura, se propuso la creación de una aplicación para la detección automática de zonas con expresión diferencial. La arquitectura de la aplicación está diseñada de tal forma que pueda ser adaptada a cualquier otro organismo y para cualquier sistema experimental que produzca medidas de expresión génica, como RNA-Seq.

Objetivos y solución bioinformática

Para satisfacer esta necesidad de búsqueda de regiones diferencialmente expresadas en levadura, se desarrolló **TilingScan**[30] (Juanes *et al.,* 2015), una aplicación web para la visualización de datos de *microarrays*. Esta aplicación no solo permite una exploración interactiva para la visualización y detección de dichas regiones manualmente, sino que además incorpora un nuevo algoritmo para la detección de regiones diferencialmente expresadas de manera no supervisada. De entre las diferentes técnicas desarrolladas para la detección de cambios abruptos en una señal (Basseville y Nikiforof, 1993), se decidió utilizar métodos de procesos aleatorios estacionarios y, más concretamente, se optó por implementar y adaptar una versión del algoritmo de media móvil.

El algoritmo desarrollado realiza una lectura de la señal utilizando ventanas escalables y deslizantes, y busca regiones sobreexpresadas o subexpresadas de un tamaño mínimo de nucleótidos flanqueados por regiones neutras. En lugar de centrarse exclusivamente en los cambios de expresión en los genes codificadores de proteínas, la aplicación detecta los cambios de expresión que ocurren a lo largo de todo el genoma. Luego, asigna estos cambios a los elementos genéticos ya conocidos (en el caso de que los hubiese). En caso de

[30] Sitio web de la aplicación **TilingScan**: http://tilingscan.uv.es

encontrar nuevas regiones no anotadas en el genoma, para anotarles, se desarrolló la aplicación **Annotate**[31] (**Figura 3.4**).

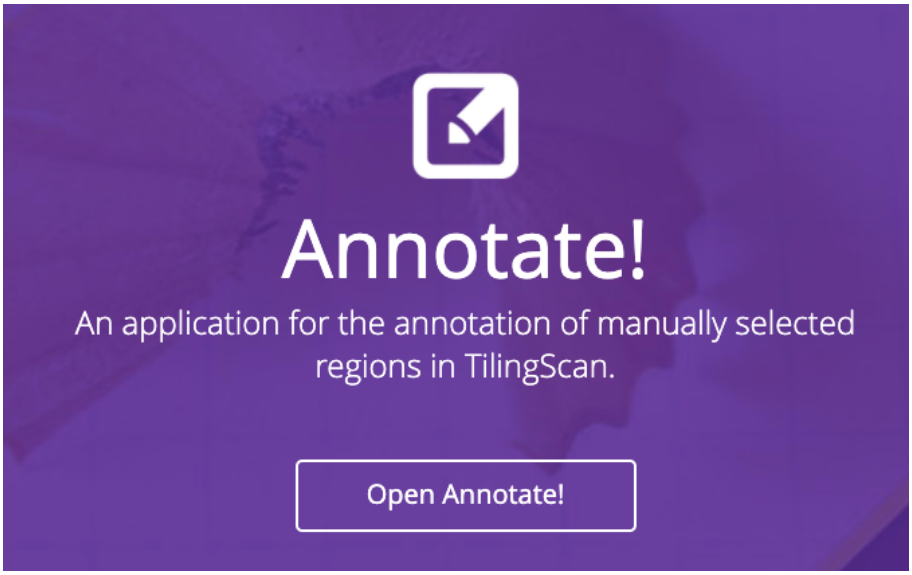


Figura 3.4. *Página de inicio de la aplicación* ***Annotate****, destinada a la anotación de las regiones de interés detectadas en* ***TilingScan****.*

Además, dado que las señales de datos de *microarrays* a menudo muestran perfiles ruidosos, **TilingScan** incorpora un algoritmo de suavizado basado en un filtro gaussiano, que elimina el ruido de la señal antes de realizar la búsqueda, permitiendo una visualización más clara de los datos. La aplicación también puede ser utilizada para analizar datos procedentes de experimentos de NGS. Para ello, se desarrolló el programa **cover2tiling**, que permite convertir los

[31] Aplicación **Annotate**: http://jmjuanes.github.io/annotate/

archivos BAM procedentes de este tipo de experimentos en archivos compatibles con **TilingScan**.

3.1.2 **Resultados**

Procesado de datos

Datos de entrada

Generalmente, un experimento de *tiling array* requiere de tres tipos de ficheros: uno correspondiente a la anotación de los genes y otros dos que contienen los valores de las sondas (uno para la hebra Watson y otro para la hebra Crick).

Fichero de anotación

El fichero de anotación contiene la información de todos los genes de un organismo específico. Para cada gen, proporciona información sobre el cromosoma y hebra en el que se encuentra, posición y nombre, entre otros. También puede contener información sobre exones o UTRs (regiones no traducidas de los genes), entre otros. El formato más utilizado para los ficheros de anotación es el formato **GFF** (*General Feature Format*). Pese a que existen varias versiones del formato GFF, esta aplicación solo acepta la versión más reciente (GFF3). En la **Figura 3.5** se presenta una captura de las primeras líneas de un archivo con este formato.

```
##gff-version   3
# File name: C_albicans_SC5314_version_A21-s02-m03-r06_features.gff
# Organism: Candida albicans SC5314
# Genome version: A21-s02-m03-r06
# Date created: Sun Sep 16 07:01:10 2012
# Created by: The Candida Genome Database (http://www.candidagenome.org/)
# Contact Email: candida-curator AT lists DOT stanford DOT edu
# Funding: NIDCR at US NIH, grant number 1-R01-DE015873-01
#
chr1    CGD     chromosome      1       3188548 .       .       .       "ID=Ca21chr1_C_albicans
chr2    CGD     chromosome      1       2232035 .       .       .       "ID=Ca21chr2_C_albicans
chr3    CGD     chromosome      1       1799406 .       .       .       "ID=Ca21chr3_C_albicans
chr4    CGD     chromosome      1       1603443 .       .       .       "ID=Ca21chr4_C_albicans
chr5    CGD     chromosome      1       1190928 .       .       .       "ID=Ca21chr5_C_albicans
chr6    CGD     chromosome      1       1033530 .       .       .       "ID=Ca21chr6_C_albicans
chr7    CGD     chromosome      1       949616  .       .       .       "ID=Ca21chr7_C_albicans
chrR    CGD     chromosome      1       2286389 .       .       .       "ID=Ca21chrR_C_albicans
mtDNA   CGD     chromosome      1       40420   .       .       .       "ID=Ca19-mtDNA;Name=Ca1
chr6    CGD     ORF     1028818 1031331 .       +       .       "ID=orf19.2163;Name=orf19.2163;
chr3    CGD     ORF     390312  390578  .       -       .       "ID=orf19.1667.1;Name=orf19.166
chr2    CGD     ORF     1995004 1997964 .       -       .       "ID=orf19.1373;Name=orf19.1373;
chr1    CGD     ORF     1141123 1142880 .       -       .       "ID=orf19.419;Name=orf19.419;No
chr4    CGD     ORF     240313  240615  .       +       .       "ID=orf19.4672;Name=orf19.4672;
```

Figura 3.5. *Captura de las primeras líneas de un fichero **GFF3** para «Candida albicans». Las primeras 9 líneas del fichero corresponden a la parte de la cabecera, mientras que el resto de líneas contienen los elementos del genoma de «Candida albicans», uno por cada línea.*

Un archivo GFF se divide en dos bloques. La primera parte del archivo GFF se conoce como la **cabecera,** y suele contener información extra como por ejemplo la versión del archivo GFF, el organismo o el autor, entre otros. La cabecera se diferencia del resto de contenido del archivo porque cada línea empieza un carácter «**#**».

El segundo bloque del archivo GFF contiene una línea para cada uno de los elementos del genoma, ya sean genes, exones, UTR, etc... Cada una de estas líneas contienen los siguientes 9 campos separados por tabulador:

1. **Localización**: cromosoma en el que se encuentra el elemento.
2. **Fuente**: nombre de la base de datos o programa con el que se ha generado.
3. **Tipo**: denota el tipo del elemento en cuestión (ORF, gen, exón,

etc).

4. **Posición** inicial: posición de inicio del elemento.
5. **Posición** final: posición en la que termina el elemento.
6. **Puntuación**: puntuación del elemento.
7. **Hebra**: hebra en la que se encuentra el elemento. Se representa con un $(+)$ para indicar que el elemento se encuentra en la hebra positiva o directa, y con un $(-)$ para indicar que se encuentra en la hebra negativa o inversa.
8. **Fase**: indica la fase donde el elemento empieza con referencia al marco de lectura. Este campo sólo contendrá un valor en elementos que sean del tipo *Coding DNA Sequence*, mientras que para el resto de elementos este campo estará vacío.
9. **Atributos**: listado de atributos del elemento, separados cada uno de ellos por un punto y coma.

Ficheros experimentales de expresión de sondas

Los ficheros experimentales se construyen a partir de la información obtenida de los microchips de GenChip Tiling Array de Affymetrix. Como el objetivo es buscar regiones que tengan una expresión diferencial entre dos condiciones, generalmente entre un caso control y un caso problema (muestra que ha sido sometida a un estrés o mutante), estos ficheros deberán contener la posición en la que se encuentra cada sonda junto con su valor de señal total. La señal total se puede calcular aplicando la siguiente fórmula:

$$Señal\ total = \frac{Señal\ caso\ problema}{Señal\ caso\ control}$$

En la **Figura 3.6** se puede observar una captura de las primeras líneas de un archivo que contiene esta información, el cual sigue el siguiente formato:

- **Cabecera del fichero**, en la que se muestra información relativa al contenido de dicho fichero. Cada una de las líneas de la cabecera debe empezar con un carácter «#», y el contenido de cada una de las líneas debe seguir el formato de propiedad y valor, separados por un carácter tabulador. Como mínimo la cabecera debe incluir dos líneas, en la que se debe indicar:
    - El número de secuencias contenidas en el archivo, especificado con la propiedad *Number of sequences*.
    - La hebra sobre la que proceden los datos, que se especifica con la propiedad *Signal* y solo puede tomar los valores *Forward* para la hebra positiva o *Reverse* para la hebra negativa.
- **Bloques de secuencias**, tantos como se hayan especificado en la cabecera del fichero. Cada bloque de secuencias representa la señal de un cromosoma entero. Cada una de estos bloques se puede dividir a su vez en dos secciones:
    - **Cabecera del bloque de secuencias**, que almacena información relativa a la secuencia. Al igual que la cabecera del fichero, cada una línea comienza con un carácter «#» y sigue el mismo formato de propiedad y valor. Las siguientes propiedades son obligatorias:

- Índice de la secuencia en el fichero, que debe ser especificado con la propiedad *Sequence*.
- Nombre de la secuencia, que normalmente coincide con el nombre del cromosoma. Este valor debe ser especificado en la propiedad *Name*.
- Número de sondas contenidas en la secuencia, que se especifica en la propiedad *Number of Hits.*

- **Listado de sondas**, una por línea y tantas como se hayan indicado en la propiedad *Number of Hits* de la cabecera de la secuencia. Cada una de estas líneas deberá contener la posición de la sonda, así como el valor de la señal total de la sonda en esa posición, separados por un tabulador.

```
# Number Sequences      143
# Signal        Forward

# Sequence      1
# Name  TagA
# Number of Hits        11

33      5.40000
76      1.00000
80      1.00000
125     4.20000
156     1.00000
276     0.25050
300     0.21739
307     0.23212
370     0.42258
379     0.42258
420     4.87442
```

Figura 3.6. *Fichero experimental, compuesto por dos líneas de cabecera, y un bloque con la relación sonda-señal para cada cromosoma.*

Este formato es similar al que se puede extraer si se utiliza el software proporcionado por Affymetrix para sus microarrays, cuya única diferencia son varias líneas adicionales en la cabecera de cada secuencia.

Ficheros BAM

La tecnología de los microarrays está siendo desplazada por otros tipos de tecnologías más modernas, tales como *RNA-Seq*, que harán que en un futuro no muy lejano dejen de utilizarse totalmente. Muchas de las nuevas herramientas de NGS utilizan ficheros de tipo SAM y BAM para guardar las secuencias de alineamientos. El formato SAM contiene las secuencias delimitadas por tabuladores, mientras que el formato BAM es la versión binaria y comprimida del SAM. Para poder manejar los alineamientos en este tipo de formatos, se utiliza el software SAMTOOLS (Li *et al*., 2009).

Sin embargo, estos tipos de archivos suelen tener un tamaño excesivamente grande como para ser subidos a un servidor, por lo que no es viable hacer que la aplicación acepte este tipo de formatos. Así pues, para poder solucionar esto se desarrolló una aplicación en C++ que, junto con SAMTOOLS, permite generar un formato más reducido y compatible con esta aplicación. Este software, denominado **cover2tiling**, está distribuido bajo licencia MIT de código libre y disponible en *GitHub*[32].

[32] Repositorio de **cover2tiling**: https://github.com/TilingScan/cover2tiling

Conversor de BAM a ficheros experimentales

Para poder convertir el fichero BAM a un formato compatible con la aplicación **TilingScan** primero el usuario deberá obtener el fichero de coberturas en formato *PILEUP* con el programa SAMTOOLS ejecutando el siguiente código:

```
$ samtools mpileup samples.bam > cover.txt
```

Con este comando se obtiene un fichero de texto tabulado llamado *cover.txt,* que contiene la cobertura de las lecturas en cada base (**Figura 3.7**).

```
gi|37552371|ref|NT_011255.14|   1   T   5   TTTTT   2JJ22
gi|37552371|ref|NT_011255.14|   2   T   4   TTTT    22GH
gi|37552371|ref|NT_011255.14|   3   C   6   CCCCCC  22H22$
gi|37552371|ref|NT_011255.14|   4   T   5   TTTAT   222G@
gi|37552371|ref|NT_011255.14|   5   A   5   AAAAA   @@@22
gi|37552371|ref|NT_011255.14|   6   A   6   AAAAAT  22@@@
gi|37552371|ref|NT_011255.14|   7   C   3   CCC 222
gi|37552371|ref|NT_011255.14|   8   A   5   AACAA   2JJG2
gi|37552371|ref|NT_011255.14|   9   G   5   GCGGG   2GG22
gi|37552371|ref|NT_011255.14|   10  C   7   TCCCCCC J22GG22
gi|37552371|ref|NT_011255.14|   11  A   5   AAAAA   22222
gi|37552371|ref|NT_011255.14|   12  G   5   GGGGT   22H22
gi|37552371|ref|NT_011255.14|   13  A   3   AAA 22@
gi|37552371|ref|NT_011255.14|   14  G   5   GGGGG   22222
gi|37552371|ref|NT_011255.14|   15  C   5   CCCCC   22@22
gi|37552371|ref|NT_011255.14|   16  T   4   TATT    22GG
gi|37552371|ref|NT_011255.14|   17  G   2   GG  HJ
gi|37552371|ref|NT_011255.14|   18  G   1   G   J
```

Figura 3.7. *Primeras líneas de un archivo con formato* ***pileup****. Cada línea está formada por 6 campos separados por un tabulador: nombre de la secuencia, posición del nucleótido, base de ese nucleótido en la referencia, bases de ese nucleótido en las lecturas, y calidad de las bases.*

Este archivo de coberturas puede ser convertido al formato compatible con **TilingScan** mediante el programa **cover2tiling** ejecutando el siguiente comando:

```
$ cover2tiling cover.txt tiling-output.txt N
```

Donde:

- *cover.txt* es el archivo con la cobertura obtenido en el paso

anterior.

- *tiling-output.txt* es el nombre del archivo que generará el programa **cover2tiling**.
- *N* es el número de nucleótidos a agrupar.

El programa **cover2tiling** simula un experimento con microarrays utilizando las coberturas del archivo *cover.txt* (**Figura 3.8**). Para ello, cada sonda se obtiene realiza grupos de N nucleótidos y calcula su expresión sumando los valores de coberturas de los nucleótidos del grupo y dividiéndolo entre N:

$$expresión = \frac{\sum_{i=1}^{N} cobertura(i)}{N}$$

Además, el programa también detecta los cambios de cromosoma, para así poder separar cada uno de ellos en un bloque independiente.

```
# File generated with Cover2Tiling
# http://github.com/TilingScan/cover2tiling

# Sequence 1
# Name  gi|37552371|ref|NT_011255.14|
# Number Of Hits    6

2   5
5   5.33333
8   4.33333
11  5.66667
14  4.33333
17  2.33333

```

Figura 3.8. *Primeras líneas del archivo generado por* ***cover2tiling****. Este archivo tiene el formato adecuado para ser utilizado en la aplicación web.*

Algoritmo de búsqueda de regiones significativas

Uno de los requisitos de la aplicación era la detección automática de las zonas susceptibles de sufrir expresión diferencial en levadura. Para ello se desarrolló un algoritmo de detección automática de regiones significativas, que está basado en una versión modificada del algoritmo de búsqueda mediante ventanas deslizantes (Basseville & Nikiforof, 1993). Como idea general, el algoritmo calcula el valor promedio de la expresión de un conjunto de sondas (definidas en el algoritmo como ventana), y la compara con el valor promedio del siguiente conjunto, de forma que, cuando la diferencia sea superior a un valor dado, lo considera como el inicio de una región significativa (**Figura 3.9**).

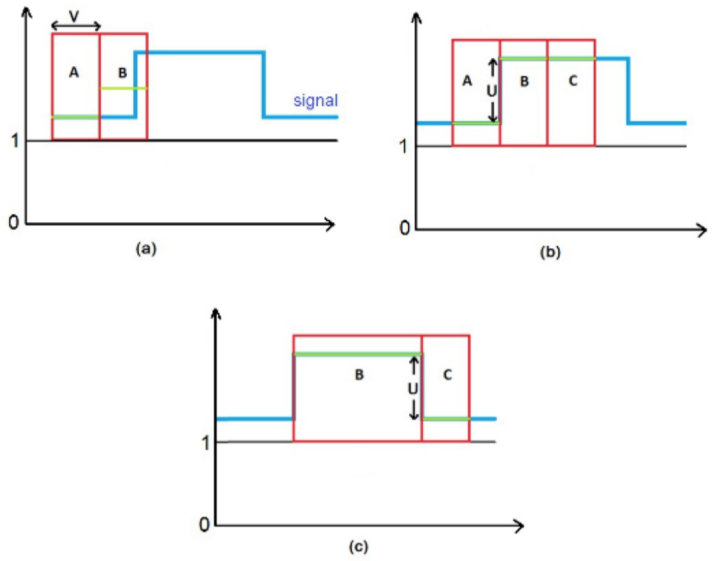


Figura 3.9. *Representación gráfica del algoritmo de detección por medio de ventanas deslizantes de tamaño V.* ***a)*** *Las dos ventanas A y B (en rojo) se desplazan calculando el promedio de la expresión (en verde).* ***b)*** *Cuando la diferencia de ambos promedios es superior a un valor umbral U, se marca como inicio de la región y se crea una nueva ventana C.* ***c)*** *La ventana B se extiende mientras que la ventana C se desplaza, hasta que la diferencia de los promedios de estas dos ventanas sea superior al valor umbral U, momento en el cual se marca la región como finalizada.*

Algoritmos para la representación gráfica

Filtro de Gauss

El filtro de Gauss permite eliminar el ruido de fondo que puede estar presente en los datos procedentes de experimentos con *microarrays*, contiene una variabilidad (ruido) de fondo debido a los sesgos y

posibles fluctuaciones en el ensayo de la hibridación o el escaneo de las sondas.

Para minimizar ese ruido y obtener valores más suavizados que no entorpezcan los algoritmos de búsqueda de señales se realiza un procesamiento inicial a la señal a la que se le aplica un filtrado de Gauss.

El **filtrado de Gauss** es una técnica utilizada para eliminar el ruido presente en una imagen, utilizando una función gaussiana. La fórmula de la función gaussiana de una dimensión es la siguiente (Haddad & Akansu, 1991):

$$g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Donde μ es la media y σ la varianza. En la **Figura 3.10** se muestra la gráfica de esta función para algunos valores de μ y σ.

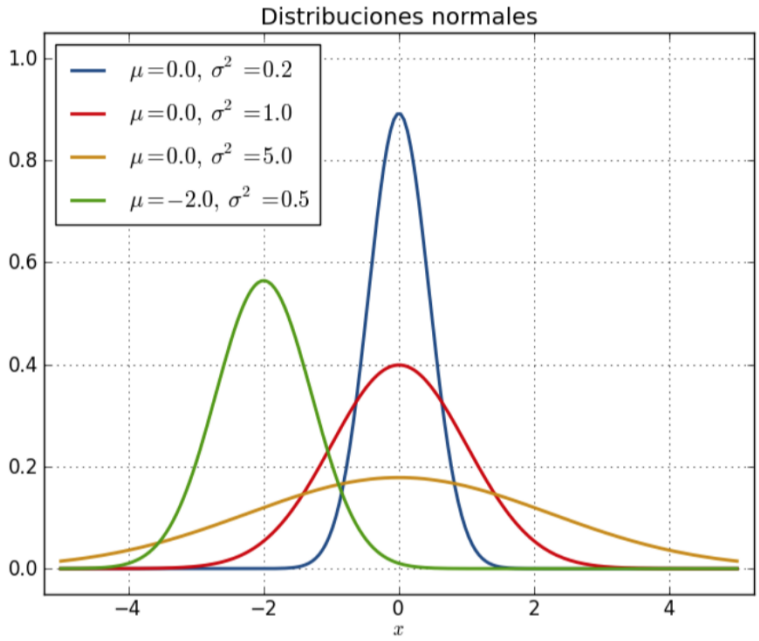


Figura 3.10. *Gráfica de la función gaussiana de una dimensión, para ciertos valores de μ y σ. Se trata de una función simétrica respecto de $x = \mu$ y cuya área es exactamente 1.*

Esta aplicación implementa un filtrado de Gauss de siete coeficientes con los valores por defecto de $\mu = 0$ y $\sigma = 1$, y los valores de $x \in \{-3, -2, -1, 0, 1, 2, 3\}$, por lo que los coeficientes que se aplican sobre cada valor de la señal son los siguientes:

0.006	0.061	0.242	0.383	0.242	0.061	0.006

Aplicando este filtro se obtiene una gráfica suavizada uniformemente, la cual suele producir una sensación de mayor estabilidad y ayuda a

detectar visualmente los patrones relevantes sin que la mente del usuario tenga que perder atención procesando picos procedentes del ruido. La aplicación permite especificar el número de veces que se desea aplicar este filtro, aunque se sugiere aplicarlo 3 veces, ya que por debajo de este valor no se consigue eliminar suficientemente todo el ruido, y por encima se suaviza demasiado la señal, lo que puede llevar a una pérdida de información.

Visualización de la expresión

Para explorar gráficamente los niveles de expresión contenidos en los ficheros experimentales, se ha desarrollado un sistema de visualización que genera unas gráficas en la que se representa la posición de la sonda frente a la señal de dicha sonda. De esta forma, en el eje horizontal se sitúa la posición de las sondas en nucleótidos, que para el caso de los chips de Affymetrix serán en múltiplos de 12 bases, ya que las sondas se solapan cada 12 nucleótidos (**Figura 3.2**).

Por otro lado, en el eje vertical se sitúa la señal o ratio de intensidad de cada una de las sondas, definidas como *señal total*. Esa señal, extraída del fichero experimental para cada una de las hebras, viene como un cociente de dos señales (caso estrés contra caso control). Por lo tanto, se cumple que:

- Los valores de señal total superiores a 1 se corresponden con valores de señal del caso problema superiores al caso control, por lo que dicha región se está expresando más en el caso problema que en el caso control.
- Por el contrario, valores de señal entre 0 y 1 significan que la señal en el caso control es superior al del caso problema, por

lo que la expresión de dicha región está reprimida en el caso problema.

Pero trabajar con ratios conlleva a errores de percepción visuales y cognitivas (**Figura 3.11**), ya que la zona de sobreexpresión (valores superiores a 1) no es simétrica con respecto a la zona de subexpresión (que va de 1 a 0). La sobreexpresión es lineal hasta infinito pero la subexpresión es asintótica a 0.

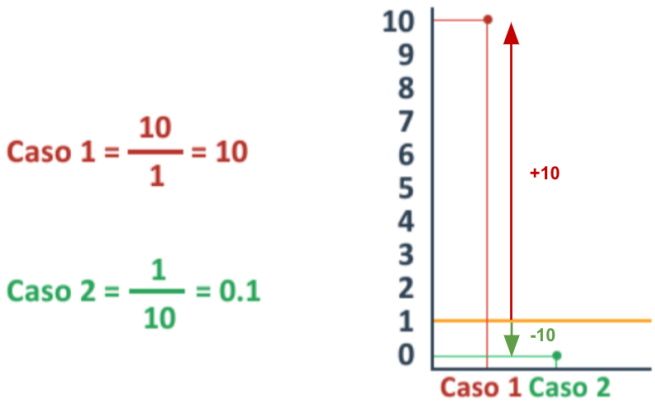


Figura 3.11. *Representación de la expresión. Para el Caso 1, si la señal en el caso estrés es diez veces superior al del caso control, el cociente de ambas señales será 10. En el Caso 2, si la señal en el caso control es diez veces superior al del caso estrés, el cociente será 0.1. Si representamos ambos valores, dicha representación puede llevar al usuario a pensar que la expresión del caso estrés en el Caso 1 es muchísimo mayor que en caso control para el Caso 2, cuando en verdad la diferencia es la misma.*

Por ello, para conseguir una simetría visual de los valores de sobre y subexpresión, se aplica una transformación logarítmica a la ratio y así

evitar este tipo de errores. De esta forma, la gráfica pasará a estar centrada en el 0, y tanto la zona por encima como la zona por debajo serán equitativas. En la **Figura 3.12** se representa el mismo ejemplo de la figura anterior, utilizando el logaritmo en base 2.

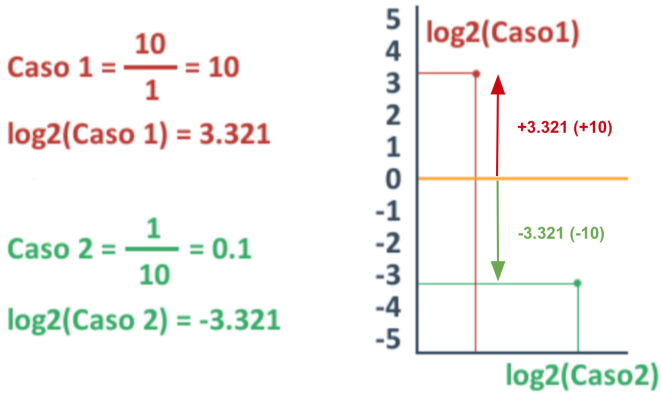


Figura 3.12. *Ejemplo de representación del logaritmo de la expresión, manteniendo la simetría en el 0 en ambos casos.*

3.1.3 **Arquitectura de la aplicación**

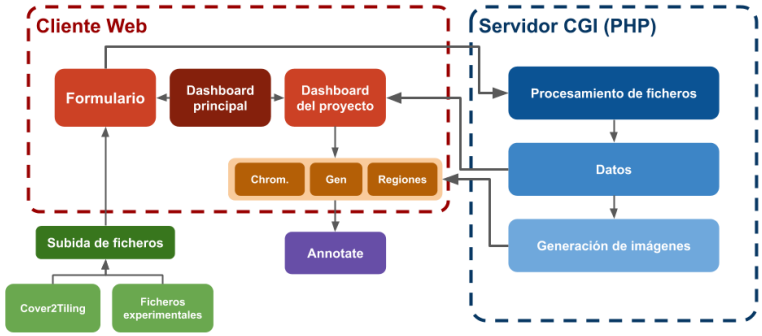


Figura 3.13. *Esquema de la arquitectura de la aplicación* ***TilingScan****.*

La aplicación ha sido implementada en *PHP* (*back end* o lado del servidor) y *JavaScript* (*front end* o lado del cliente), siguiendo una arquitectura clásica **MPA**. El código fuente, que se distribuye bajo licencia **MIT**, está accesible en un repositorio de *GitHub*[33].

El lado del servidor se basa en la tecnología web con Interfaz de Entrada Común (*Common Gateway Interface*, CGI), en la actualidad reemplazada por tecnologías basadas en servicios REST, y en el uso de PHP. El lado del servidor es donde está la lógica y los algoritmos de la aplicación para la detección de las regiones con señal de expresión diferencial, y donde se generan los gráficos con las curvas de expresión a lo largo del genoma (**Figura 3.13**).

[33] Repositorio de **TilingScan**: https://github.com/TilingScan

En el lado del cliente web, el sistema de visualización interactiva de las regiones detectadas se desarrolló en JavaScript, utilizando la tecnología *Canvas* de HTML5.

El servicio desarrollado consta de:

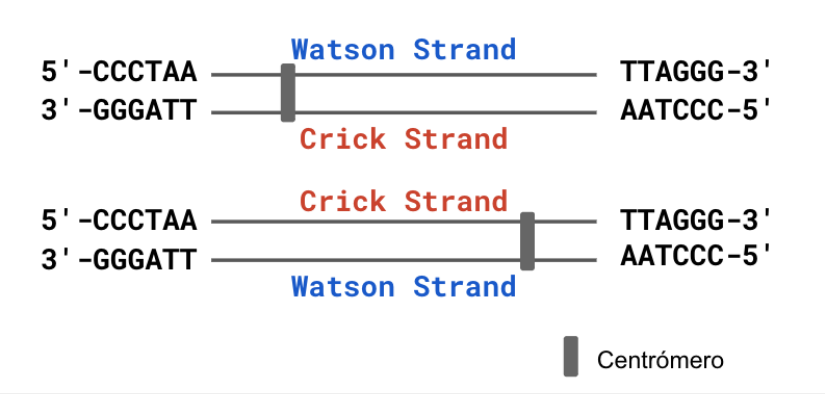
- Un software de preparación de datos en el lado del cliente.
- Un sistema en la aplicación web, que permite subir al servidor los ficheros a procesar.
- Un programa con algoritmos de análisis de expresión diferencial en el lado servidor.
- Un sistema de generación de los gráficos de nivel de expresión, también en el lado servidor.

Este servicio no necesita de un sistema de colas dedicado, ni de sistemas de gestión para el almacenamiento seguro, ni de control de acceso. Los datos depositados se borran a los 60 días, y la única forma de restringir el acceso es añadiendo una cadena aleatoria o compleja al final del nombre del proyecto, ya que cualquier usuario puede acceder a cualquier proyecto del que conozca el ID. Este servicio no almacena datos sensibles y, por tanto, no se profundizó en las complejidades de los sistemas de autenticación en esta aplicación.

La aplicación también incorpora un tutorial y un conjunto de datos de prueba, con los que el usuario puede explorar su funcionamiento antes de empezar a trabajar con sus propios datos.

Como nota de nomenclatura, cabe mencionar que, dentro de la comunidad del estudio de levaduras, es costumbre nombrar las hebras

de ADN de referencia y complementaria como *Watson* y *Crick*, respectivamente (**Cuadro 3.1.1**).



La base de datos *Saccharomyces Genome Database* usa la nomenclatura «hebra de Watson» y «hebra de Crick» (*Watson-strand* y *Crick-strand*) para asignar un sufijo a cada gen en su nomenclatura sistemática basada en su posición el en genoma de S. cerevisiae (Cherry *et al.,* 1998).

Definición estandarizada de las hebras de Watson y Crick.

En sentido estricto, cuando se usa una referencia genómica, se le asigna la etiqueta Watson a la hebra que tiene el 5' en el brazo corto del cromosoma, y la hebra de Crick a la que tiene el extremo 3' en ese brazo corto. En sentido laxo, como por convención las referencias genómicas empiezan a contar las bases desde el inicio del brazo corto y se usa como referencia la hebra 5' → 3', en la práctica se dice que la hebra de referencia o (+) es la hebra de *Watson* y la complementaria o (−) es la hebra de *Crick*.

Fuera del ámbito de los círculos genómicos de levaduras, esta nomenclatura está en desuso. En levadura aún se usa porque la nomenclatura estandarizada de los genes añade una última letra «W» o «C» al gen para decir si está en el sentido de la referencia o en la complementaria. Por ejemplo, el gen alcohol deshidrogenasa I (ADH1) tiene asignado el nombre sistemático YOL086C, donde la última C indica que está localizado en la hebra complementaria (Cartwright y Graur, 2011).

Cuadro 3.1.1. *Nomenclatura de las hebras del genoma (**Watson** y **Crick**) en levaduras*

3.1.4 **Flujo de uso de la aplicación**

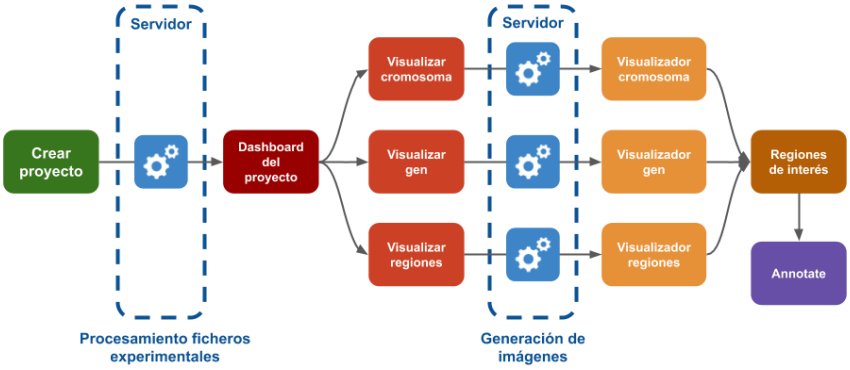


Figura 3.14. *Flujo de trabajo de la aplicación **TilingScan**.*

La aplicación **TilingScan** está compuesta por una serie de vistas conectadas, cuyo flujo se puede observar en la **Figura 3.14**. La

estructura y composición de pantallas de la aplicación (**Figura 3.15**) se divide en las siguientes partes:

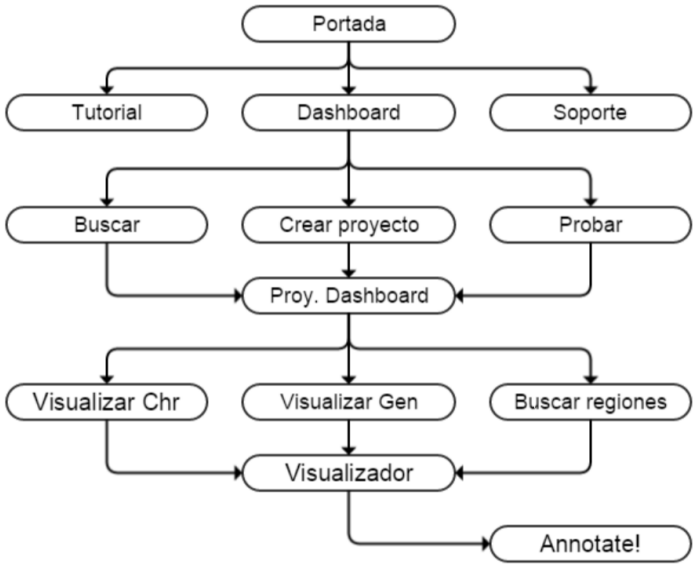


***Figura 3.15**. Diagrama que representa las diferentes vistas que componen la aplicación y la relación entre cada una de ellas*

- **Portada, tutorial y soporte**: la portada es la primera vista que vería el usuario al conectarse a la aplicación, y a través de la cual se puede acceder al tutorial y a la página de soporte, además de al cuadro de mandos principal o *dashboard* (**Figura 3.17***)*.
- **«Dashboard» o cuadro de mandos principal**: esta vista se puede considerar el centro de la aplicación, desde la cual se puede crear un nuevo proyecto, buscar alguno existente o probar la aplicación con unos datos de ejemplo (**Figura 3.19**).

- **«Dashboard del proyecto», o cuadro de mandos del proyecto**: cada proyecto dispone de su propio cuadro de mandos desde el que ejecutar cualquiera de las tres utilidades que dispone la aplicación: visualizar un cromosoma entero, visualizar un gen o buscar regiones significativas (**Figura 3.21**).
- **Visualizador**: es la vista en la cual el usuario explora los resultados que se obtienen tras ejecutar cualquiera de las tres utilidades de las que dispone la aplicación (**Figura 3.22**).
- **Visualizador de regiones**: es una vista especial, construida a partir de la vista anteriormente mencionada, que ofrece al usuario la posibilidad de explorar una región de interés definida de forma manual (**Figura 3.28**).
- **Annotate**: es una aplicación externa que facilita el registro de una serie de datos de interés del visualizador de regiones y que permite su posterior descarga como un fichero de texto tabulado (**Figura 3.30**).

La unidad de gestión principal de la aplicación son los proyectos, que no son más que contenedores para los ficheros experimentales y anotaciones que el usuario ha subido a la aplicación. Cada proyecto tiene asignado un identificador único, con el cual el usuario podrá acceder en el futuro a ese proyecto para su análisis.

Para facilitar la navegación por las diferentes vistas, la aplicación muestra en la parte superior de cada una de ellas una cabecera en la que se localizan los accesos a las diferentes vistas principales de la aplicación (**Figura 3.16**).

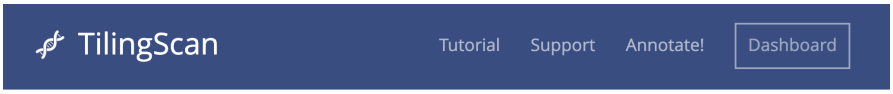


Figura 3.16. *Captura de la cabecera de la aplicación* ***TilingScan****, en la que se muestran enlaces a las vistas del tutorial, soporte, la aplicación* ***Annotate*** *y el cuadro de mandos principal, en ese orden. Además, la portada también es accesible a través del nombre de la aplicación.*

Portada, tutorial y soporte

La portada de la aplicación es la primera página que se mostrará cuando cualquier usuario acceda a la aplicación. En ella, además del título de la aplicación y una breve descripción de la misma, se muestran una serie de botones que dan acceso a diferentes vistas: acceso al cuadro de mandos principal, el tutorial de la aplicación y la vista de soporte (**Figura 3.17**).

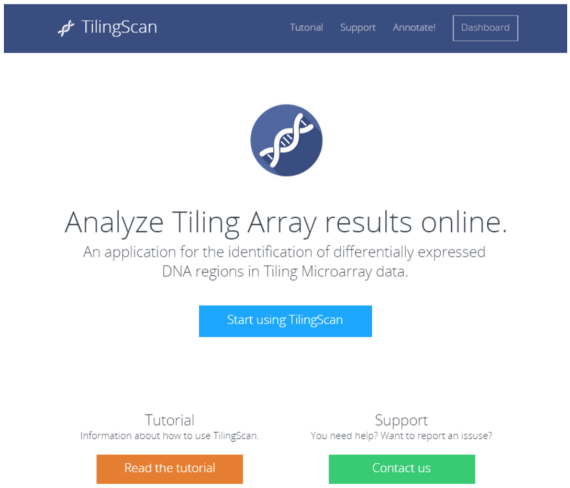


Figura 3.17. *Portada de* ***TilingScan****, que permite el acceso a todas las secciones principales de la aplicación.*

El tutorial de la aplicación proporciona una guía detallada de las secciones de la aplicación, así como la descripción de los ficheros necesarios y la explicación de los resultados que se obtienen. Además, proporciona unos enlaces para que se puedan descargar unos datos para poder probar la aplicación (**Figura 3.18**).

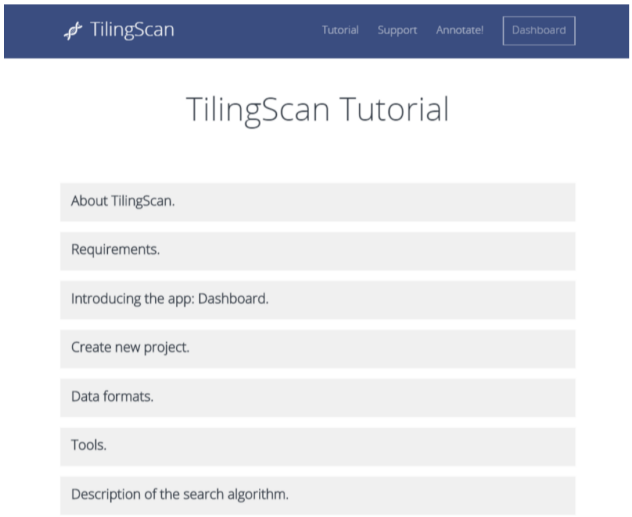


*Figura 3.18. Tutorial de **TilingScan**.*

Cuadro de mandos principal

El cuadro de mandos principal o «**Dashboard**» es el punto central de la aplicación, desde el cual se puede realizar una búsqueda de proyectos, crear un nuevo proyecto o probar la aplicación con un proyecto de prueba. La distribución de los elementos de esta vista se puede observar en la **Figura 3.19**.

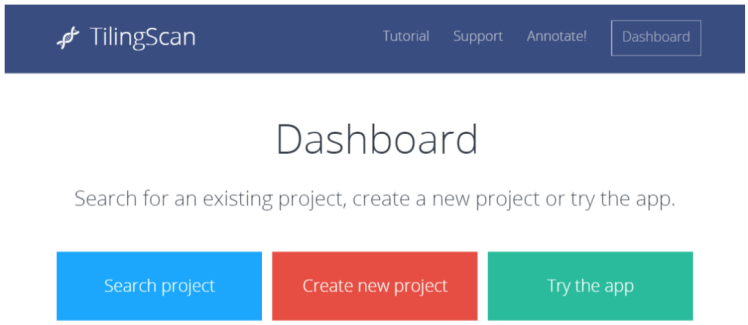


Figura 3.19. *Cuadro de mandos principal de* ***TilingScan****. El botón azul permite la búsqueda de un proyecto (por nombre, identificador o autor), el botón rojo permite crear un nuevo proyecto y el botón verde permite probar la aplicación utilizando un proyecto de prueba.*

Desde la vista de buscar se realizar una búsqueda por entre los proyectos que hayan sido creados en la aplicación. Para ello, se proporciona un cuadro de búsqueda mediante el cual la aplicación buscará entre los proyectos cuyo identificador, título, descripción o autor coincidan con las palabras que el usuario haya introducido en dicho cuadro de búsqueda.

Por otro lado, también se puede crear un nuevo proyecto, rellenando un sencillo formulario como el que se muestra en la **Figura 3.20** y en cual que se solicita:

- **Título** del proyecto a crear, mediante el cual se puede especificar la finalidad o procedencia de los datos del proyecto.
- **Descripción** del proyecto, en el que se puede proporcionar información más detallada del experimento.
- **Autor**: nombre del autor o usuario que va a subir los datos.
- **Fichero de anotación**: permite adjuntar el archivo GFF del

organismo.

- **Tipo de experimento**: permite especificar si el experimento es específico de hebra o no.
- **Ficheros experimentales**: permite adjuntar los archivos que contienen la información de la expresión de la hebra directa y de la hebra inversa. En el caso de que se especifique que el experimento no es específico de hebra, sólo se permite adjuntar un único fichero experimental.

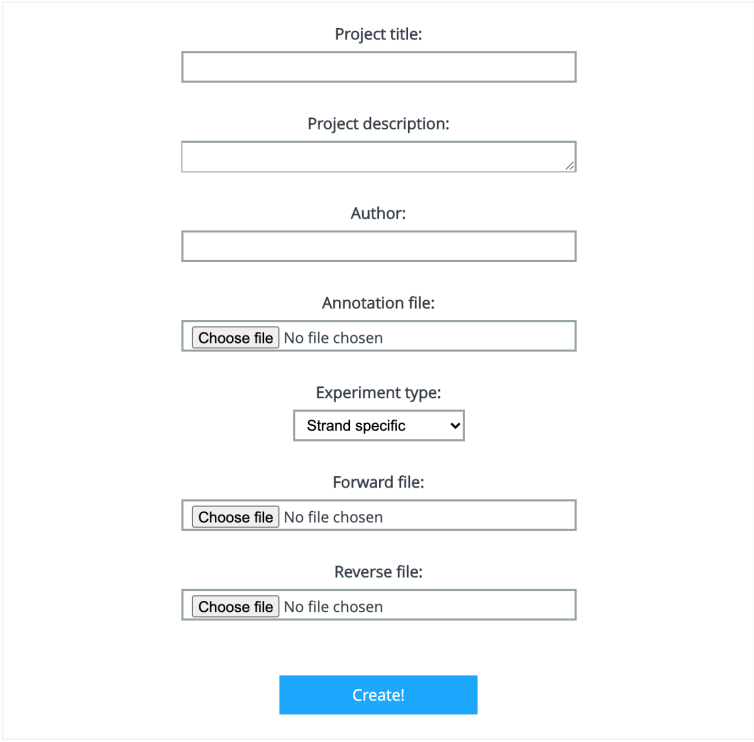


***Figura 3.20**. Formulario para la creación de un nuevo proyecto en **TilingScan**.*

Dado que los ficheros experimentales suelen tener un tamaño considerable (alrededor de los 50 MB), la aplicación también permite que se adjunten los archivos comprimidos en un ZIP (cada uno por separado), permitiendo así reducir el tiempo de subida de estos archivos al servidor. Una vez el formulario se haya completado y los ficheros se hayan subido al servidor, la aplicación asignará al proyecto un identificador único (*Project ID*), con el cual puede acceder al cuadro de mandos del proyecto.

Cuadro de mandos del proyecto

Cada proyecto creado en la aplicación dispone de su propio cuadro de mandos o ***Project Dashboard*.** Como se puede observar en la **Figura 3.21**, el cuadro de mandos del proyecto proporciona la información básica del proyecto (título, descripción, identificador, autor y fecha de expiración del proyecto), así como los accesos a las tres herramientas principales de la aplicación.

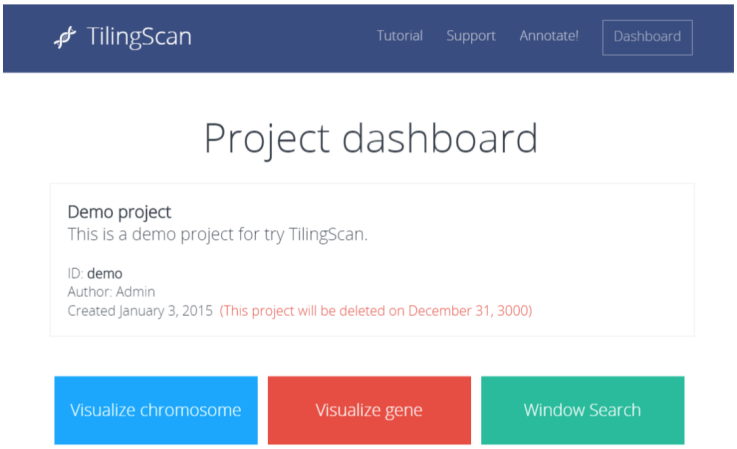


Figura 3.21. *Cuadro de mandos de un proyecto de* ***TilingScan****. Pulsando en los botones se muestran las opciones que permiten la ejecución de las diferentes herramientas de la aplicación.*

Herramienta de visualización de un cromosoma

Esta herramienta permite visualizar la expresión de un cromosoma completo. Las siguientes opciones deben ser proporcionadas para poder ejecutar esta herramienta:

- **Cromosoma**: permite seleccionar el cromosoma de interés por medio de un menú desplegable. La aplicación únicamente muestra en dicho menú los cromosomas que tengan expresión (es decir, los que estén presentes en los ficheros experimentales).
- **Filtro de Gauss**: permite establecer el número de veces que se aplicará el filtro de Gauss a los valores de expresión. El valor por defecto es 3.

La visualización de los valores de expresión del cromosoma especificado se realiza en una nueva vista, en la que se muestra de nuevo la información del proyecto, además de con un botón para volver al cuadro de mando del proyecto, y las gráficas de la expresión de todo el cromosoma (**Figura 3.22**).

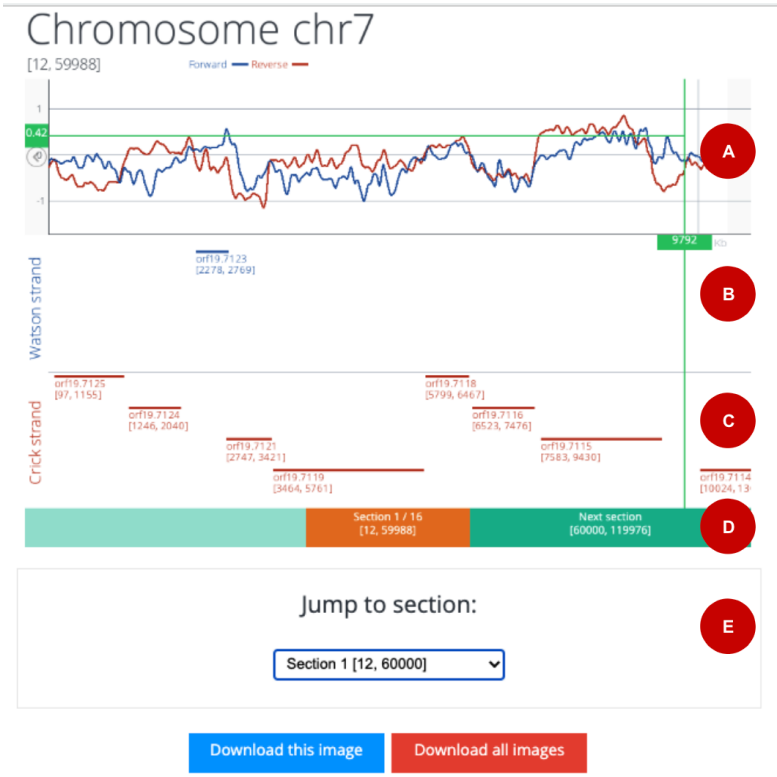


Figura 3.22. *Vista del visualizador de un cromosoma. En (A) se muestra la gráfica en la que está representada la posición de la sonda contra la señal en dicha sonda, para las hebras Watson (azul) y Crick (roja). En (B) se muestran los genes de la hebra Watson mientras que en (C) se muestran los genes de la hebra Crick. En (D) se muestra la información de la sección del cromosoma que se está visualizando, así como dos botones laterales que permiten cambiar a la anterior y a la siguiente sección. Por último, en (E) se muestra*

un menú desplegable que permite la navegación entre todas las secciones en las que ha sido fraccionado el cromosoma que se está visualizando.

Debido a las limitaciones técnicas que supone mostrar una única imagen para todo un cromosoma (por ejemplo, para el cromosoma 1 en los datos de prueba tendría un tamaño de 2.700.000 de píxeles), la aplicación fragmenta cada cromosoma en imágenes individuales de aproximadamente 50.000 píxeles. A cada una de estas imágenes se puede acceder consecutivamente mediante botones situados en la parte inferior del visor.

Herramienta de visualización de un gen

La aplicación también permite visualizar la expresión de una región específica del genoma. Las opciones que se deben proporcionar para poder utilizar esta herramienta son las siguientes:

- **Cromosoma**: permite especificar el cromosoma en el que se encuentra el gen de interés, por medio de un menú desplegable que contiene todos los cromosomas que tienen expresión.
- **Gen**: tras seleccionar el cromosoma, se debe seleccionar el gen por medio de un menú desplegable que contiene todos los genes que están en dicho cromosoma. Este menú se genera a partir de los genes disponibles en el fichero de anotación.
- **Margen**: permite establecer el número de sondas que se desean mostrar a ambos lados del gen. Cualquier otro gen que se encuentre dentro de este margen será también mostrado en la región, ampliándose el margen de forma automática para

que el gen quede totalmente incluido en la región.

- **Filtro de Gauss**: permite especificar el número de veces que se va a aplicar dicho filtro a los valores de expresión de la región.

La vista que se generará tras proporcionar estas opciones será similar a la mostrada en la **Figura 3.23**. Al igual que en la vista del cromosoma, también se mostrará la descripción del proyecto junto con un botón para volver al cuadro de mandos del proyecto.

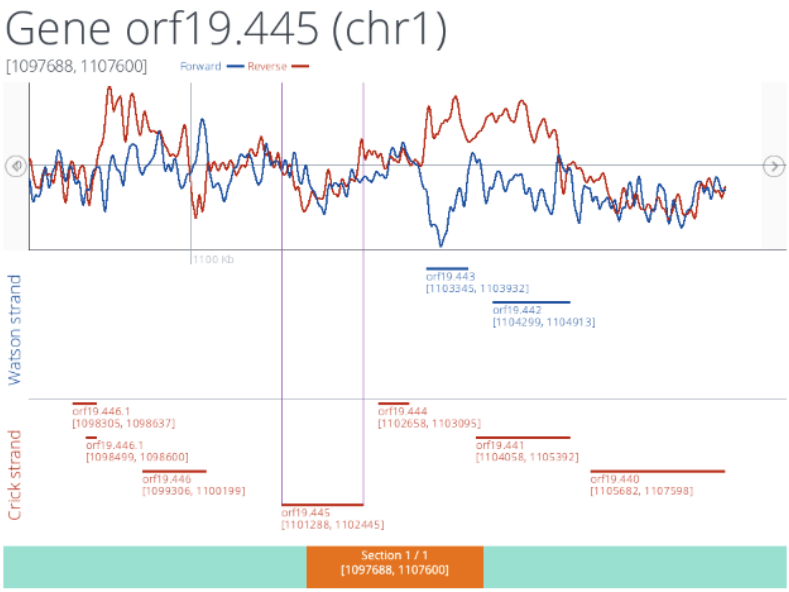


Figura 3.23. *Vista de visualización de un gen de interés. Las dos rectas verticales de color morado determinan la posición de inicio y final del gen que se ha seleccionado. Todos los genes que se encuentran dentro del margen también son representados en el gráfico.*

Herramienta de búsqueda de regiones significativas

La tercera herramienta de la aplicación permite ejecutar el algoritmo de búsqueda de regiones que tengan una expresión significativa. Al igual que con el resto de herramientas, se debe rellenar una serie de parámetros para poder ejecutar esta herramienta:

- **Cromosoma**: permite especificar el cromosoma en el que se va a aplicar el algoritmo de búsqueda de regiones.
- **Hebra**: permite especificar la hebra sobre la que se aplicará el algoritmo. En el caso de que el experimento no sea específico de hebra, esta opción no estará disponible.
- **Filtro de Gauss**: al igual que en las otras dos herramientas, permite especificar el número de veces que se aplicará este filtro para eliminar el ruido de fondo.
- **Tamaño de la ventana, umbral y porcentaje**: permiten establecer los valores de tamaño de la ventana, umbral y porcentaje, necesarios para poder aplicar el algoritmo de búsqueda de regiones.
- **Margen**: número de sondas que se desean mostrar a ambos lados de cada una de las regiones detectadas.

La vista que muestra las regiones detectadas tiene una estructura similar a la vista del resto de las herramientas (**Figura 3.24**). Para cada región detectada se genera una imagen individual, cuyo tamaño será el tamaño de la región junto con los márgenes. La posición de inicio y la de final de la región queda delimitada por dos líneas verticales en morado.

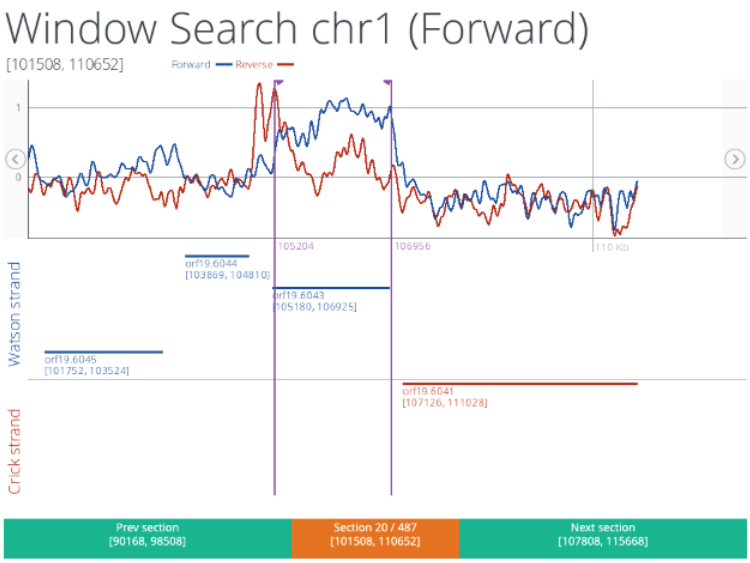


Figura 3.24. *Vista de una región detectada utilizando el algoritmo de búsqueda de regiones significativas.*

Esta vista también permite la generación de un archivo de texto en la que se recopila, además de los parámetros con los que se ha ejecutado el algoritmo de búsqueda de regiones, información básica de cada una de estas regiones (posición de inicio y final de la región y el promedio de las ventanas utilizadas en el algoritmo).

Visualizador de regiones

Las tres herramientas disponibles desde el cuadro de mandos del proyecto generan una serie de imágenes en formato PNG, las cuales pueden ser exploradas dentro de lo que en la aplicación se denomina el **visualizador de regiones**. Cada una de estas imágenes representa una región (ya sea una fracción de un cromosoma, un gen o una región

de interés), y está dividida en las siguientes tres partes:

- En la parte superior de la imagen se sitúa el nombre de la región, así como la posición inicial y final de la región y la leyenda de color. Dependiendo de la herramienta que haya generado la imagen, el título será diferente: por ejemplo, en la herramienta de exploración de un cromosoma, el nombre de la región será el nombre del cromosoma.
- En la parte central de la imagen se muestra la expresión en dicha región. En el eje X se sitúa la posición de cada una de las sondas, mientras que en el eje Y se representa la expresión asociada a dicha sonda. En el caso de que el experimento sea específico de hebra, se representarán dos líneas en dos colores: en azul correspondiente a la expresión en la hebra positiva, y en rojo la expresión en la hebra negativa. En el caso de que el experimento no sea específico de hebra, se mostrará una única línea.
- En la parte inferior de la imagen se distribuyen los genes y otros elementos genómicos de la región, siempre y cuando estén definidos en el fichero de anotación. Los elementos que se encuentren en la hebra positiva estarán con color azul y se sitúan en la mitad superior de esta parte, mientras que los elementos que están en la hebra negativa estarán con color rojo y se situarán en la mitad inferior de esta parte.

Para poder interaccionar estas imágenes, el visualizador añade una capa sobre cada una de ellas permitiendo que puedan ser desplazadas en ambos sentidos, navegar entre estas imágenes,

obtener información de los genes mostrados en dicha imagen o seleccionar un fragmento de dicha región para su exploración en más detalle. En la **Figura 3.25** se muestra una captura en la que se pueden observar las diferentes partes que lo componen.

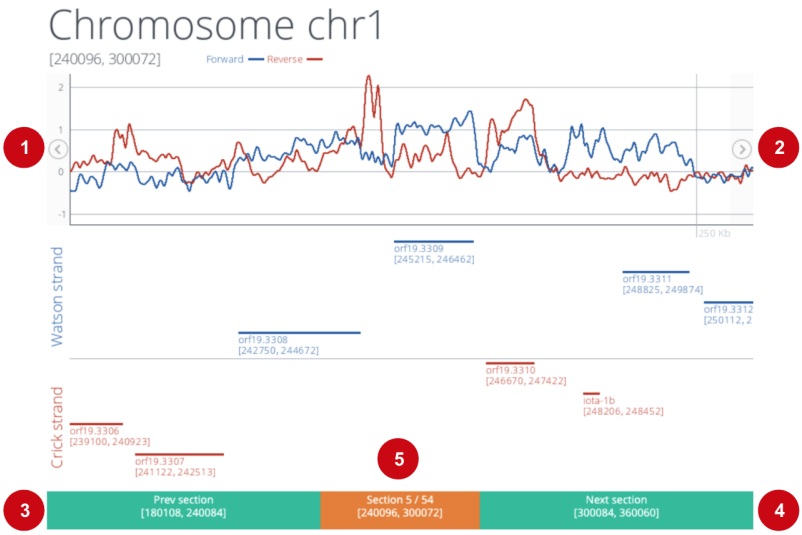


***Figura 3.25**. Estructura del visualizador de regiones. En **(1)** y **(2)** se localizan las áreas que permiten el desplazamiento a izquierda y a derecha del gráfico, respectivamente. Los botones **(3)** y **(4)** permiten acceder a la siguiente región disponible, mientras que el recuadro naranja en **(5)** muestra las coordenadas genómicas de la región representada en la imagen actual, así como el índice que ocupa dicha imagen en el listado de imágenes disponibles.*

Para facilitar la exploración de todas las imágenes, además de permitir acceder a la siguiente o a la anterior, el visualizador también dispone de un menú desplegable mediante el cual puede saltar a la visualización de una imagen específica, sin tener que buscarla de forma individual.

Por otro lado, la aplicación también incluye un sistema para mostrar el valor de la expresión, así como de la posición de la sonda, tan solo situando el ratón sobre el punto de interés. En la **Figura 3.26** se puede observar una captura de este comportamiento del visualizador de regiones.

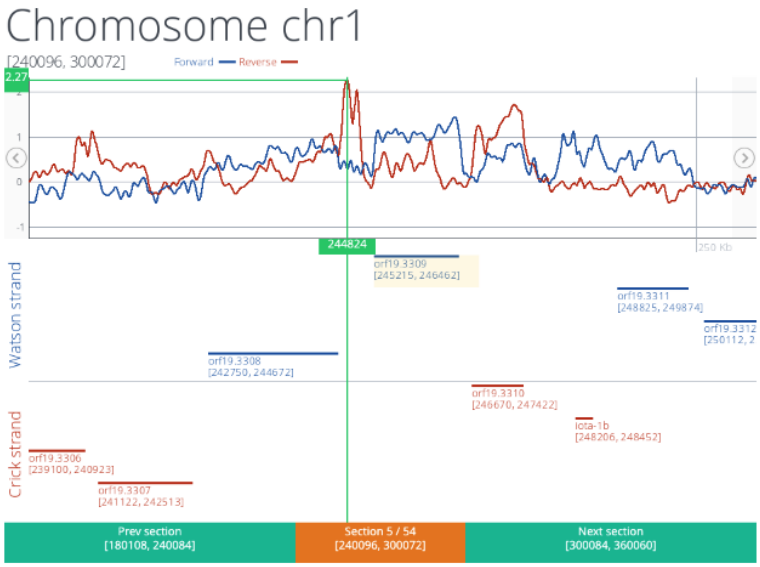


__Figura 3.26__. Región del cromosoma 1 en la que se ha situado el ratón sobre uno de los picos de la hebra negativa (rojo), de forma que en el eje horizontal se muestra en color verde la sonda en la que se encuentra, y en el eje vertical se muestra el valor de expresión en dicho punto

El visualizador de regiones también permite delimitar regiones que sean de interés para el investigador o usuario de la aplicación, de forma que puede explorarlas con más detalle de forma independiente a la región total (**Figura 3.27**). Para poder seleccionar una región de interés, basta delimitar los puntos de inicio y final pulsando con el

botón izquierdo del ratón: tras pulsarlo por primera vez, se añadirá una marca en naranja indicando el inicio de la región de interés justamente en la posición en la que se ha pulsado, mientras que si se vuelve a pulsar en otra posición diferente se añadirá una segunda marca y se marcará la región comprendida en color naranja. Si se pulsa de nuevo sobre el área naranja, se abrirá una nueva pestaña del navegador web con el visualizado de regiones con la región de interés que se ha delimitado. Pulsando sobre cualquier otro punto de la gráfica fuera del área naranja hará que se reinicie la región delimitada.

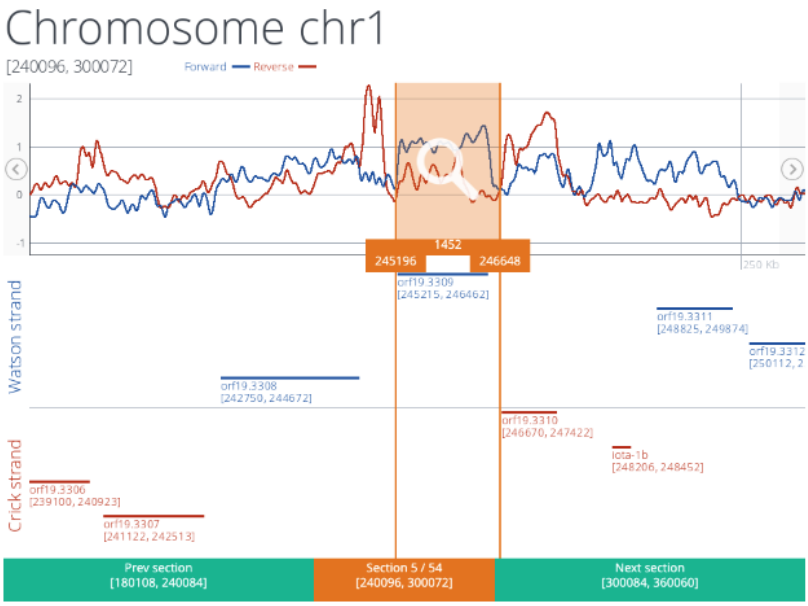


***Figura 3.27**. Región de interés delimitada por el área naranja en el cromosoma 1. Tras pulsar sobre dicha área naranja, se abrirá una nueva ventana en el navegador permitiendo explorar la región con más detalle. En la parte inferior se muestra la longitud de la región, mientras que a ambos lados se muestra la posición inicial y final de la región.*

Por último, destacar que el visualizador de regiones permite además que se puedan descargar todas las imágenes al ordenador del usuario, comprimidas en un fichero ZIP.

Visualizador de regiones de interés

El **visualizador de regiones de interés** funciona de forma similar al visualizador de regiones, con la diferencia de que permite realizar una exploración más detallada de una región delimitada (**Figura 3.28**). Este visualizador se ejecuta en una ventana diferente a la que se realiza la exploración de un cromosoma, gen o regiones expresadas significativamente.

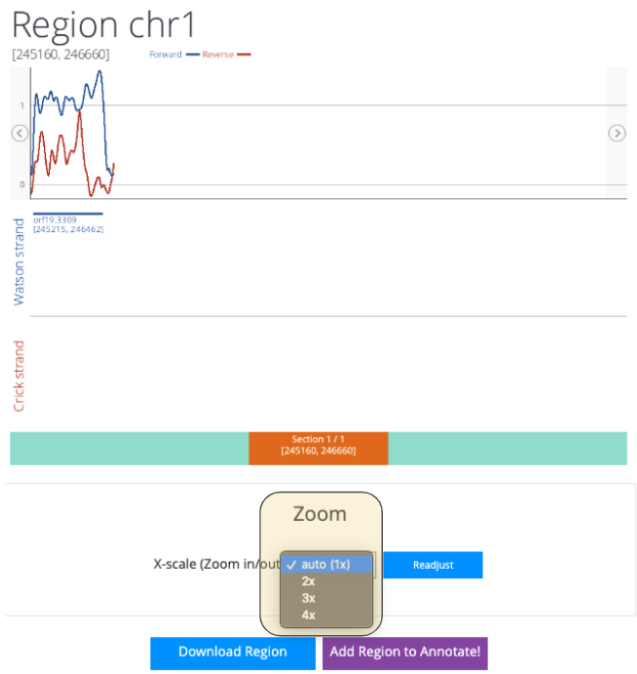


***Figura 3.28**. Vista detallada de una región de interés. En la parte inferior se puede escoger el nivel de zoom que se desea aplicar a la región (1x, 2x, 3x y 4x).*

La funcionalidad de este visualizador es la misma que el visualizador del apartado anterior, con la característica añadida de que se puede escoger un nivel de zoom para aumentar el detalle de la zona.

En la **Figura 3.28** se puede observar la vista inicial de una región de interés tras ser abierta con este visualizador, mientras que en la **Figura 3.29** se muestra la misma región, pero esta vez con un zoom mayor que la original (4 veces el tamaño de la región original).

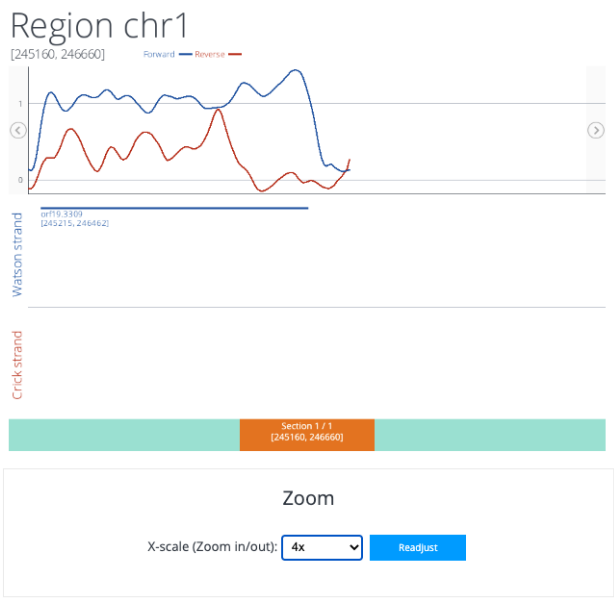


Figura 3.29. *Vista detallada de una región de interés a la que se le ha aplicado una amplificación de 4x.*

Aplicación para la anotación de regiones

La **aplicación para la notación de regiones de interés**, llamada **Annotate**, es una aplicación externa desarrollada totalmente en *JavaScript*, que permite la anotación de una serie de parámetros extraídos de las regiones que el usuario delimita de forma manual. Estos parámetros son:

- Nombre del proyecto.
- Cromosoma en el que se localiza la región delimitada.
- Longitud de la región delimitada.
- Coordenada de inicio de la región delimitada.

- Coordenada final de la región delimitada.
- Valor medio de intensidad de la región en la hebra directa.
- Valor medio de intensidad de la región en la hebra inversa.
- El nombre de las ORF o de los genes que hay en la región. Para cada uno se indica si se encuentran en la hebra directa (mediante una **W**) o en la hebra inversa (mediante una **C**). En el caso de que en la región no exista ningún gen u ORF, se mostrará el texto «NO ORF».

Toda esta información se guarda en una nueva ventana, cuya estructura es la que se muestra en la **Figura 3.30**. Los registros se organizan en una tabla, de forma que cada fila es un nuevo registro. **Annotate!** permite editar cada celda, así como eliminar una fila entera o añadir una nueva de forma manual.

El visualizador de regiones permite registrar una serie de datos de la región que se está explorando en esta herramienta de anotación.

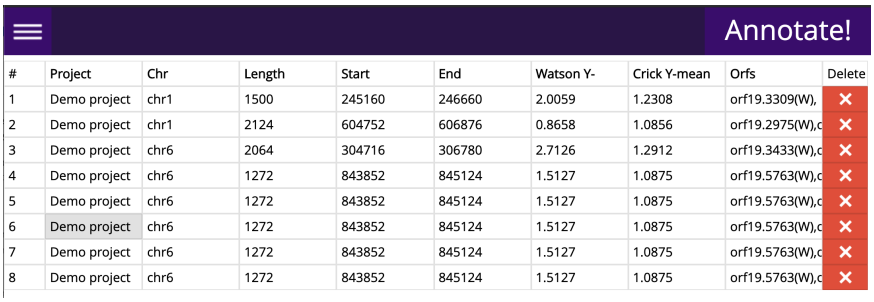


***Figura 3.30**. Aplicación **Annotate**. Cada registro (fila) en la aplicación corresponde a los datos de una región de interés. La aplicación permite la modificación de los campos de cada registro, así como la eliminación de los registros de forma individual.*

El contenido que se guarda en esta herramienta está solo disponible de forma local en el ordenador en el que se están realizando los análisis. Para poder conservarlos, se puede descargar un fichero tabular con toda la información que se haya guardado en **Annotate**.

3.2 VISMapper

Juanes *et al. BMC Bioinformatics* (2017) 18:421
DOI 10.1186/s12859-017-1837-z

BMC Bioinformatics

SOFTWARE Open Access

VISMapper: ultra-fast exhaustive cartography of viral insertion sites for gene therapy

José M. Juanes[1,2†], Asunción Gallego[3,4†], Joaquín Tárraga[2,5], Felipe J. Chaves[6,7], Pablo Marín-García[6,8], Ignacio Medina[5], Vicente Arnau[1,2,8] and Joaquín Dopazo[3,9,10*]

Abstract

Background: The possibility of integrating viral vectors to become a persistent part of the host genome makes them a crucial element of clinical gene therapy. However, viral integration has associated risks, such as the unintentional activation of oncogenes that can result in cancer. Therefore, the analysis of integration sites of retroviral vectors is a crucial step in developing safer vectors for therapeutic use.

Results: Here we present VISMapper, a vector integration site analysis web server, to analyze next-generation sequencing data for retroviral vector integration sites. VISMapper can be found at: http://vismapper.babelomics.org.

Conclusions: Because it uses novel mapping algorithms VISMapper is remarkably faster than previous available programs. It also provides a useful graphical interface to analyze the integration sites found in the genomic context.

Keywords: Gene therapy, Viral insertion, Viral integration, Sequence mapping, Genome viewer

***Figura 3.31**. Artículo de **VISMapper**, publicado en 2017 en la revista **BMC Bioinformatics** (Q1): https://pubmed.ncbi.nlm.nih.gov/28931371.*

3.2.1 **Introducción y objetivos**

Necesidad biomédica

Una de las necesidades básicas en los estudios que emplean la integración de virus en el genoma es el conteo y la localización de las copias del virus que se han insertado en el genoma.

Hoy en día, el uso de vectores virales es un procedimiento convencional en la terapia génica clínica (Gaspar *et al.*, 2004; Cartier *et al.*, 2009). Pero, a pesar de su éxito, las terapias basadas en la integración viral no están exentas de riesgos, como puede ser la activación accidental de oncogenes que pueden causar la transformación maligna de las células (Cavazzana-Calvo *et al.*, 2010; Paruzynski *et al.*, 2010).

Las ubicaciones de los vectores en el genoma del paciente pueden servir de marcadores moleculares que ayudan a rastrear el destino de las células afectadas. El análisis de estos lugares de inserción de vectores (IS) se lleva a cabo mediante la amplificación de secuencias de los vectores retrovirales con una repetición terminal larga (LTR) (Ishak *et al.*, 2020). Actualmente, esto se consigue usando tecnologías de secuenciación de nueva generación (NGS).

Los cebadores que mapean las LTR producen lecturas de secuencia con uniones cromosómicas LTR, que pueden usarse para determinar con precisión la región cromosómica de inserción del vector viral (Paruzynski *et al.*, 2010). La monitorización de estas posiciones de

inserción es necesaria puesto que se sabe que distintos vectores de transferencia de genes pueden tener preferencias para dirigirse a regiones de genes codificantes, islas *CpG* o sitios de inicio de la transcripción (Schröder *et al*., 2002; Mitchell *et al*., 2004; Wu *et al*., 2004).

Objetivos y solución bioinformática

Para satisfacer esta necesidad de la detección de lugares de inserción de virus en experimentos de terapia génica, se desarrolló una aplicación web de análisis y visualización de datos denominada **VISMapper** (Juanes *et al.*, 2017).

VISMapper realiza el cálculo de los sitios de inserción de manera muy eficiente, y proporciona una interfaz gráfica muy intuitiva que permite la visualización interactiva de los lugares de inserción víricos y los genes cercanos a esos sitios, de forma gráfica y tabular. Además, proporciona una vista del contexto genómico de cada inserción.

El código generado para esta aplicación está libremente disponible en el repositorio de *GitHub*[34]. También se puede acceder directamente a la aplicación desde la url: http://vismapper.babelomics.org.

[34] Repositorio de **VISMapper**: https://github.com/MGvizPro/vismapper

3.2.2 **Resultados**

VISMapper ha sido escrita en *Node.js* y utiliza una arquitectura **híbrida** para las diferentes pantallas o vistas. La aplicación utiliza una herramienta de terceros, *GenomeMaps* (Medina *et al.*, 2013), para la visualización de los resultados en el contexto del genoma. Los sitios de inserción viral resultantes de un experimento se pueden visualizar junto con las regiones genómicas que tienen alrededor, incluidas las lecturas mapeadas, los genes y otros tipos de elementos genómicos (extraídos de las bases de datos de COSMIC (Forbes *et al.*, 2010) y *CellBase* (Bleda *et al.*, 2012)). Esta aplicación funciona tanto con versiones del genoma humano *GRCh37* como con el *GRCh38*.

Para la visualización de los lugares de inserción se creó una biblioteca gráfica en *JavaScript* y SVG, y que está disponible en un repositorio en GitHub[35]. Esta biblioteca facilita la exploración de los datos a nivel cromosómico, dando una primera vista de todos los lugares de inserción en el cariotipo (**Figura 3.32**). Además, permite seleccionar el cromosoma de interés y pasar a una vista más detallada (Figura 3.2.4) que integra una serie de módulos para la interactividad con la aplicación, haciendo que la exploración de los datos sea dinámica e intuitiva.

[35] Repositorio de **Karyo.js**: https://github.com/biowt/karyojs

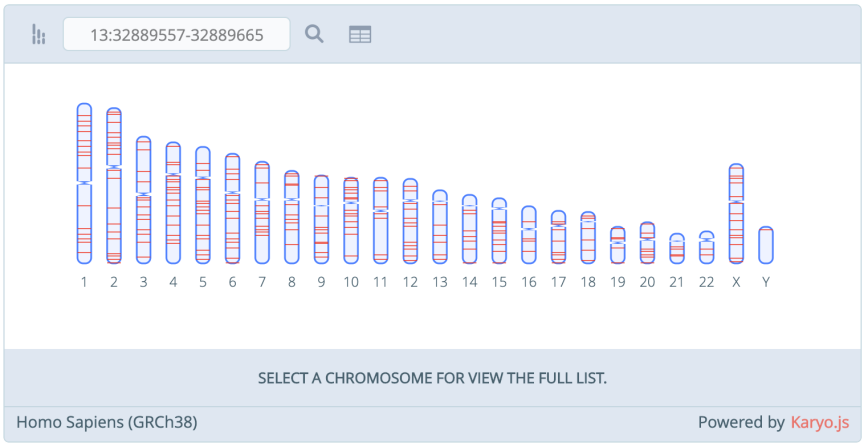


Figura 3.32. *Vista inicial de la biblioteca* ***Karyo.js****.*

El módulo de representación en SVG de la vista detallada del cromosoma contiene una serie de eventos para la captura de la posición del cursor, con tal de mostrar la posición cromosómica en todo momento (**Figura 3.33**:2, etiqueta azul). También se muestra la información sobre las inserciones al situar el ratón encima de cada una de ellas (**Figura 3.33**:2, etiqueta roja).

*Figura 3.33. Distribución de los módulos de la biblioteca **Karyo.js**. Consta de tres módulos principales: la cabecera, con un área de búsqueda **(1)**; una zona de dibujo SVG, para el cariotipo y cromosomas **(2)**; y un módulo de tabla ligado al ideograma **(3)**.*

El módulo de la cabecera (**Figura 3.33**:1) está dotado con un cuadro de búsqueda que permite definir una región de interés para su posterior exploración en el visor genómico (*GenomeMaps*). La región introducida en la zona de búsqueda produce un evento que es escuchado tanto por el módulo de SVG, que representa gráficamente la zona seleccionada en el cromosoma (**Figura 3.34**:1), como por el visor genómico, que muestra en detalle los lugares de inserción y los genes de la base de datos de *COSMIC* que se encuentran dentro de la región proporcionada.

El módulo de la tabla (**Figura 3.33**:3) lista todos los lugares de inserción que se encuentran en el cromosoma. Para cada uno de los lugares de inserción se muestra su posición en dicho cromosoma, el número de lecturas y el gen de la base de datos de *COSMIC* más

cercano a izquierda y a derecha del lugar de inserción. Posicionando el cursor encima de una fila se lanza un evento con la información de la región de inserción de esa fila, el cual es escuchado por el módulo de representación en SVG y actualiza el gráfico mostrando una etiqueta con la información de la inserción en la posición correspondiente del ideograma (**Figura 3.34**:2).

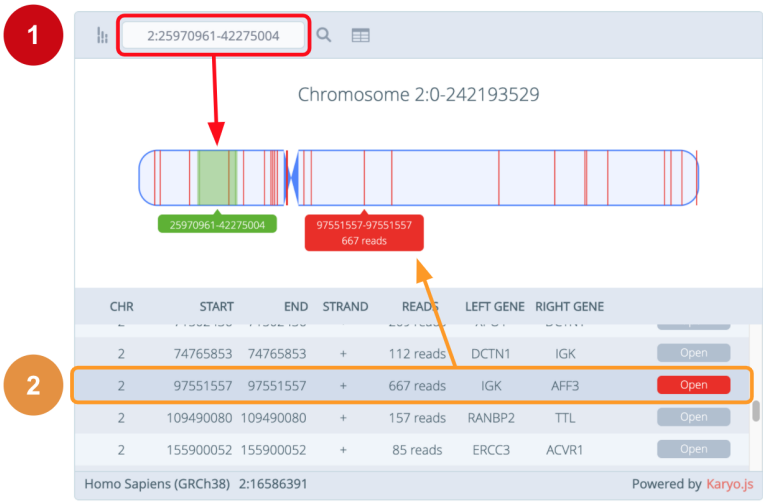


Figura 3.34. *Esquema de la interactividad entre los módulos. La región introducida en la zona de búsqueda se traduce en un evento que es escuchado por la zona de representación gráfica, que responde resaltando la región introducida en el cromosoma (1). Por otro lado, cada fila de la tabla se conecta con la inserción representada en el ideograma, haciendo que al situar el ratón sobre cada una de las filas se resalta la inserción en el cromosoma (2).*

Subida de datos a la aplicación

VISMapper acepta archivos de tipo FASTQ que contengan las lecturas correspondientes a los sitios de inserción del virus. También acepta archivos de tipo FASTA, aunque a diferencia de los archivos FASTQ, estos últimos no incluyen la calidad de cada base, y, por tanto, la aplicación los convierte en archivos FASTQ asignándole a todas las bases una puntuación de calidad de 20 (*Phred quality score)*. Este valor minimiza el ratio de falsos positivos cuando las secuencias tienen una calidad estándar. En cualquier caso, el uso de archivos FASTQ es más recomendable.

Los archivos se pueden comprimir en un ZIP para que el tamaño de la subida sea menor. Durante la subida, el usuario puede proporcionar también un email para ser notificado cuando el proceso de análisis de sus datos haya finalizado (aunque debido a la rapidez con la que los datos son analizados, usualmente no es necesario).

Mapeo de las lecturas

Las lecturas subidas en el archivo FASTQ se alinean en el genoma de referencia humano utilizando **BWA** (Li y Durbin, 2009) o ***HPG-Aligner*** (Tárraga *et al*., 2014).

Los sitios de inserción se detectan utilizando lecturas que hayan sido mapeadas solo parcialmente. Para ello, se utiliza la información contenida en el campo *CIGAR* (Slater y Durbin., 2005) del resultado del mapeo para filtrar las lecturas: cuando el *CIGAR* de una lectura

mapeada contenga secuencias con bases cortadas (*hard clipping*) o enmascaradas (*soft clippings*) indica que la correspondiente lectura tiene parte de la secuencia del genoma y además también tiene parte de la secuencia del virus. Estas lecturas se ordenan por cromosoma utilizando *SAMTools* (Li *et al*., 2009) y se registran en el sistema en una base de datos *MySQL* para facilitar el acceso a ellas por medio de la aplicación web (**Figura 3.35**).

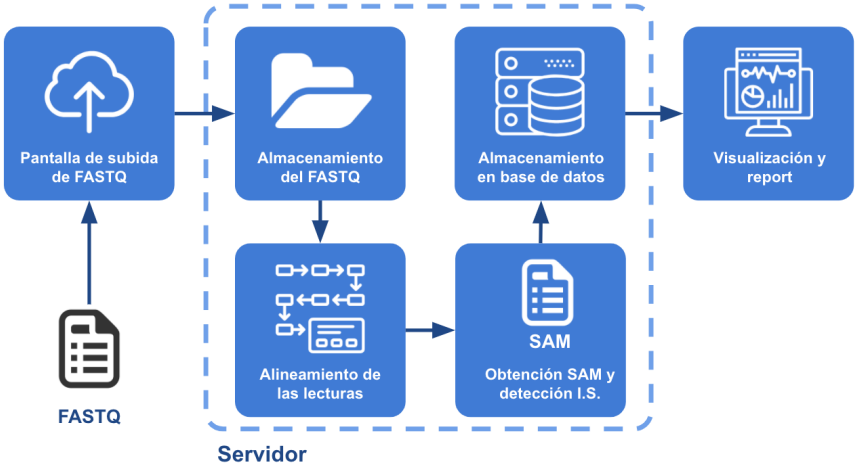


Figura 3.35. *Flujo del procesamiento y análisis de las secuencias subidas al servidor de* ***VISMapper****.*

Interfaz de usuario

El interfaz de usuario es un entorno de trabajo gráfico compuesto por tres paneles: el visor de cromosomas, el visor genómico y el panel de filtros (**Figura 3.36**).

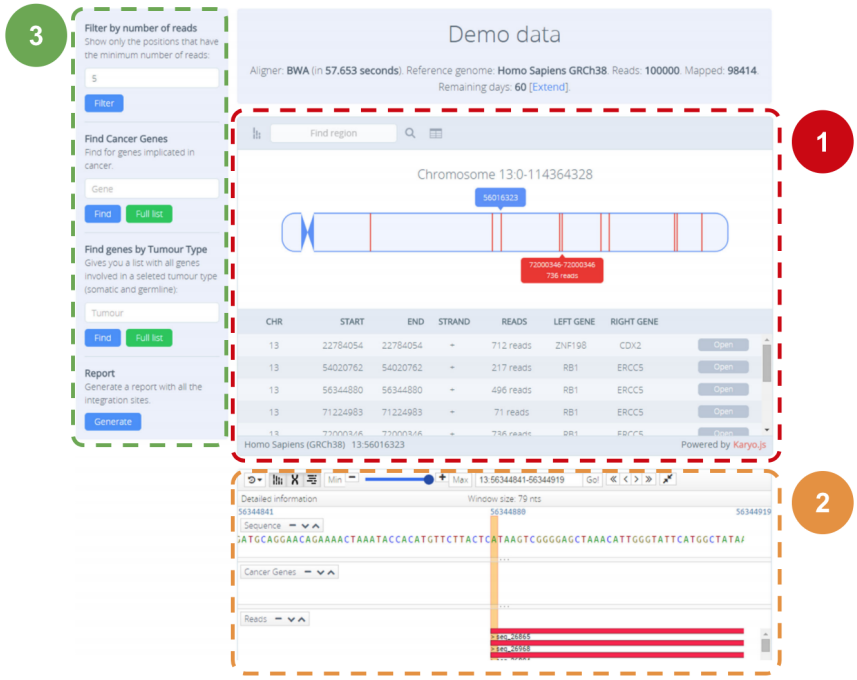


Figura 3.36. Captura de pantalla de las diferentes secciones del cuadro de mandos de la aplicación. El ideograma, el visor genómico y la tabla con el listado de lugares de inserción.

El visor de cromosomas proporciona una perspectiva general de todos los lugares de inserción a lo largo de los cromosomas. Al hacer clic con el botón izquierdo del ratón sobre uno de los cromosomas, se amplía dicho cromosoma, pudiendo explorar con más detalle los lugares de inserción marcados con líneas rojas. Situando el ratón sobre cada uno de los lugares de inserción se puede obtener información más detallada sobre su ubicación, así como el número de lecturas que respaldan dicho lugar de inserción. El panel de filtros situado a la izquierda permite filtrar los lugares de inserción por el número de lecturas que los soportan. También permite realizar

búsquedas de las lecturas más cercanas a los oncogenes relacionados con tipos de tumores específicos.

Se puede obtener una vista más detallada de la región en la que se producen los lugares de inserción (que se puede seleccionar pulsando con el ratón en el visor de los cromosomas) con el visor genómico, que implementa *GenomeMaps*. Dependiendo del nivel de zoom en el visor genómico se pueden presentar diferentes paneles:

1) La región genómica circundante.
2) Oncogenes cercanos. Situando el ratón sobre ellos muestra información sobre los genes.
3) Lecturas mapeadas en los lugares de inserción. Cuando se sitúa el ratón sobre ellos se muestra la información sobre la lectura, como por ejemplo el sentido de la hebra de ADN, la calidad del mapeo, etc.

Por último, el panel de filtros permite establecer un umbral basado en el número de lecturas que admiten lugares de inserción y permite encontrar genes de cáncer específicos o genes de tipos de cáncer específicos. Específicamente, en un apartado de este panel permite establecer un umbral con el número mínimo de lecturas para considerar un lugar de inserción (5 por defecto). Otro de los apartados de este panel de filtros permite seleccionar un oncogén específico (que puede ser buscado por nombre o seleccionarse de una lista), o mostrar solo los genes que se sabe que están asociados con un tumor dado. La lista de oncogenes mostrados en la aplicación ha sido extraída de la base de datos de COSMIC.

Informe

Finalmente, se diseñó una pantalla de informe, que presenta una tabla que contiene todos los lugares de inserción encontrados. Los datos mostrados en dicha tabla se pueden filtrar por número de lecturas y por la distancia a los genes de relevancia oncológica. También se pueden ordenar según los criterios que se muestran en el encabezado de las columnas (cromosoma, posición, calidad, etc.). Esta lista puede ser exportada como archivo de texto delimitado por tabuladores (TSV) y también se puede descargar un archivo BAM con las alineaciones generadas por el alineador (**Figura 3.37**).

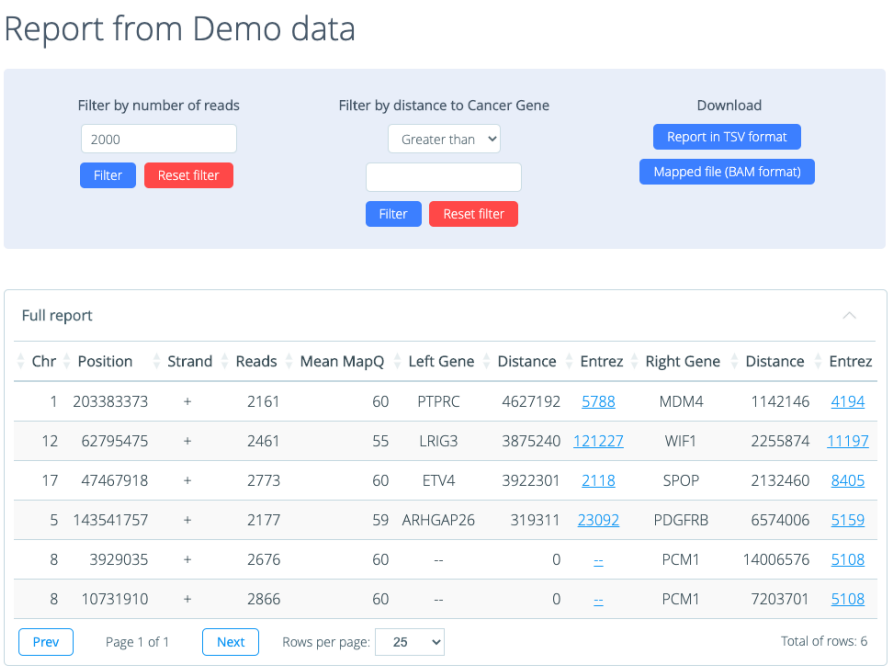


Figura 3.37. *Tabla con el informe final de las inserciones encontradas donde se le pueden aplicar los filtros por número de lecturas y de distancia a genes de relevancia oncológica.*

Para cada lugar de inserción se proporciona la siguiente información, organizada en columnas:

- Cromosoma
- Posición genómica
- Número de lecturas que han mapeado en esta posición.
- Promedio de la calidad de las lecturas que han mapeado en esta posición concreta.
- Oncogen más cercano.
- Distancia al oncogen más cercano (si el valor es 0 significa que el lugar de inserción ha sido mapeado en el oncogen).
- Posición del oncogen con respecto al lugar de inserción.
- Identificador NCBI del oncogen.
- Dirección web a la página de NCBI con información más detallada del oncogen.

3.2.3 **Comparación con otros servicios y aplicaciones similares**

En el momento de la realización de este trabajo (2017), existían otros servicios y herramientas web para el análisis de lugares de inserción de vectores virales, como HISAP (Arens *et al.*, 2012), SeqMap (Jiang y Wong, 2008) (aunque requiere que el usuario se registre en el sitio web), QuickMap (Applet *et al*., 2009) o VISA (Hocum *et al.*, 2015), esta última publicada recientemente (en comparación con la fecha de publicación de **VISMapper**). Sin embargo, todos ellos usan BLAST (Altschul *et al*., 1990) o BLAT (Kent, 2002) para el mapeo de las lecturas, lo que implica tiempos de ejecución mucho más largos que

con **VISMapper**. La **Figura 3.38** muestra una comparación de tiempos de ejecución, donde el aumento de la velocidad obtenido gracias al uso de algoritmos de mapeo más sofisticados en **VISMapper** es evidente. Los datos utilizados en la comparación han sido extraídos del sitio web de VISA y también se pueden descargar desde el repositorio de **VISMapper** en GitHub[36].

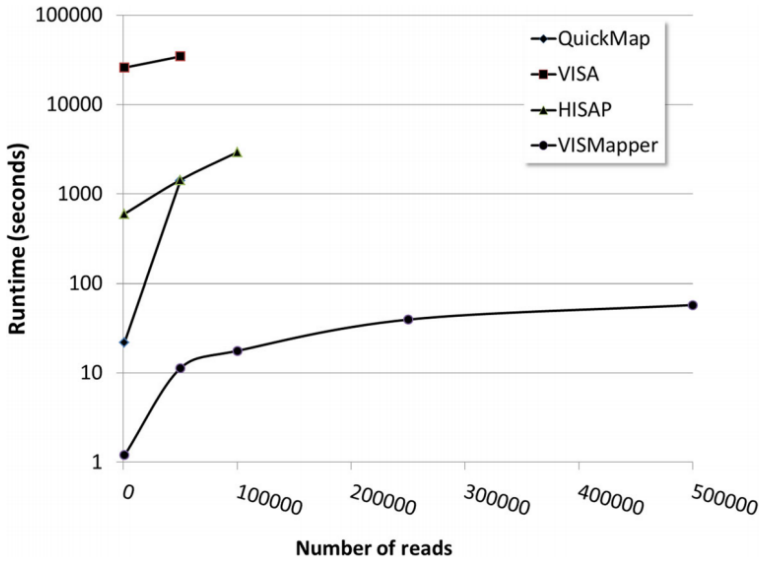


Figura 3.38. *Tiempo empleado por diferentes programas para encontrar las posiciones de inserción vírica.*

Además, se realizó una comparación más detallada con el programa VISA generando 4 conjuntos de datos con un número conocido de

[36] Datos de prueba para **VISMapper**:
https://github.com/jmjuanes/vismapper/tree/master/ismapper-test

lugares de inserción utilizando el programa generador disponible en el sitio web VISA[37]. La Tabla 3.2.1 muestra los resultados de la comparación. Los tiempos de ejecución relativos son similares a los que se muestran en la **Figura 3.38**. Si bien ambas herramientas dan un número muy pequeño de falsos positivos, en general **VISMapper** es capaz de mapear un mayor porcentaje de lecturas y encuentra más lugares de inserción que VISA.

Tabla 3.2.1. *Comparación entre* ***VISA*** *y* ***VISMapper*** *utilizando los datos generados desde la página web de* ***VISA****.*

Dataset	Lecturas	IS	Rendimiento	VISA	VISMapper
			Tiempo	~72 h	~60s
Input 1	100.000	100.000	IS	99.694	99.793
			Lecturas mapeadas	99.694	99.881
			Tiempo	~72 h	~60s
Input 2	50.000	50.000	IS	49.854	49.897
			Lecturas mapeadas	49.855	49.936
			Tiempo	~72 h	~30s

[37] VISA: https://visa.pharmacy.wsu.edu/bioinformatics

Input 3	10.000	10.000	IS	9.992	9.969
			Lecturas mapeadas	9.995	9.981
			Tiempo	~5 h	~30s
Input 4	1000	1.000	IS	906	929
			Lecturas mapeadas	906	930

Respecto al número de lecturas que puede procesar cada herramienta, QuickMap no procesa más de 50.000 lecturas y VISA está limitado a entre 50.000 y 100.000 lecturas. HISAP puede procesar hasta 100.000 lecturas en aproximadamente 50 minutos, pero no puede procesar más de 250.000 lecturas. Al contrario de **VISMapper**, ninguna de las otras herramientas proporciona una interfaz gráfica para visualizar los resultados, y QuickMap y HISAP no son compatibles con GRCh38.

3.2.4 **Conclusiones**

Debido a su velocidad y sensibilidad, **VISMapper** constituye una alternativa atractiva a las opciones actualmente disponibles para el análisis de los lugares de inserción de virus. **VISMapper** ofrece un entorno de trabajo gráfico único e interactivo que permite una

exploración detallada y exhaustiva de las consecuencias y riesgos potenciales de los vectores virales insertados en el genoma analizado.

3.2.5 **Tutorial**

En la wiki del repositorio en GitHub se preparó un tutorial de uso para manejo de la aplicación. Dicho tutorial es accesible a través de la url: https://github.com/jmjuanes/vismapper/wiki.

3.2.6 **Actualización**

En el capítulo 3.5 se ha realizado una re-implementación esta herramienta aplicando los avances en visualización desarrollados como parte de los objetivos propuestos para este proyecto de tesis (**Figura 3.39**).

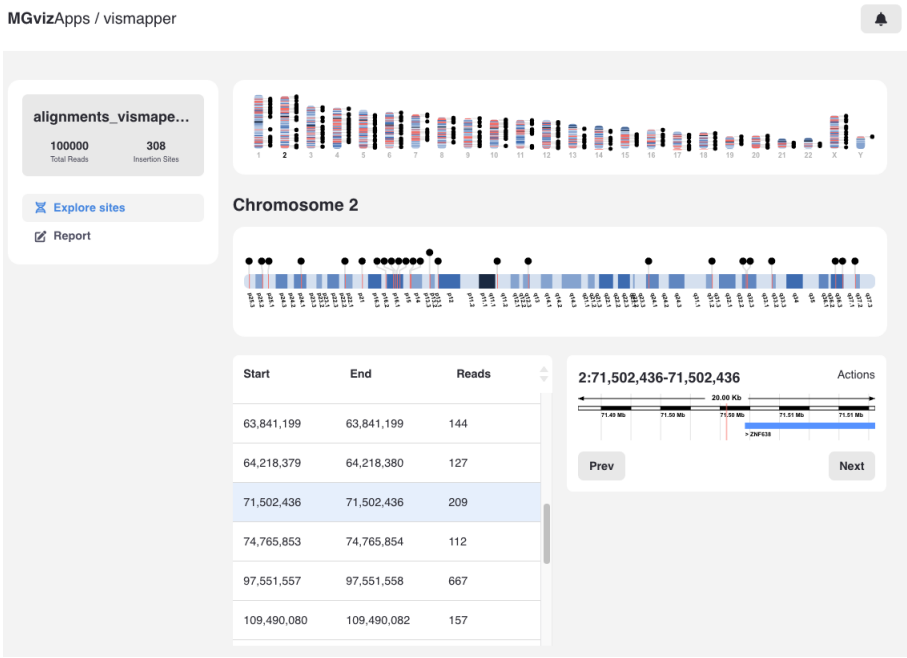


Figura 3.39. *Re-implementación de* ***VISMapper*** *con la nueva tecnología reactiva que se ha desarrollado en la plataforma* ***MGvizApps*** *como uno de los resultados de esta tesis.*

3.3 MGvizSAP

3.3.1 **Introducción**

Necesidad biomédica

Una de las obligaciones del contrato predoctoral consistía en la estancia, de dos años, en la empresa *Seqplexing* para desarrollar aplicaciones bioinformáticas para sus kits de diagnóstico. Esta estancia supuso una oportunidad para estudiar y evaluar las fortalezas y debilidades de las aproximaciones técnicas usadas hasta la fecha en este tipo de aplicaciones. Esto permitió generar nuevas hipótesis sobre los modelos y metodologías en desarrollo de *software*.

Los resultados de este estudio han servido de fundamento para el resto de la investigación de este trabajo, centrado en el desarrollo de aplicaciones modulares y sus componentes de visualización interactiva.

Objetivos y soluciones bioinformáticas

La principal necesidad a cubrir en el proyecto de NGS desarrollado junto a *Seqplexing* fue permitir a los clientes analizar las muestras secuenciadas mediante una aplicación web. Para permitir estos análisis, se estudió cómo desarrollar un sistema automático de procesado de los resultados de secuenciación proporcionados por los kits de diagnóstico, incluyendo sus controles de calidad. También se estudió y desarrolló una aplicación web, orientada para la priorización de variantes muestras tumorales y la realización de un informe genético.

Estos kits de diagnóstico están orientados para el análisis genético de muestras oncológicas. La naturaleza de los datos de secuenciación de muestras oncológicas confiere una complejidad extra a los procesos de análisis y control de calidad (pureza, clonalidad, etc.). Por tanto, para poder llegar a representar esos datos de forma útil en la plataforma, fue necesaria la realización de un trabajo minucioso de *data wrangling* y control de calidad, con tal de depurarlos y anotarlos. Así, se dió al usuario la información pertinente para su uso en diagnóstico genético.

Con respecto a la automatización, se estudió y se diseñó un flujo de trabajo secuencial, incluyendo análisis de calidad en cada uno de los diferentes pasos y pantallas en la aplicación para hacer el seguimiento del estado de las muestras. Como objetivo principal de la misma, se ideó la creación de un visor y priorizador de variantes que permitiera hacer informes genéticos con los resultados. Además, se diseñó la

subida de muestras al servidor de análisis para aquellos clientes que tuviesen su propio sistema de secuenciación en remoto.

Como se ha mencionado anteriormente, la solución a estudiar estaba orientada al análisis de datos de cáncer, lo que implica ciertos ajustes. Esto se debe a que el llamado de variantes en cáncer no produce genotipos, ya que el tumor contiene una mezcla heterogénea de genomas. Por lo tanto, las variantes no se definen por su genotipo sino por su fracción alélica o *Variant Allele Fraction* (VAF) en el *pool* de células secuenciadas del individuo.

Este concepto de fracción alélica es importante, pues es la razón por la que las muestras de cáncer deben ser analizadas por NGS en vez de por secuenciación Sanger para mantener una buena precisión en el diagnóstico durante los estados iniciales de la enfermedad. La técnica Sanger no detecta fracciones alélicas por debajo del 15-20%, por lo que pierde sensibilidad a la hora de detectar muestras con poca carga tumoral (por ejemplo, en estados iniciales de una leucemia).

También es importante ver las fracciones alélicas de diferentes mutaciones, incluso en el mismo *locus*, para determinar el estado de evolución clonal, que puede ser importante a la hora de detectar o predecir la resistencia a fármacos oncológicos (**Figura 3.40**). Y, finalmente, el cálculo de la VAF también es importante, ya que sirve de estimación de cuántas células cancerosas hay con una mutación concreta en el tumor (**Figura 3.41**).

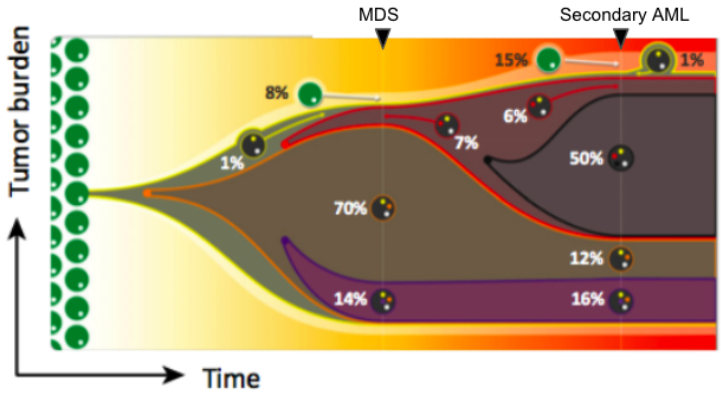


Figura 3.40. *Modelo que resume la evolución clonal desde el estado de síndrome mielodisplástico (MDS) hasta el estado secundario de leucemia mieloide aguda (AML) utilizando datos de fracción alélica de variantes (VAF). Las células verdes en el lado izquierdo representan células madre hematopoyéticas normales (HSPC). Cada HSPC tiene su propio conjunto de mutaciones únicas adquiridas durante el envejecimiento (punto blanco). Al ocurrir un evento transformador, una única célula gana una ventaja de crecimiento y se expande, arrastrando todas las mutaciones preexistentes (blanco), además de las 192 mutaciones del grupo 1 (amarillo). Las células del clon 1 (amarillo) contienen mutaciones del grupo 1. Las células del clon 2 (naranja) se originan a partir de una única célula del clon 1 (ya que todas las mutaciones del grupo 1 son heterocigotas y están presentes en la mayoría de células del AML secundario) y, por tanto, contienen todas las mutaciones de los grupos 1 y 2. Las células del clon 3 (rojo) se originan a partir de una célula del clon 1 y contienen todas las mutaciones de los grupos 1 y 3. Las células del clon 4 (morado) se originan desde una célula del clon 2 y contienen las mutaciones de los grupos 1, 2 y 4. Las células del clon 5 (negro), el último de los clones en surgir (englobando el 50% de la celularidad de la médula ósea en el AML secundario), contiene mutaciones de los grupos 1, 3 y 5. Durante la transformación al AML secundario, el clon 2, que arrastra una mutación en la nucleofosmina (NPM1), colapsa desde un 70% de las*

células en el MDS hasta un 12% de las células en el AML secundario, al mismo tiempo que surge el clon 5. Al realizar solamente un seguimiento del VAF de la mutación sobre NPM1 en el paciente, podría subestimarse la carga tumoral y la respuesta terapéutica a lo largo del tiempo. Esto resalta una limitación del seguimiento de la clonalidad y la carga tumoral estudiando un solo gen responsable (Jacoby, Duncavage y Walter, 2015).

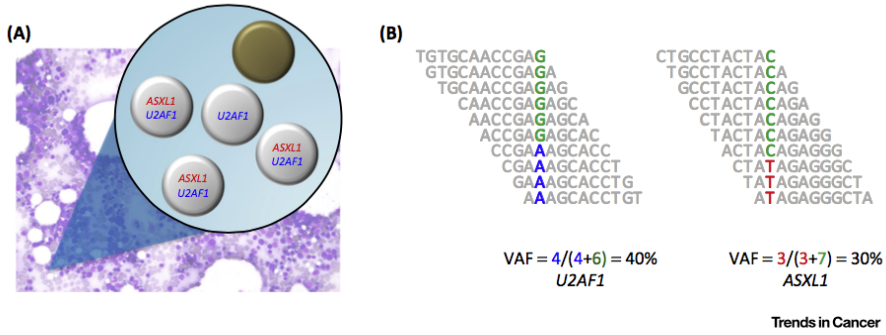


Figura 3.41. *Cálculo de la fracción alélica de variantes (VAF) a partir de lecturas de NGS. (A) Muestra de una mielodisplasia, de donde se extrae en la biopsia una célula normal (en gris oscuro) y cuatro tumorales (en gris claro). Mirando las mutaciones, el gen U2AF1 está mutado en 4 células (80%) y el gen ASXL1 en tres células (que, además, tiene mutado el U2AF1). Por esto, se deduce que la mutación en el gen U2AF1 aparece en la clona inicial y la mutación ASXL1 en una subclona. (B) Muestra de 10 lecturas para cada gen, con el nucleótido de referencia en verde, y las variantes de U2AF1 en azul y las variantes de ASXL1 en rojo. U2AF1 tiene 4 lecturas mutadas sobre 10 (40%) y ASXL1, 3 mutadas sobre 10 (30%). La fracción de células con mutaciones es directamente proporcional a la VAF de mutaciones heterocigotas observada; por lo tanto, una VAF del 40% representa el 80% de las células con mutaciones en U2AF1, y un 60% de las células porta mutaciones en ASXL1. (Jacoby, Duncavage y Walter, 2015).*

3.3.2 **Resultados**

Este proyecto se orientó como un trabajo de investigación, donde se planteó la hipótesis de que la visualización de los experimentos y sus resultados, junto a la exploración de los datos, servirían para mejorar los kits y ayudarían a crear nuevos flujos de trabajo basados en las evidencias mostradas con las visualizaciones.

En una primera etapa, se trabajó en la visualización de los experimentos estudiando las necesidades de los análisis de laboratorio. Posteriormente, esto sirvió para desarrollar **jviz**, una biblioteca de visualización reactiva y el principal resultado de esta tesis.

Como ejemplo, la primera visualización necesaria en todo proceso analítico de NGS es la de diseño del experimento, con la que se puede transmitir a los bioinformáticos la complejidad del proceso del laboratorio. Esto es vital para que puedan detectar posibles incompatibilidades entre el nuevo experimento y sus flujos de trabajo establecidos (**Figura 3.42**).

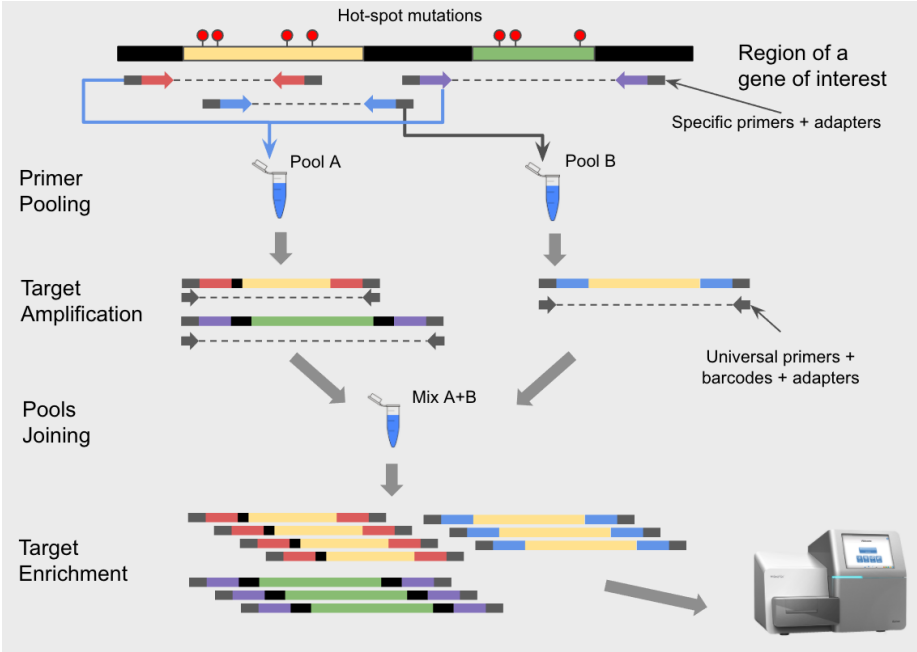


Figura 3.42. *Infografía del protocolo de un experimento.*

Este proyecto estaba basado en la secuenciación de exones por amplicones de PCR. Esta secuenciación implica la presencia de amplicones solapantes en los exones largos que, por tanto, no pueden amplificarse en la misma PCR, obligando a agruparlos en diferentes *pools* de PCR. La visualización de los amplicones, su localización, solapamiento y calidad de la secuenciación son importantes para el control de calidad. Lo mismo sucede con los cebadores (*primers*), para los que la visualización de la calidad y la especificidad a la región destino del genoma también deben tenerse en cuenta.

Teniendo en cuenta la importancia de estas visualizaciones para el control de calidad, se crearon módulos específicos para evaluar *a priori* la calidad de los amplicones diseñados. Así, el laboratorio puede

predecir la eficacia de los diseños y reanalizarlos si fuera necesario (**Figura 3.43**).

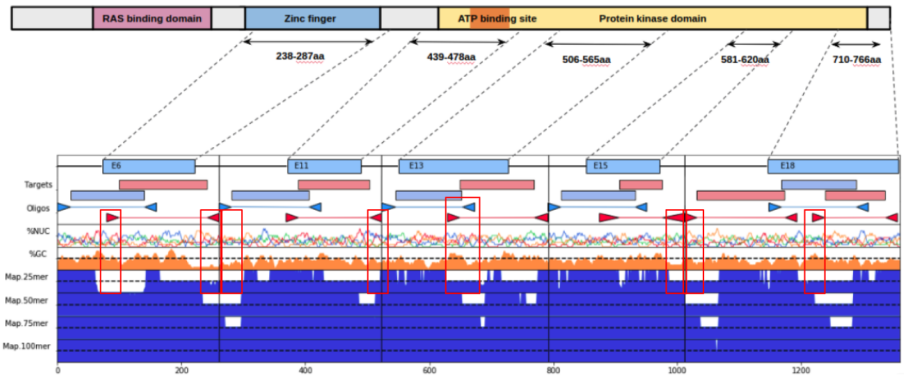


Figura 3.43. *Ejemplo de un gen con amplicones diseñados solo para ciertos exones de interés. Debido a la naturaleza de los amplicones, algunos cebadores acaban localizados en zonas con regiones de alta similaridad con otras partes del genoma. Toda visualización de cebadores debe ir acompañada de gráficos de mapabilidad para oligómeros cortos (25 pares de bases) y de tamaños de 75-100 pares de bases (las lecturas de secuenciación a mapear en el genoma). Los cuadros rojos indican amplicones que tendrán poca eficiencia y contribuirán a lecturas fuera de la zona de interés debido a hibridaciones no deseadas.*

Otro aspecto que necesita visualización para el control de calidad es la localización de las regiones amplificadas, en el que se marcarían las que estuvieran fuera de las regiones de interés (ROI) (**Figura 3.44**). Su visualización es fundamental para el control de calidad del experimento. Su implementación puso en evidencia que en los primeros experimentos se creaban amplicones artefactuales (en rojo en la **Figura 3.43**). Estos amplicones, aunque podían ser limpiados bioinformáticamente, representaban una pérdida de eficiencia.

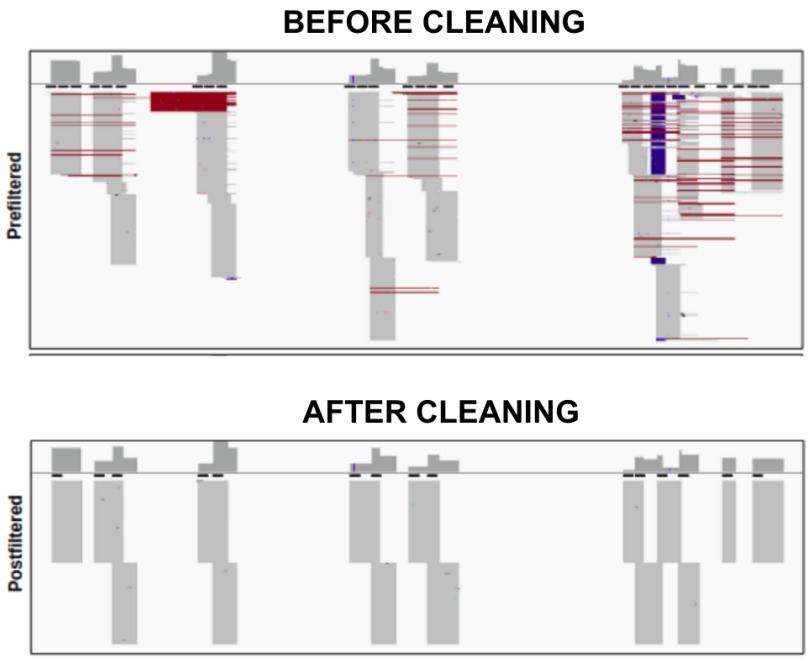


Figura 3.44. *Visualización de las lecturas alineadas. El panel superior muestra en rojo las lecturas del secuenciador que no corresponden a los amplicones predefinidos e indican emparejamientos de cebadores cercanos no deseados. En el panel inferior se muestra la visualización de las lecturas después de aplicar una pipeline especial para limpiar estos artefactos bioinformáticamente.*

Para el experimento de optimización de reducción de artefactos, se realizó otra visualización que mejora la comprensión del diseño (**Figura 3.45**) y de los resultados del experimento (**Figura 3.46**). Se diseñó un experimento para ver los 3 factores que podrían afectar a los artefactos: el efecto de cuándo se juntaban los grupos de cebadores, la aplicación de un aditivo y la modificación de la muestra añadida a la 2ª PCR. Las fuentes de error podían deberse a arrastrar cebadores de la PCR anterior, lo que creaba amplicones de extremo

a extremo de zonas solapadas o amplicones cercanos (*artefactos internos o on-target*). Otro problema es que los cebadores presentaran complementariedad con otras zonas del genoma y dieran amplificados espurios (*artefactos externos* u *off-target*). En la **Figura 3.46** se ven los resultados y como el gráfico muestra claramente que no combinar los cebadores en la 2ª PCR tiene un efecto muy beneficioso en la mejora de los *on-target* y ligero sobre los *off-target*, y que un porcentaje determinado de un aditivo da una clara mejora sobre el porcentaje de *off-targets* y finalmente muestra que, como cabía esperar, las diluciones de muestra solo afectan cuando se juntan los *pooles* en el PCR2. Este experimento de tres factores con replicados, muestra que un aditivo concreto al 20% y no mezclar los *pooles* puede reducir un 90% los artefactos, pasando de un 30% a un 4% de artefactos totales.

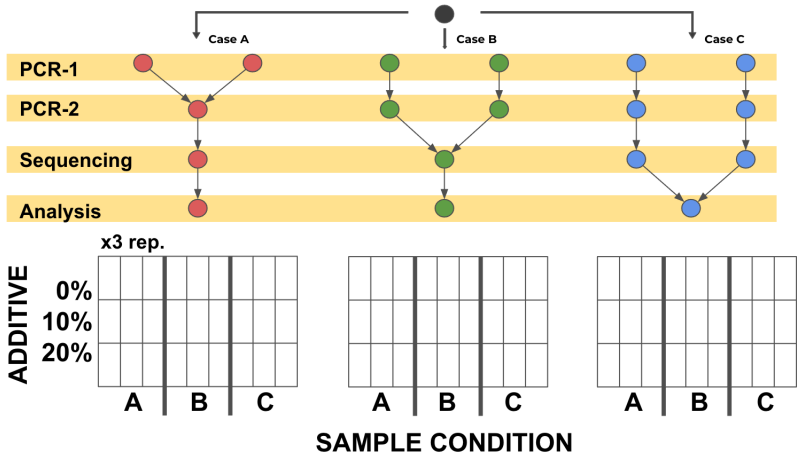


Figura 3.45. *Diseño del experimento para ver 3 factores: el efecto de juntar los grupos de cebadores, la adición de un aditivo y modificaciones del protocolo (A, B y C).*

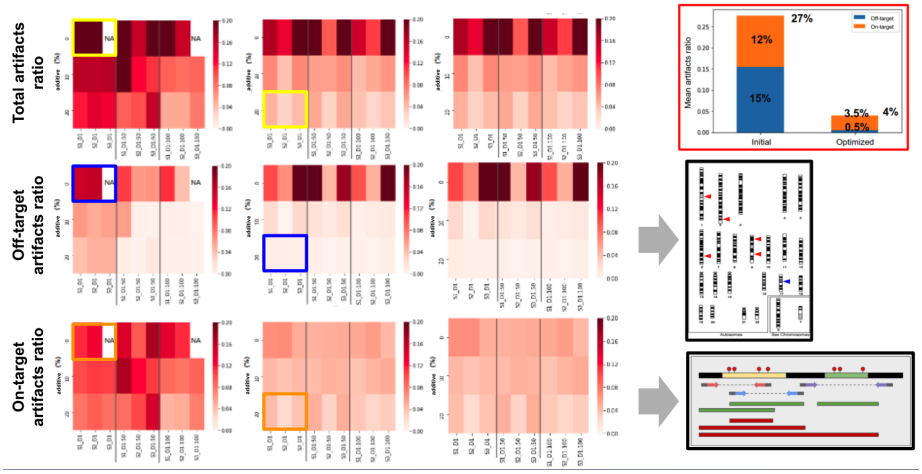


Figura 3.46. *Resultado del experimento por triplicado donde se ve que no juntar los pooles hasta secuenciación ayuda a mejorar los on-target (columna central vs izquierda) y añadir un aditivo mejora los off-target (fila central) pero casi nada los on-target (fila inferior)*

Finalmente, dentro del análisis de visualización hay otro tipo de figuras muy importantes en NGS como son las de cobertura (**Figura 3.48**) y frecuencia de variantes según diferentes condiciones de agrupación para descubrir sesgos o errores de protocolo en los experimentos (**Figura 3.49**).

Un ejemplo de la utilidad de las gráficas de coberturas (**Figura 3.47**) es la visualización del resumen de cobertura en grupos de muestras (**Figura 3.48**:A). La zona azul muestra la variabilidad entre muestras y la línea azul oscuro la mediana y la verde la media. La cobertura por amplicones debería dar una línea recta por amplicón como en la **Figura 3.47**, pero en la **Figura 3.48**:A se ve que algunos amplicones tienen una especie de rampa en los extremos, lo que visualmente alerta de un posible problema en la secuenciación. Cuando estos

exones se miran más detenidamente con la herramienta IGV[38] se observa que la polimerasa tiene problemas con esas zonas y comete muchos errores. Estas visualizaciones ayudan al laboratorio a mejorar los diseños y muestra al investigador qué zonas hay que evitar al diseñar los cebadores.

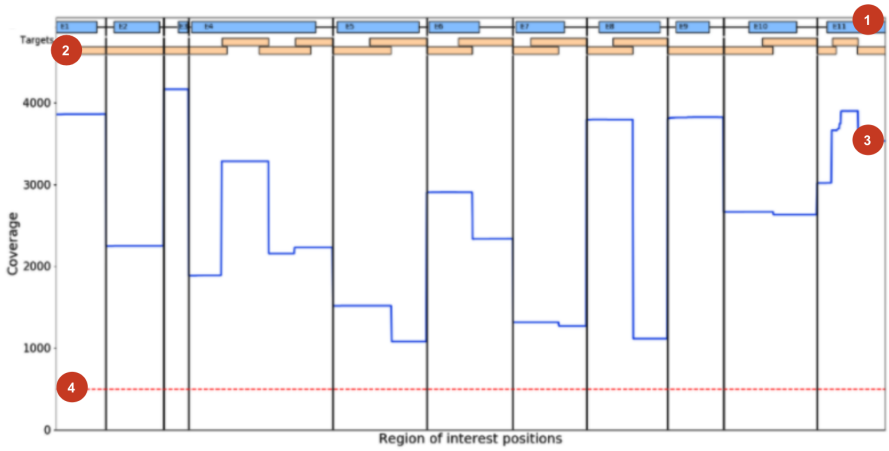


Figura 3.47. *Gráfica de cobertura para una muestra de un gen. Mostrando (1) los exones de interés, (2) las regiones secuenciadas (ROI), (3) perfil de cobertura (número de lecturas por base) y (4) umbral mínimo de calidad de cobertura (regiones por debajo del umbral quedan excluidas del análisis y se deben informar como no analizadas)*

[38] **Integrative Genomics Viewer:**
http://software.broadinstitute.org/software/igv

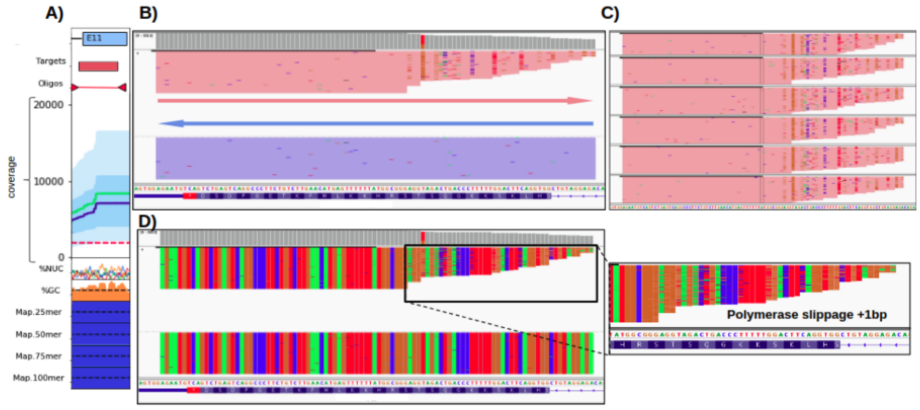


Figura 3.48. *Ejemplo de visualización de la cobertura de un exón en un grupo de muestras (A). La zona azul muestra la variabilidad entre muestras y la línea azul oscuro la mediana y la verde a media. El panel B muestra el efecto del tartamudeo de la polimerasa y su efecto en la pérdida de lecturas y acúmulo de variaciones en el extremo del amplicón.*

Por último, mostrar la importancia de los gráficos exploratorios de datos en los procesos de control de calidad. Toda *pipeline* de NGS debe tener visualizaciones que muestren los resultados de los análisis de laboratorio a lo largo del tiempo, para detectar anomalías como en la **Figura 3.49**, donde en la parte superior se ve el sesgo por cadena y la calidad de las variantes obtenidas por cada una de las carreras del secuenciador para el total de muestras. En el recuadro rojo se ve una anomalía durante varias carreras debido posiblemente al uso de un *batch* defectuoso en los reactivos del secuenciador. En la fila inferior, se muestran diferentes parámetros de calidad para variantes de un gen agrupados por exones, donde se detecta que el *strand bias* es superior en el exón 11 y que también ese mismo exón tiene menor calidad, lo que ayuda al laboratorio a rediseñar ese exón. También se ve que ciertos exones tienen mayor variabilidad en la profundidad de

lectura y puede indicar que haya que cambiar o concentración de cebadores o mejorar el diseño.

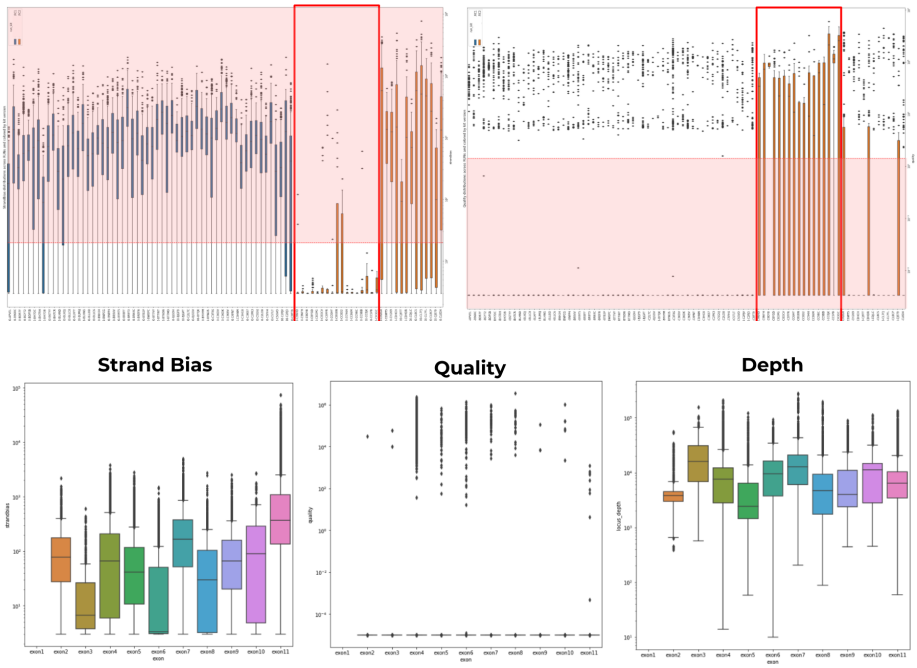


Figura 3.49. *Visualizaciones de los datos resumen de varias carreras de secuenciación con múltiples muestras para un gen. (Parte superior) boxplots mostrando strand bias y calidad de las variantes encontradas para cada una de las carreras. (Parte inferior) Strand bias, calidad y profundidad de lectura de las variantes del total de muestras de un gen agrupadas por exones.*

En este trabajo, además de diseñar visualizaciones para ayudar a representar los experimentos del laboratorio, se ha realizado también el estudio de UI y UX sobre herramientas de priorización de variantes y creación de informes genéticos. Como fruto de este estudio, se ha creado la herramienta **Sequencing Analysis Platform** (SAP). Se trata de una aplicación web intuitiva para el procesamiento y el control

de calidad (QC) de los datos obtenidos en secuenciación NGS e identificación de variantes en las regiones analizadas. Esta aplicación se ha desarrollado para uso profesional en el ámbito de la genómica clínica y el diagnóstico de precisión.

A diferencia de las dos aplicaciones anteriores, en esta se ha implementado un sistema de validación de las credenciales de los usuarios, usando *JSON Web Tokens* (JWT), y un área de usuario donde se guardan todos los análisis subidos y realizados por el mismo (**Figura 3.50**).

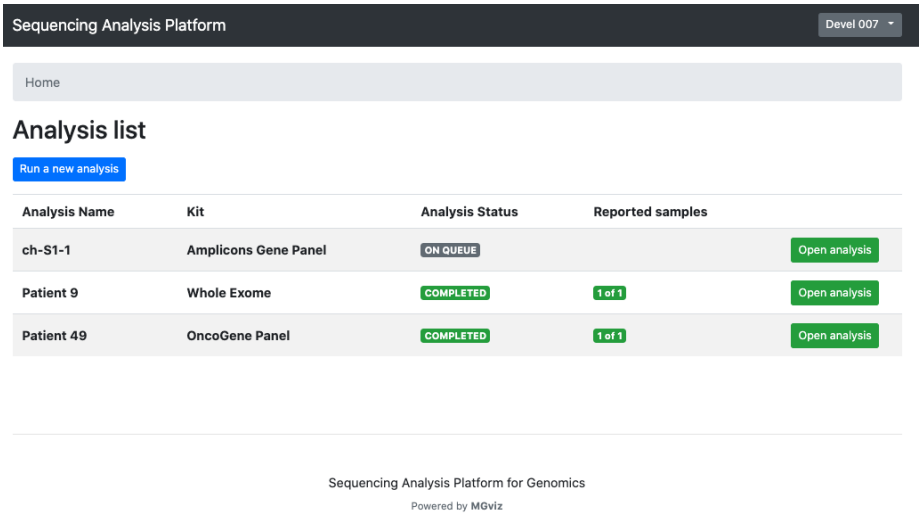


***Figura 3.50**. Área de usuario de la aplicación SAP que muestra los análisis que tiene creados y arriba a la derecha la identidad del usuario.*

La aplicación ha sido diseñada para la realización de interpretaciones clínicas de las variantes genéticas, mediante filtros automáticos para la priorización de variantes relevantes clínicamente, y permite refinamientos manuales de las variantes detectadas en las muestras

analizadas. En un primer paso, el usuario debe subir sus muestras al servidor desde una vista de subida de datos. El servidor posee un flujo de trabajo para el procesado y análisis automático de las muestras. Con el fin de poder comprobar el estado del procesamiento en todo momento, la aplicación cuenta con una vista que informa del estado del proceso global, siguiendo un código de colores para ello (**Figura 3.51**):

- Gris oscuro para indicar que la muestra estaba pendiente de ser procesada.
- Naranja para indicar la muestra que se está procesando.
- Verde para indicar que la muestra se ha procesado correctamente.
- Rojo para mostrar errores y avisos.

En cada una las muestras completadas se muestra el informe de los controles de calidad, una gráfica de coberturas (que muestra en rojo las zonas por debajo de un umbral establecido y que permite, a simple vista, captar el estado global del conjunto de muestras del análisis), así como un resumen de las variantes con posible interés clínico o informables y una serie de ficheros descargables.

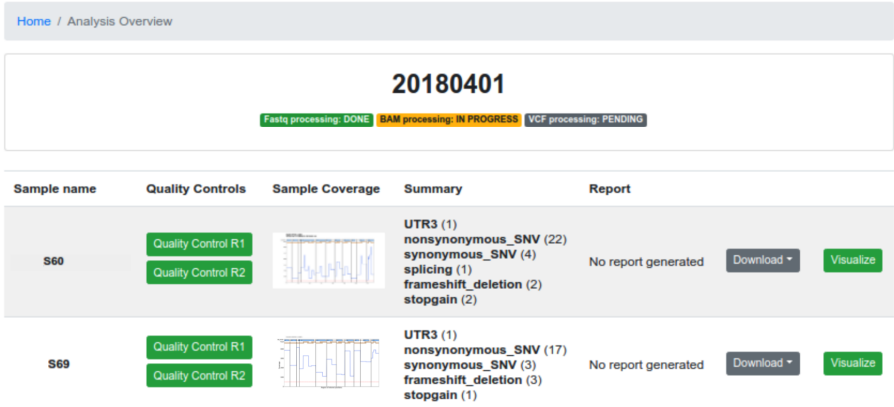


***Figura 3.51**. Pantalla de muestras analizadas en la aplicación **SAP** y el estado de la cola de análisis.*

El UI de la aplicación está orientado a la ayuda en la toma de decisiones durante la selección de variantes génicas de relevancia clínica. Por tanto, se ha planteado un diseño claro e intuitivo, con tal de facilitar el manejo de las distintas vistas al analista.

La vista de *Interpretación Clínica* presenta un UI sencillo, que permite filtrar y agrupar los resultados, así como seleccionar automática o manualmente las variantes de interés y asignarles diferentes niveles de certidumbre clínica. Para gestionar las selecciones de variantes, presenta dos pestañas extra, una para contener las variantes relevantes (*Relevant variants*) y otra para variantes que formarán parte del informe (*Report variants*). Estas dos pestañas permiten a los usuarios con diferentes roles centrarse en aquel conjunto de variantes de su interés (**Figura 3.52**).

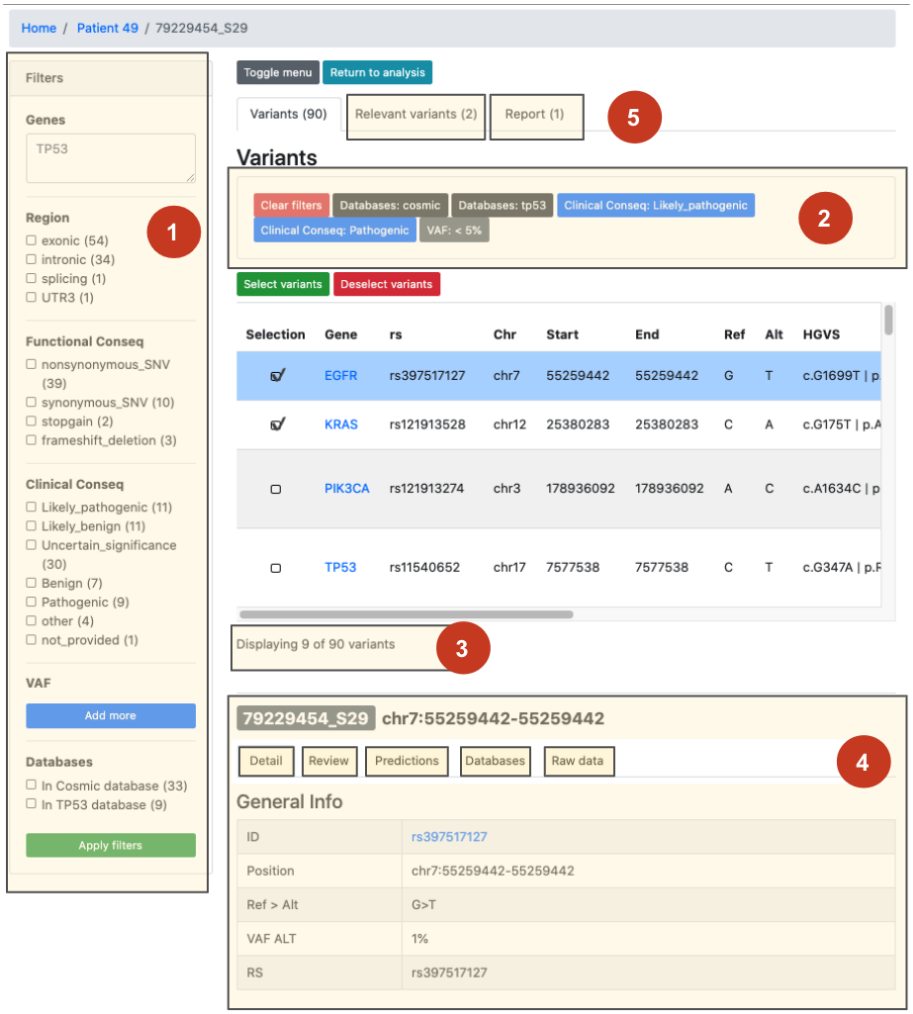


Figura 3.52. *Pantalla principal de la vista de priorización de variantes donde se muestra (1) el panel con los posibles filtros a aplicar a las variantes, (2) área de manejo de los filtros activados, (3) contador de variantes filtradas respecto de las totales, (4) área de pestañas con información detallada por variante y (5) pestañas de priorización clínica, «Relevant» y «Report».*

En términos de UI, lo más costoso ha sido encontrar y testar la estructura idónea de un sistema de ayuda a la toma de decisiones

(DSS) para la vista de «Interpretación Clínica», donde, por un lado, se debe mostrar unos datos mínimos en la tabla donde se seleccionan las variantes, y por otro, se debe dar un acceso rápido y fluido a toda la información necesaria, para que el clínico decida si acepta o rechaza dicha variante. Finalmente se ha conseguido una exploración interactiva eficaz gracias al rápido acceso que tiene el usuario a los datos extra que necesita, accediendo a ellos a partir de pestañas de información detallada.

Cuando el usuario presiona sobre una variante, se muestran estas pestañas de información sobre la variante que facilitan su priorización. Estas pestañas muestran información biológica y clínica necesarias para valorar las consecuencias clínicas de las variantes detectadas. El contenido de las pestañas se ha fraccionado condensando la información por categorías conceptualmente significativas. Facilitando de esta forma el refinamiento manual de los filtros y priorizaciones, y mejorando la toma de decisiones.

Las pestañas de información detallada más relevantes que se han implementado en la aplicación han sido la de control de calidad (**Figura 3.53**), la de anotación clínica de la variante (**Figura 3.54**), la de información de bases de datos específicas (**Figura 3.55**) y la de revisión de la variante (**Figura 3.56**). Esta última es la de más utilidad para la creación de una plataforma de análisis bioinformático, donde siempre debe haber un nivel de curación de datos para la mejora de análisis futuros y para permitir mantener las referencias actualizadas. Este módulo de revisión ha sido refactorizado en las aplicaciones finales de **MGvizApps** (capítulo 3.5)**,** cumpliendo así uno de los

objetivos importantes de este trabajo: la mejora de los sistemas de DSS en bioinformática.

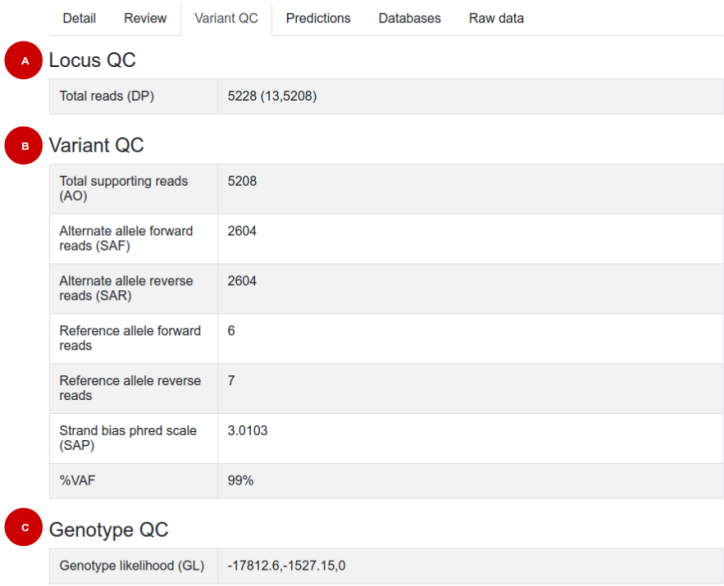


Figura 3.53. *Pestaña de control de calidad (variant QC), en la que se muestra información sobre valores de control de calidad para valorar la fidelidad a nivel de locus (A), de variante (B) y de genotipo (C).*

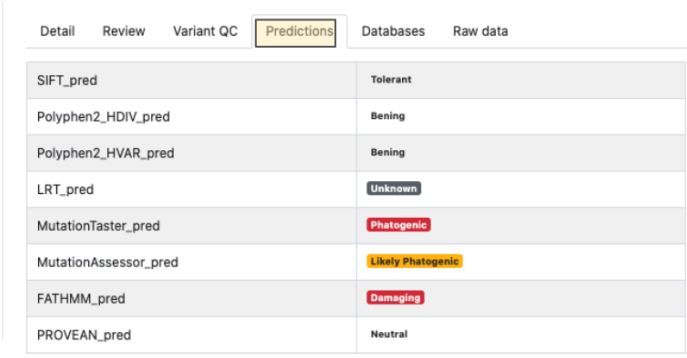


Figura 3.54. *Vista con los detalles de las anotaciones clínicas de la variante (**Predictions**).*

Cosmic

ID	COSM1318437 COSM1318436 COSM1318435 COSM3388196 COSM1318438 COSM43951 COSM3388197
Occurrence	2(liver) 4(lung) 9(large_intestine) 1(upper_aerodigestive_tract) 1(endometrium) 4(oesophagus) 1(haematopoietic_and_lymphoid_tissue) 1(salivary_gland) 2(breast) 1(central_nervous_system) 1(pancreas) 1(urinary_tract)

TP53

g_description	g.7578206T>C
c_description	c.643A>G
ProtDescription	p.S215G
Structural_motif	NDBL/beta-sheets
Hotspot	yes
Mut_rate	0.1580
Mut_rateAA	0.1580
cBioportalCount	12
COSMIClink	43951
CLINVARlink	265337
PubMed	12792784 14612556 14669308 15138567 15492791 15564288 18467159 21567059 0001851662 0008080737 0007627932 0009816061 0008932338 0010674608

Figura 3.55. *Pestaña con información de la variante en bases de datos de referencia clínica y bases de datos específicas del gen (**Databases**).*

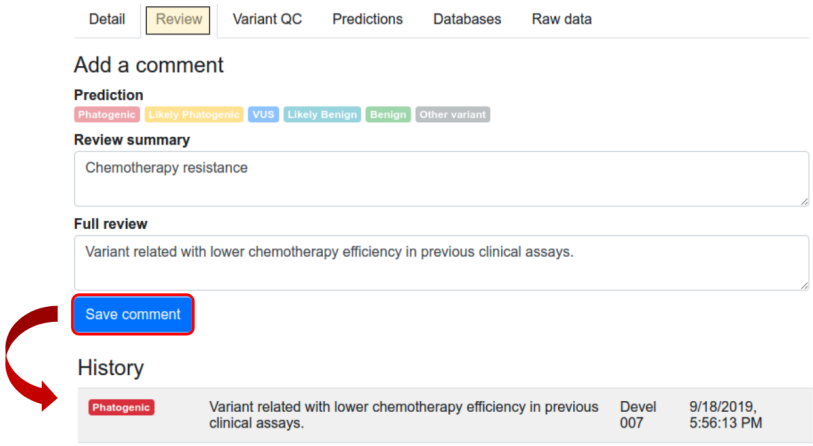


***Figura 3.56**. Pestaña de revisión de variantes (Review) donde se puede asignar la categoría clínica de la variante y añadir, para los informes, anotaciones relevantes sobre la evidencia de esta variante en el fenotipo.*

Como resultado de la investigación sobre el UI para la priorización manual de las variantes para el informe y se obtuvo una vista donde la priorización manual se hace en tres pasos. Tras el filtrado automático en la pestaña de ***Variants***, se pueden seleccionar todas las variantes encontradas por el filtro o refinar esta selección manualmente gracias a las pestañas de detalle para cada variante. Las variantes seleccionadas, quedan automáticamente marcadas como relevantes y pasan a la tabla de la pestaña de ***Relevant Variants***.

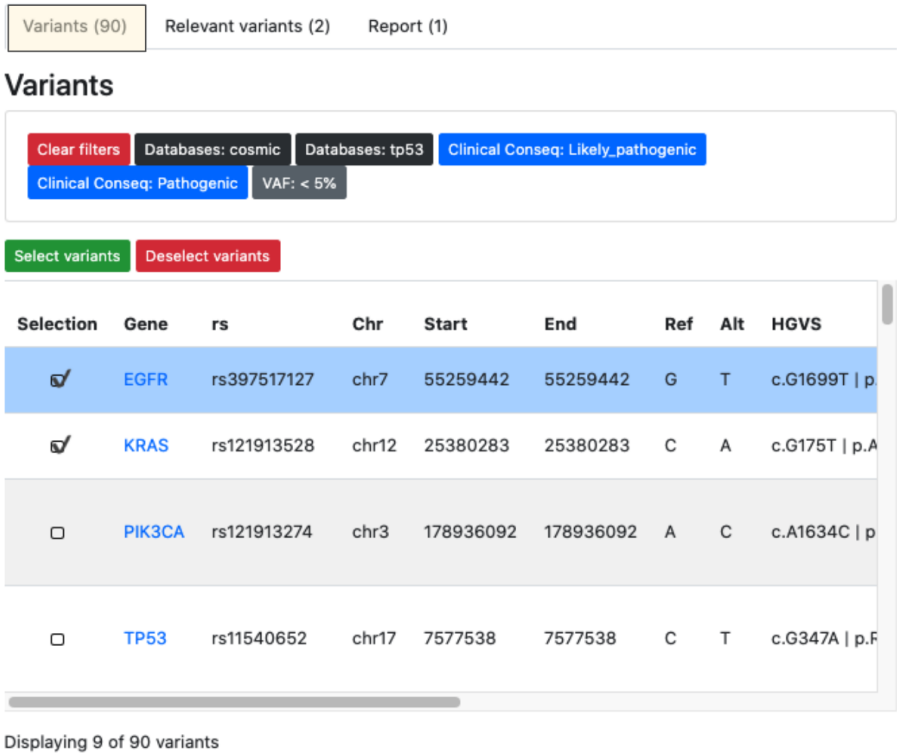


***Figura 3.57**. Ejemplo de variantes que cumplen las reglas de filtrado seleccionadas. Contiene la tabla con información general de las variantes seleccionadas; muestra el número de variantes seleccionadas tras filtrado frente al total de variantes detectadas en la muestra; y permite la selección de variantes para guardarlas en el estado o pestaña de «Relevant variants».*

En la pestaña ***relevant variants*** (**Figura 3.58**), el usuario puede seguir filtrando variantes, consultar información para continuar con la priorización y seleccionar las variantes que se van a reportar.

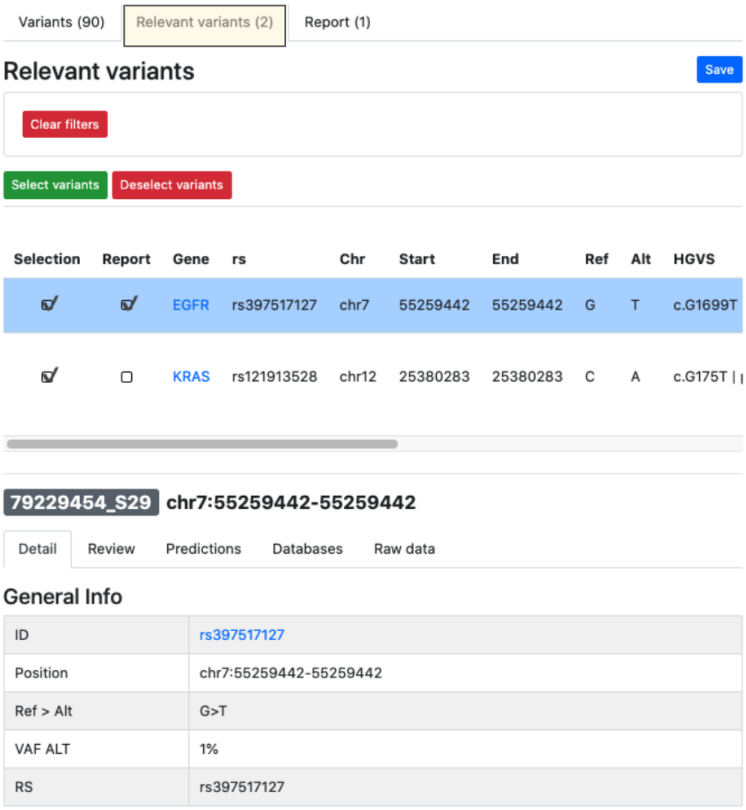


Figura 3.58. Variantes seleccionadas para realizar el informe técnico-clínico.

Durante el desarrollo de esta aplicación, se hicieron varios estudios sobre diferentes formas de confeccionar los informes clínicos y su automatización. Finalmente, se optó por crear una vista de informe en una pestaña resumen, donde los datos del caso se pueden rellenar automáticamente (mediante llamadas al HIS por HL7 en sistemas hospitalarios) o a mano. También se da una opción de revisar la selección y rellenar la interpretación del informe viendo los datos de las pestañas de detalle de la variante (**Figura 3.59**).

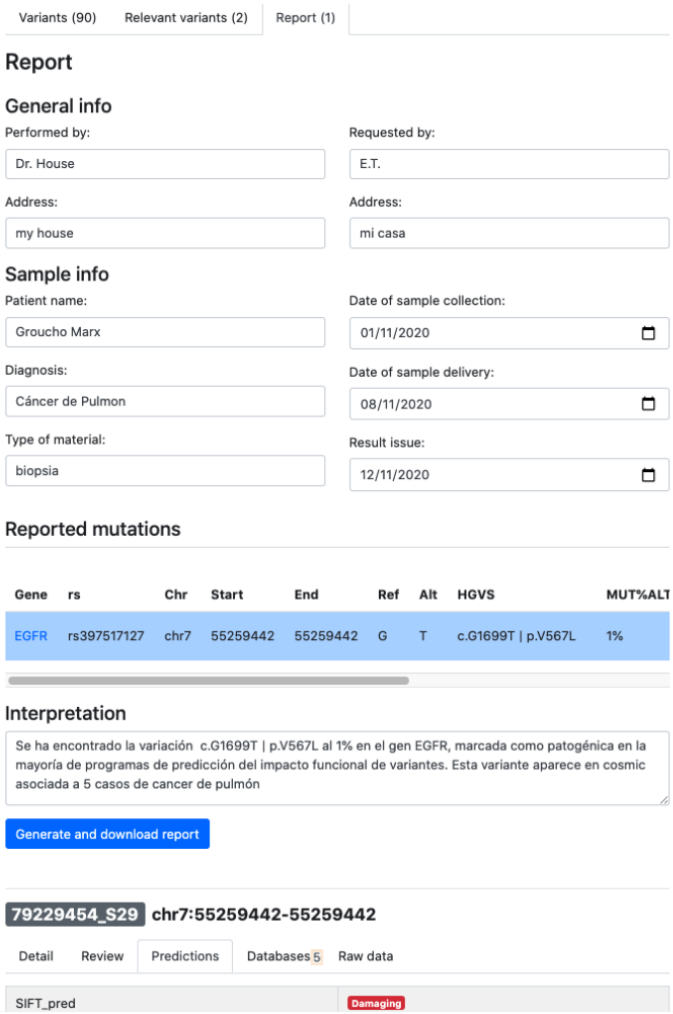


Figura 3.59. *Vista de la pantalla que permite la generación del informe, donde se puede anotar la información de la muestra y la interpretación, que posteriormente se puede descargar en formato MS WORD.*

3.4 jviz

3.4.1 **Introducción**

Esta sección presenta **jviz**, un sistema de visualización con un lenguaje declarativo para la creación de visualizaciones reactivas. **jviz** consta de tres partes:

1. La **gramática**, con la cual se describen los elementos del gráfico, la apariencia visual y el comportamiento interactivo en formato JSON.
2. El **intérprete** en JavaScript, que convierte el esquema en JSON en un gráfico SVG reactivo.
3. **E**l **editor online,** que permite probar este sistema.

El sistema de visualización **jviz** está basado en el modelo formalizado en el libro de Leland Wilkinson, *The grammar of graphics* (Wilkinson, 2005), y en el trabajo de H. Wickham, *A Layered Grammar of Graphics* (Wickham, 2010) En este sistema, los elementos que forman parte de cualquier gráfico se pueden dividir en varias capas lógicas, tal y como se muestra en la **Figura 3.60**.

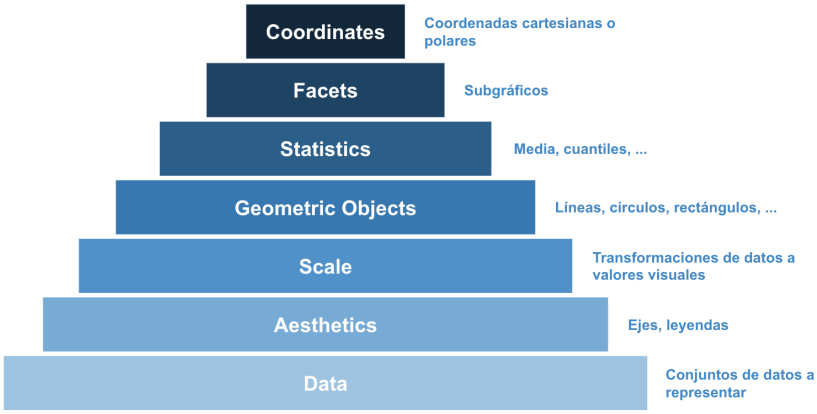


***Figura 3.60**. Capas del «Grammar of Graphics».*

Cada una de estas capas lógicas actúa sobre diferentes elementos visuales presentes en un gráfico (**Figura 3.61**). Como paso previo al desarrollo de **jviz**, se realizó el estudio de cómo está construido un gráfico, cuáles son sus elementos primarios y cómo se definen las relaciones entre sus elementos. El resultado fue la especificación de las capas de **jviz** y los elementos principales de su gramática.

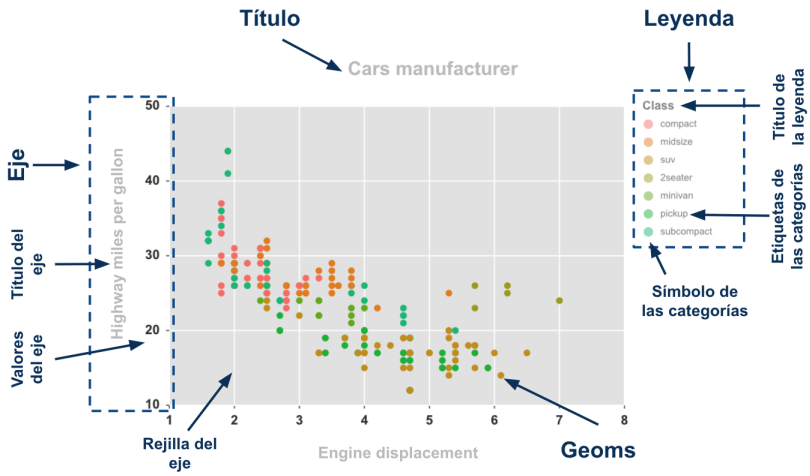


Figura 3.61. Anatomía de los elementos visuales de un gráfico.

3.4.2 Gramática de jviz

Jviz proporciona una gramática mediante la cual se pueden describir los elementos que componen cualquier visualización, su apariencia y su comportamiento. Esta gramática se debe expresar en formato JSON, la cual **jviz** es capaz de traducir a un gráfico SVG. El gráfico generado puede ser tanto estático (por ejemplo, para ser incluido en un documento de texto o PDF) como interactivo (para ser utilizado en aplicaciones web).

La gramática de **jviz** permite definir bloques de construcción básicos para una amplia variedad de diseños de visualización: obtención y transformación de datos, escalas, ejes, leyendas y elementos gráficos primitivos como rectángulos, líneas, símbolos, etc. Además, permite definir variables dinámicas que pueden ser modificadas como

respuesta a la interacción del usuario con los elementos visuales que componen el gráfico.

Elementos de un gráfico de jviz

Para la creación de un gráfico con **jviz** es necesario construir un **esquema en formato JSON,** cuyos elementos siguen la gramática detallada en esta sección. Para seleccionar qué elementos son los que definen el aspecto y comportamiento de un gráfico, se revisaron los trabajos sobre la Gramática de Gráficos de Wilkinson, 2005. A partir de este estudio se decidió separar los elementos de un gráfico reactivo en 12 capas, las cuales se muestran en la Tabla 3.4.1. En la **Figura 3.62** se puede ver la comparación entre las capas propuestas en este trabajo y las definidas por la Gramática de Gráficos, mostradas en la **Figura 3.60**.

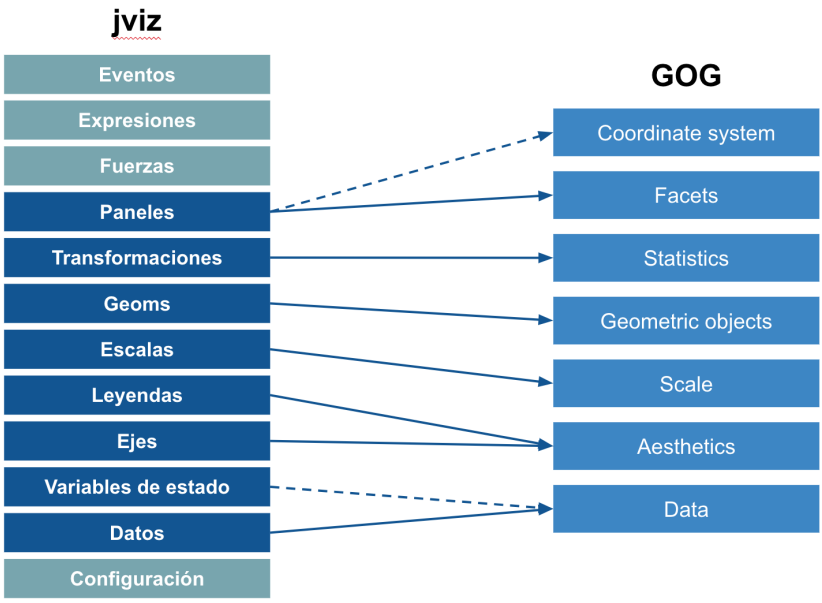


Figura 3.62. *Analogía entre las capas de* ***jviz*** *y las del «Grammar of Graphics».*

Tabla 3.4.1. Elementos de un gráfico en jviz.

Capa	Descripción
Configuración	Configuración global del gráfico (altura, anchura, márgenes).
Variables de estado	Variables dinámicas cuyo valor puede ser modificado en tiempo real (por ejemplo, por interacción del usuario con el gráfico).
Datos	Diferentes conjuntos de datos que se van a utilizar para generar el gráfico.

Expresiones	Lenguaje para la evaluación de fórmulas matemáticas.
Transformaciones	Operaciones que se aplican sobre los datos.
Paneles	Distribución del gráfico en subgráficos independientes.
Escalas	Funciones que transforman valores procedentes generalmente de los conjuntos de datos en valores visuales como posición en píxeles o colores.
Ejes	Especificación de cómo se van a representar los diferentes ejes de coordenadas.
Leyendas	Visualización gráfica de las escalas.
Fuerzas	Generación de gráficos basados en diagramas de fuerzas.
Geoms	Elementos geométricos representados en el gráfico, como por ejemplo rectángulos, círculos, líneas, etc..
Eventos	Permiten registrar las interacciones del usuario con el gráfico.

Generalmente, un esquema de **jviz** sigue una estructura como la

mostrada en el **Código 3.1**. En la primera parte del esquema se suelen definir las variables globales del gráfico (altura, anchura, márgenes y estilos), así como el tema (configuración de colores) y el título. A continuación, se declaran las variables de estado (*state*) y los datos (*data*), fuerzas físicas (*forces*), las escalas, leyendas, ejes y, por último, las figuras geométricas (*geoms*) que serán representadas.

```
{
    "width": 100,
    "height": 100,
    "margin": 0,
    "outerMargin": 0,
    "theme": "default",
    "title": null,
    "state": [],
    "data": [],
    "forces": null,
    "scales": [],
    "legends": [],
    "axes": [],
    "geoms": []
}
```

***Código 3.1**. Estructura general de un esquema de **jviz**.*

Cualquier otro campo diferente a los que están especificados en el esquema anterior será considerado como información adicional, y no será utilizado para la generación del gráfico. Por ejemplo, se podrían añadir campos a este esquema como "$author$" y "$description$" para especificar el autor del gráfico y la descripción del mismo, respectivamente.

Configuración

Un gráfico en **jviz** necesita de una serie de parámetros iniciales que permitan definir las dimensiones y la estética del mismo (color de fondo, fuente, tamaño del texto, etc.). Estos parámetros pueden ser divididos en las siguientes categorías:

1. **Dimensiones**: para configurar el tamaño del gráfico.
2. **Márgenes**: para configurar los márgenes usados en el gráfico, que pueden ser internos o externos.
3. **Título**: para configurar el título del gráfico.
4. **Temas**: para configurar el estilo general del gráfico.

Dimensiones y márgenes del gráfico

La dimensión del gráfico se puede definir en el esquema utilizando los atributos "$width$" para la anchura y "$height$" para la altura. Ambos parámetros son obligatorios, y el valor proporcionado debe ser un número entero positivo, representando el tamaño en píxeles de la anchura y la altura (**Figura 3.63**).

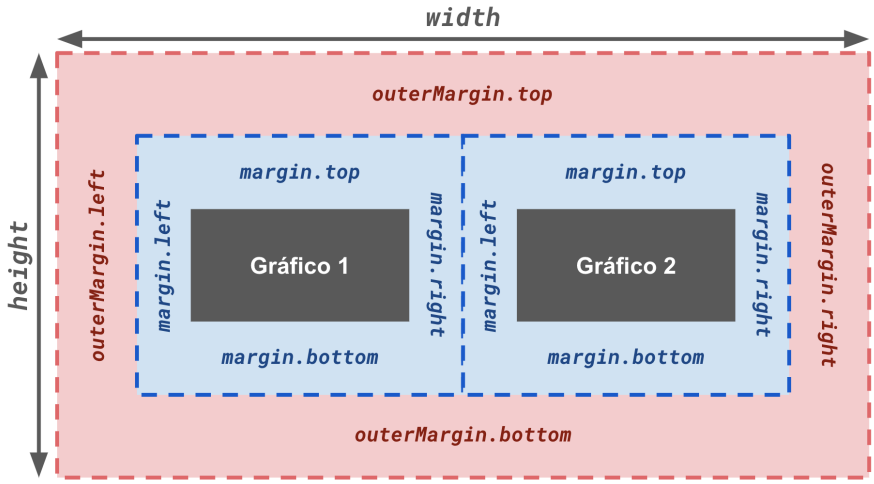


Figura 3.63. Representación visual de la forma en que los márgenes afectan a la dimensión final del gráfico.

En la **Tabla 3.4.2** se muestran los parámetros que permiten configurar las dimensiones y márgenes del gráfico en **jviz**.

Tabla 3.4.2. Parámetros de dimensión del gráfico.

Nombre	Descripción
"$width$"	Almacena la anchura del gráfico, en píxeles. Solo admite un valor numérico.
"$height$"	Almacena la altura del gráfico, en píxeles. Solo admite un valor numérico.
"$margin$"	Almacena los márgenes internos del gráfico, en píxeles. Admite un valor numérico si se desea aplicar el mismo margen a los cuatro lados del

	gráfico, o un objeto con los valores que se desean aplicar a cada lado individualmente: • "top" para especificar el tamaño del margen superior. • "$bottom$" para especificar el tamaño del margen inferior. • "$left$" para especificar el tamaño del margen izquierdo. • "$right$" para especificar el tamaño del margen derecho. En el caso de este campo no se especifique, su valor por defecto es 0 (sin margen). En el caso de que se especifiquen los márgenes de forma individual, pero alguno de los lados no tenga valor asignado, también pasará a valer 0 por defecto.
"$outerMargin$"	Almacena los márgenes externos del gráfico, en píxeles. Admite los mismos valores que la propiedad "$margin$".

Generalmente, aunque la posición en la que se especifiquen estos valores en el esquema no afecta al gráfico generado, se suelen situar siempre al inicio del esquema para ser identificables visualmente de forma rápida.

En el **Código 3.2** se muestra un ejemplo de cómo se especifican estos valores: el gráfico final tendrá una anchura de $400px$ y una altura de

$300px$. La zona de dibujo estará reducida $50px$ a la izquierda y $50px$ en la parte superior del mismo. Notar que como los otros márgenes no han sido especificados, su valor será de $0px$, al igual que los márgenes externos.

```
{
    "width": 400,
    "height": 300,
    "margin": {
        "top": 50,
        "left": 50
    },
    . . .
}
```

Código 3.2. *Fragmento de esquema de* ***jviz*** *en el que se define la dimensión del mismo y los márgenes superior e izquierdo.*

Estos valores pueden ser utilizados en expresiones, escalas y en los parámetros de las *geoms*.

Título

El campo "$title$" permite especificar las propiedades del título del gráfico. El título siempre se sitúa en la parte superior del gráfico, en el área del margen externo. Los parámetros mostrados en la **Tabla 3.4.3** permiten configurar la apariencia y el texto del título.

Tabla 3.4.3. *Parámetros admitidos en la declaración de variables de estado.*

Nombre	Descripción
"$text$"	(**Obligatorio**) Permite especificar el texto del título.
"$align$"	Alineación del texto. Por defecto el texto estará centrado en el gráfico ("$center$"), aunque puede estar alineado a la izquierda ("$left$") o a la derecha ("$right$").
"$fill$"	Color del título.
"$fontSize$"	Tamaño del título.

Temas y estilos

Utilizar un tema u otro no afecta a cómo los datos son representados en el gráfico utilizando las *geoms* o cómo son transformados por medio de las escalas. Los temas sí que ayudan a que el gráfico sea estéticamente agradable o coincida con una guía de estilo existente, sin cambiar las propiedades perceptivas del gráfico.

Los temas se pueden configurar de dos formas:

- Por medio del parámetro "$theme$" del esquema del gráfico, en el que se puede especificar un tema predefinido o especificar un tema propio (**Figura 3.64**).

- Por medio de los parámetros de estilo de las *geoms*, los ejes o las leyendas.

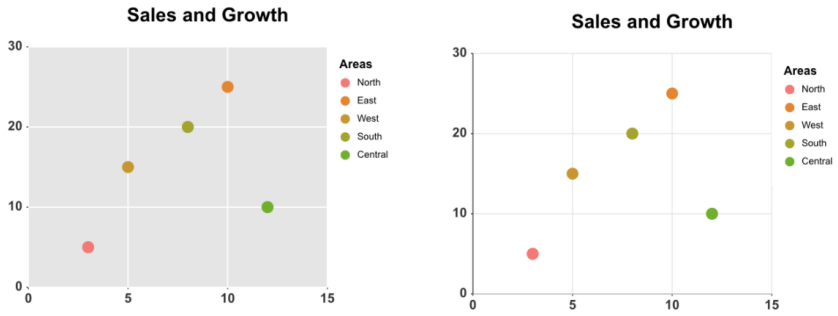


Figura 3.64. *Diferentes temas aplicados a un mismo gráfico: tema por defecto (izquierda) o tema claro (derecha).*

Estas dos formas son complementarias, ya que las propiedades estéticas que no sean proporcionadas por la primera opción pueden ser proporcionadas por la segunda. Además, en el caso de que ambos métodos den valor a una misma propiedad, predomina la especificada como parámetro de estilo en el elemento.

Variables de estado

Dada la propiedad reactiva de los gráficos generados con **jviz**, uno de los elementos importantes del sistema es la presencia de **variables de estado** que permiten almacenar valores que pueden ser utilizados por otros elementos del gráfico (como en las escalas o en las *geoms*), y que son fundamentales para que el gráfico sea reactivo.

Los valores de las variables de estado pueden ser actualizados en cualquier momento y desde diferentes elementos de la aplicación web. Cuando el valor almacenado en una variable de estado cambia, automáticamente se actualizan todos los elementos del gráfico que dependían de dicha variable. Esa dependencia se fija en el archivo de configuración del gráfico de **jviz**.

Las variables de estado se definen en el campo "$state$" del esquema, y son procesadas a continuación de la configuración del gráfico. Los parámetros expuestos en la **Tabla 3.4.4** son necesarios para poder declarar una variable de estado.

***Tabla 3.4.4**. Parámetros admitidos en la declaración de variables de estado.*

Nombre	Descripción
"$name$"	(**Obligatorio**) Permite especificar el nombre de la variable de estado.
"$value$"	Permite especificar el valor inicial de la variable de estado. Si no se especifica, la variable de estado tendrá un valor inicial $null$ (ver tipos de datos en la sección de expresiones).

En el **Código 3.3** se muestra un fragmento de esquema de **jviz** en el que se ha declarado una variable con nombre "$threshold$" y un valor inicial asociado de 0.5. Esta variable, una vez declarada, puede ser utilizada por otros elementos del gráfico, como en escalas o *geoms*.

```
{
    . . .
    "state": [
        {"name": "threshold", "value": 0.5}
    ],
    . . .
}
```

Código 3.3. *Fragmento de esquema de* ***jviz*** *en la que se declara una variable de estado.*

Las variables de estado pueden ser modificadas de dos formas diferentes: **(1)** como respuesta a la interacción del usuario con el gráfico (por medio de eventos) o **(2)** a través de llamadas externas por medio de la API pública del gráfico.

Datos

Los **datos** son la base de la pirámide para la construcción de los gráficos basados en la GOG (**Figura 3.60**). En las aplicaciones orientadas a los análisis exploratorios de datos (EDA), es esencial que los gráficos sigan la filosofía *data-driven* para poder explorar los datos rápidamente desde diferentes puntos de vista, lo que permite la generación de conocimiento mediante la manipulación interactiva de los elementos visuales por parte del usuario.

Siguiendo esta idea, el sistema de representación gráfica utilizada en **jviz** está orientada a la realización de gráficos reactivos y dirigidos por los datos. Los datos son la parte fundamental del sistema y los gráficos

son las diferentes visualizaciones del dato subyacente. Siguiendo los patrones de *clean architecture*, separamos las responsabilidades en cuanto a datos entre: **(1)** gestión; **(2)** análisis; **(3)** representación y **(4)** renderizado. La representación se basa en la lógica del gráfico y el renderizado es la implementación en un soporte específico.

La entrada de datos en **jviz** es flexible y se pueden especificar directamente en el esquema del gráfico (predefinidos o inyectados dinámicamente mediante la API de **jviz**) o proporcionando una URI que apunte a un JSON con el formato correcto. Esos datos se pueden expandir o resumir aplicando transformaciones sobre ellos. Los parámetros necesarios para poder definir conjuntos de datos se muestran en la **Tabla 3.4.5**.

Tabla 3.4.5. *Parámetros admitidos en la declaración de conjuntos de datos.*

Nombre	Descripción
"$name$"	(**Obligatorio**) Permite asignar un nombre al conjunto de datos.
"$value$"	Permite especificar el contenido del conjunto de datos, de forma predefinida. Este parámetro no puede ser utilizado con los parámetros "url" y "$format$".
"url"	Permite especificar una dirección web desde la que jviz debe obtener el contenido del conjunto de datos. Este parámetro requiere también del parámetro "$format$".

"$format$"	Permite especificar el formato para *parsear* los datos obtenidos a través de una dirección web.
"$source$"	Permite definir el nombre de otro conjunto de datos para utilizarlo como origen. Este parámetro es útil cuando se utiliza con el parámetro "$transform$" para generar nuevos datos a partir del original (por ejemplo, generar resúmenes del conjunto de datos original).
"$transform$"	Listado de transformaciones para aplicar en el conjunto de datos.
"$metadata$"	Contiene un diccionario con los elementos como el origen de los datos, la licencia de uso, la descripción de la columna, su formato, propiedades para la ordenación, etc.

Cada conjunto de datos consiste en una lista de elementos (filas), en el que cada elemento puede contener un conjunto de *named attributes* (o columnas).

Por ejemplo, dado el conjunto de datos en formato TSV mostrado en el **Código 3.4** con información de personas (en la que la primera fila corresponde a la cabecera).

```
name    age    city
Bob     27     NY
```

```
Susan   25     SF
. . .
```

Código 3.4. *TSV con información de personas (nombre, edad y ciudad en la que residen).*

Esta tabla debe ser transformada a formato JSON como un array de objetos (**Código 3.5**), en el que cada objeto corresponde a una fila de la tabla, y en el que cada uno de sus campos corresponde a una columna de dicha tabla (donde el nombre del campo ha sido extraído de la cabecera de la tabla).

```
[
    {"name": "Bob", "age": 27, "city": "NY"},
    {"name": "Susan", "age": 25, "city": "SF"},
    . . .
]
```

Código 3.5*. Representación en JSON de los datos mostrados en el TSV.*

A cada uno de los conjuntos de datos se le debe asignar un nombre o identificador único, que servirá para poder referenciar dicho conjunto de datos en el resto de la especificación, utilizando el atributo $"name"$ (**Código 3.6**).

```
{
    . . .
    "data": [{
        "name": "people",
        "value": [
            {"name": "Bobby", "age": 27, "city": "NY"},
```

```
            {"name": "Susan", "age": 25, "city": "SF"},
            . . .
        ]
    }],
    . . .
}
```

***Código 3.6**. Definición de el conjunto de datos de personas en el esquema de **jviz**.*

Expresiones

Para poder realizar cálculos propios durante la generación o actualización del gráfico, **jviz** incorpora su propio lenguaje de expresiones para la evaluación de fórmulas matemáticas. Dicho lenguaje está construido (en esta primera versión) sobre JavaScript, por lo que muchos de los tipos de datos, operaciones o funciones nativas de JavaScript están soportadas. Sin embargo, hay muchas otras que no están soportadas por motivos de seguridad y para evitar comportamientos secundarios no deseados, como por ejemplo las sentencias de control selectivas (*if, else-if o switch*) o las sentencias de control repetitivas (bucles *while* y *for*).

Al ser un lenguaje diseñado con la finalidad única de evaluar expresiones matemáticas, tampoco está soportado la creación y asignación de variables, así como la creación de funciones, clases, listas u objetos.

Tipos de datos simples

Los tipos de datos simples mostrados en la **Tabla 3.4.6** están admitidos dentro de las expresiones.

Tabla 3.4.6. *Tipos de datos simples admitidos en el lenguaje de expresiones de* ***jviz****.*

Tipo de dato	Descripción	Valores admitidos
Tipo ***Number***	Representa un número real de doble precisión.	Cualquier número real.
Tipo ***Bool*** o ***Boolean***	Representa una unidad lógica, y admite únicamente dos valores: $true$ para almacenar un valor cierto o $false$ para almacenar un valor falso.	$true$ o $false$
Tipo ***String***	Permite representar texto utilizando cadenas o secuencias ordenadas de caracteres. Dentro de las expresiones siempre se representan entre comillas simples, para diferenciarlas de las variables y funciones.	Cualquier cadena de caracteres (por ejemplo, $'Hola\ mundo'$).
Tipo ***Null***	Valor nulo o vacío.	$null$

Tipos de datos compuestos

Pese a que no está soportada la creación de tipos de datos compuestos, sí que está permitida su manipulación dentro de las expresiones, utilizando ciertas funciones predefinidas.

Arrays o listas

Un ***array*** es un tipo de dato que representa una colección o listado de elementos **ordenados**, a los cuales se puede acceder utilizando el índice que ocupan dentro de la lista. El índice del primer elemento siempre es el 0, y el índice del último elemento es la longitud del *array* (el número de elementos que contiene) menos 1. Cada uno de los elementos que puede contener un *array* puede ser un tipo de dato simple, otro *array* o un diccionario.

Dentro de las expresiones existen funciones que permiten manipular *arrays*, como por ejemplo acceder a un elemento a partir de su índice, calcular la suma de todos los valores, etc…

Objetos

Un **objeto** (o diccionario) es un tipo de dato que representa una colección de elementos **no ordenados**, y almacenados de la forma $key: value$, de manera que cada elemento tiene asociado un nombre o **campo** a partir del cual podemos acceder al valor de dicho elemento. Al igual que sucedía con los *arrays*, cada uno de los elementos que puede contener un diccionario puede ser un tipo de dato simple, un *array* o incluso otro diccionario anidado.

Operadores aritméticos

Los operadores aritméticos mostrados en la **Tabla 3.4.7** están permitidos dentro de las expresiones de **jviz.**

***Tabla 3.4.7**. Operadores aritméticos admitidos en el lenguaje de expresiones de **jviz**.*

Operador	Descripción	Ejemplo	Resultado
$+$	Realiza la suma o concatenación de dos valores.	$2 + 3$	5
$-$	Realiza la sustracción de dos valores. También se puede utilizar para cambiar el signo de un valor numérico.	$5 - 7$	-2
$*$	Multiplica dos valores.	$5 * 5$	25
$/$	Divide dos valores.	$5/5$	1
$\%$	Devuelve el resto de dividir dos números.	$3\%2$	1

Notar que los operadores de asignación $(=)$, incremento $(++)$ o decremento $(--)$ no están soportados, al igual que tampoco están soportados los operadores de asignación compuesta (como por ejemplo el operador $+=$ o el operador $-=$).

Operadores relacionales

Los **operadores relacionales** o de **comparación** permiten comparar dos cantidades y devuelve un booleano con valor $true$ (verdadero) si la relación es cierta, o un booleano con valor $false$ (falso) si la relación no es cierta. En la **Tabla 3.4.8** se muestra un listado con los operadores relacionales admitidos.

***Tabla 3.4.8**. Listado de operadores relaciones que están soportados en las expresiones de **jviz**.*

Operador	Descripción	Ejemplo	Resultado
$<$	Comprueba si el valor de la izquierda es menor que el valor a la derecha del operador.	$5 < 10$	$true$
$>$	Comprueba si el valor de la izquierda es mayor que el valor a la derecha del operador.	$5 > 10$	$false$
$<=$	Comprueba si el valor de la izquierda es menor o igual que el valor a la derecha del operador.	$1 <= 1$	$true$
$>=$	Comprueba si el valor a la izquierda del	$5 >= 6$	$false$

	operador es mayor o igual que el valor de la derecha.		
==	Comprueba si ambos valores son exactamente iguales.	$"abc" == "def"$	$false$
! =	Comprueba si ambos valores son diferentes.	$"abc" != "def"$	$true$

Operadores lógicos

Los **operadores lógicos** permiten crear condiciones compuestas en una fórmula o expresión. En la **Tabla 3.4.9** se muestran los operadores lógicos soportados en las expresiones de **jviz**.

Tabla 3.4.9. *Tabla de operadores lógicos soportados.*

Operador	Descripción	Ejemplo	Resultado
&&	El operador **AND** devuelve un valor booleano $true$ si y sólo si las dos expresiones a izquierda y a derecha son ciertas.	$(5 < 6)$ && $(4 >= 0)$	$true$

||	El operador **OR** devuelve un valor booleano $true$ si al menos una de las dos expresiones es cierta.	$(0 == 1) \,\|\|\, (5 > 0)$	$true$
!	El operador **NEGACIÓN** invierte el resultado de la expresión a la que acompaña.	$!\,(5 > 0)$	$false$

Orden y prioridad de los operadores

Como en cualquier expresión o fórmula matemática, las expresiones se evalúan siempre de izquierda a derecha, teniendo prioridad las subexpresiones que se encuentren encerradas entre paréntesis en el nivel más interno. Los operadores tienen también una orden de evaluación preestablecida, de manera que, en una expresión con varias operaciones, esta se evaluará por partes y en un orden determinado. En la **Tabla 3.4.10** se presenta, de mayor a menor, el orden de evaluación de los operadores.

***Tabla 3.4.10**. Orden de los operadores, ordenados de mayor a menor orden.*

Operador	Descripción
(y)	En primer lugar, se evaluarán las subexpresiones más internas en un paréntesis.
– y !	En segundo lugar, se evaluarán los operadores para cambiar el signo a un valor o a otra expresión y el operador de negación.
*, / y %	En tercer lugar, se evaluarán los operadores aritméticos de multiplicación, división y módulo.
+ y –	En cuarto lugar, se evaluarán los operadores aritméticos de suma y resta.
<, >, <=, >=	En quinto lugar, se evaluarán los operadores relacionales de mayor y menor.
== y ! =	En sexto lugar se evaluarán los operadores relacionales de igualdad y desigualdad.
&&	En penúltimo lugar se evaluará el operador lógico AND.
| |	En último lugar se evaluará el operador lógico OR.

Notar que si en una expresión coinciden dos o más operadores del mismo nivel, el orden de evaluación es siempre de izquierda a derecha.

Constantes

Los valores constantes que se muestran en la **Tabla 3.4.11** pueden ser utilizados en cualquier expresión de **jviz**.

Tabla 3.4.11. *Listado de constantes predefinidas en el evaluador de expresiones de* ***jviz****.*

Nombre	Descripción
pi	Número pi.
tau	Almacena el doble del número pi (es decir, $2 * \pi$)
e	Número e.
$epsilon$	Almacena el valor numérico más pequeño cercano a 0.

Funciones

Pese a que no está permitida la definición de funciones, **jviz** sí que proporciona un conjunto de funciones predefinidas en el evaluador de expresiones, de manera que pueden ser llamadas en cualquier expresión.

Funciones matemáticas

En la **Tabla 3.4.12** se muestra un listado con las funciones matemáticas disponibles en **jviz**.

Tabla 3.4.12. *Listado de funciones matemáticas.*

Función	Descripción
$abs(x)$	Devuelve el valor absoluto del valor pasado como argumento.
$ceil(x)$	Devuelve el mayor número entero más cercano al valor proporcionado como argumento.
$clamp(x, min, max)$	Restringe el valor proporcionado dentro del intervalo $[min, max]$.
$cos(x)$	Devuelve el coseno del valor pasado como argumento.
$exp(x)$	Devuelve el resultado de calcular e^{x}.
$floor(x)$	Devuelve la parte entera del valor pasado como argumento.
$ln(x)$	Devuelve el logaritmo neperiano de x.
$log(x, base)$	Devuelve el logaritmo de x en base $base$.
$log2(x)$	Devuelve el logaritmo de x en base 2. Alias de $log(x, 2)$.
$log10(x)$	Devuelve el logaritmo de x en base 10. Alias de $log(x, 10)$.
$max(x1, x2, \dots)$	Devuelve el mayor de los valores pasados como argumento.

$min(x1, x2, \dots)$	Devuelve el menor de los valores pasados como argumento.
$polarX(r, \theta, centerX)$	Devuelve la componente x de la transformación a coordenadas cartesianas de la coordenada polar proporcionada como argumento. El tercer argumento corresponde con la componente x del centro de coordenadas (por defecto 0).
$polarY(r, \theta, centerY)$	Devuelve la componente y de la transformación a coordenadas cartesianas de la coordenada polar proporcionada como argumento. El tercer argumento corresponde con la componente y del centro de coordenadas (por defecto 0).
$random()$	Devuelve un número aleatorio dentro del rango $[0, 1]$.
$random(max)$	Devuelve un número aleatorio dentro del rango $[0, max]$.
$random(min, max)$	Devuelve un número aleatorio dentro del rango $[min, max]$.
$sin(x)$	Devuelve el seno del valor pasado como argumento.
$sqrt(x)$	Devuelve la raíz cuadrada del valor pasado como argumento.

Funciones para manipular arrays

Las funciones mostradas en la **Tabla 3.4.13** permiten la manipulación de *arrays*.

Tabla 3.4.13. *Funciones para la manipulación de arrays.*

Función	Descripción
$average(array)$	Calcula el valor promedio de los elementos del array.
$indexOf(array, valor)$	Devuelve el primer índice en el que se encuentra el valor proporcionado dentro del array. Si dicho valor no se encuentra dentro del array, esta función devuelve -1.
$length(array)$	Devuelve el número de elementos del array.
$range(array)$	Devuelve un array con el rango $[min, max]$ en el que se encuentran todos los valores del array pasado como argumento.
$sum(array)$	Devuelve la suma de todos los valores del array.
$valueOf(array, pos)$	Devuelve el valor que se encuentra en el índice proporcionado.

Funciones para manipular objetos

En la **Tabla 3.4.14** se pueden observar las funciones que permiten la manipulación de objetos dentro de las expresiones.

Tabla 3.4.14. *Listado de funciones para la manipulación de objetos dentro de las expresiones.*

Función	Descripción
$nest(dict, camino)$	Devuelve el último valor tras recorrer el objeto siguiendo el camino proporcionado.
$valueOf(dict, campo)$	Devuelve el valor del campo dentro del objeto.

Funciones para manipular cadenas de caracteres

Estas funciones permiten la manipulación de cadenas de caracteres o *strings*. En la **Tabla 3.4.15** se muestran todas estas funciones.

Tabla 3.4.15. *Listado de funciones para la manipulación de strings o cadenas de caracteres.*

Función	Descripción
$camelcase(str)$	Devuelve la cadena original en formato *camelcase*, en el que se eliminan las separaciones (espacios en blanco) entre las palabras, y la primera letra de cada palabra

	de la cadena se cambia a mayúscula, a excepción de la primera palabra de la frase que se mantiene en minúscula.
$capitalize(str)$	Devuelve la misma cadena con el primer caracter convertido a mayúsculas.
$kebabcase(str)$	Devuelve la cadena original en formato *kebabcase*, en el que todos los caracteres de la cadena se cambian a minúscula y las separaciones entre las palabras se cambian por un guión alto.
$length(str)$	Devuelve el número de caracteres de la cadena.
$repeat(str, num)$	Devuelve una nueva cadena en la que la cadena original está repetida tantas veces como se indique.
$snakecase(str)$	Devuelve la cadena original en formato *snakecase*, en el que todos los caracteres de la cadena se cambian a minúscula y las separaciones entre las palabras se cambian por un guión bajo.
$timestamp(str, fecha)$	Permite generar una cadena de caracteres que represente la fecha proporcionada, utilizando el formato especificado como cadena de caracteres en el primer

	argumento de la función. En el caso de que no se proporcione una fecha, se utilizará la fecha actual en el momento de la ejecución de la función.
$truncate(str, long)$	Recorta la cadena de caracteres si es mayor que la longitud proporcionada.

Otras funciones

En la **Tabla 3.4.16** se muestra un listado de otros tipos de funciones que están también disponibles en las expresiones de **jviz**.

***Tabla 3.4.16**. Otros tipos de funciones disponibles en las expresiones de **jviz**.*

Función	Descripción
$isArray(valor)$	Devuelve un booleano $true$ si el valor proporcionado es un array, $false$ en caso contrario.
$isBool(valor)$	Devuelve un booleano $true$ si el valor proporcionado es un booleano, $false$ en caso contrario.
$isNull(valor)$	Devuelve un booleano $true$ si el valor proporcionado es $null$, $false$ en caso contrario.

$isNumber(valor)$	Devuelve un booleano $true$ si el valor proporcionado es un número, $false$ en caso contrario.
$isObject(valor)$	Devuelve un booleano $true$ si el valor proporcionado es un diccionario, $false$ en caso contrario.
$isString(valor)$	Devuelve un booleano $true$ si el valor proporcionado es una cadena de caracteres, $false$ en caso contrario.
$isUndef(valor)$	Devuelve un booleano $true$ si la variable o valor proporcionado no existe, $false$ en caso de que sí que exista.
$isValid(valor)$	Devuelve un booleano $true$ si el valor proporcionado existe y es un valor no nulo ($null$), $false$ en caso contrario.

Condicionales

Como se ha explicado al inicio de esta sección, las sentencias de control selectivas no están soportadas dentro del lenguaje de expresiones. Sin embargo, dado que existen muchas ocasiones en las que el resultado de una expresión viene dado por si se cumple una condición o no, sí que está implementada una función especial $if(condicion, valorCierto, valorFalso)$, conocida en lenguajes de

programación también como "operador ternario", que simula el comportamiento de una sentencia condicional $if:else$.

Esta función admite los siguientes tres argumentos:

- El primer argumento es la condición que se evaluará (cuyo resultado debe ser un valor booleano).
- El segundo argumento es el valor que devolverá si la condición del primer argumento de la función es cierta.
- Por último, el tercer argumento es el valor que devolverá si la condición del primer argumento de la función es falsa.

Variables y funciones en tiempo de ejecución

Las siguientes variables y funciones especiales permiten el acceso a otras especificaciones de la visualización, tales como escalas o variables de estado. En la **Tabla 3.4.17** se muestra un listado con todas las variables y funciones que dependen del estado del gráfico.

Tabla 3.4.17. *Variables y funciones disponibles en tiempo de ejecución.*

Nombre	Descripción
$state$	Objeto que almacena el valor de cada una de las variables de estado definidas en el esquema del gráfico.
$scale(nombre, valor)$	Permite aplicar la escala definida con el nombre pasado como primer argumento, al valor pasado como segundo argumento. Si

	la escala no existe, esta función devuelve un valor $null$.
$invert(nombre, valor)$	Permite aplicar la inversa de la escala con nombre pasado en el primer parámetro, al valor pasado como segundo parámetro. Esta función sólo está disponible para escalas de tipo continuas.
$draw.margin$	Variable que almacena un objeto con cuatro márgenes internos del gráfico (top, $bottom$, $left$ y $right$).
$draw.outerMargin$	Variable que almacena un objeto con cuatro márgenes externos del gráfico (top, $bottom$, $left$ y $right$).
$draw.width$	Variable que almacena la anchura disponible para el gráfico, en píxeles.
$draw.height$	Variable que almacena la altura disponible para el gráfico, en píxeles.

En el caso de que la expresión está siendo utilizada en una **transformación** o como atributo de una *geom*, en adición a las variables están definidas de forma automática también estarán definidas las variables que se listan en la **Tabla 3.4.18**.

Tabla 3.4.18. Tabla con las variables que estarán disponibles en la expresión si esta es utilizada dentro de una transformación o de una geom.

Nombre	Descripción
$datum$	Contiene el dato sobre el que se está realizando la transformación o representación. Cada uno de los campos puede ser accedido utilizando la notación de punto (por ejemplo $datum.campo$) o utilizando la función $valueOf$ (por ejemplo, $valueOf(datum, 'campo')$.
$index$	Almacena el índice asociado al dato $datum$ dentro del conjunto de datos específico. El índice toma valores enteros dentro del intervalo $[0, data.length - 1]$.

Si por el contrario la expresión está siendo utilizada dentro de un evento, además de las dos variables de la Tabla 3.4.18 también estará definida una variable especial llamada $\boldsymbol{event}$, que almacena un objeto con los campos que se muestran en la **Tabla 3.4.19**.

*Tabla 3.4.19. Listado de propiedades definidas dentro de la variable **event**.*

Nombre	Descripción
$event.x$	Almacena la coordenada x (en píxeles) del punto en el que se ha realizado el evento.

$event.y$	Almacena la coordenada y (en píxeles) del punto en el que se ha realizado el evento.

Transformaciones

Las **transformaciones** son operaciones que se aplican sobre los datos. Cuando se realiza una transformación, se aplica sobre cada uno de los elementos del conjunto de datos. Las transformaciones permiten fundamentalmente:

- Realizar agregaciones o sumarios de los datos.
- Realizar transformaciones en uno o varios campos.
- Computar nuevos campos.
- Ordenar los datos a partir de uno o varios campos.
- Generar nuevos conjuntos de datos, a partir de datos simulados o a partir de otros conjuntos de datos existentes.

Las transformaciones se pueden utilizar en cada uno de los conjuntos de datos o previo a la representación de las figuras geométricas, siempre a partir del atributo "$transform$". En cada transformación, existe un parámetro común que es el que se especifica a partir del atributo "$type$" y que sirve para especificar el tipo de transformación a aplicar. El resto de parámetros necesarios dependerá de cada transformación en particular. En la **Tabla 3.4.20** se muestran todas las transformaciones disponibles en **jviz**.

*Tabla 3.4.20. Transformaciones soportadas en **jviz**.*

Nombre	Descripción
Bin	Discretiza valores numéricos en un conjunto de *bines*.
Dotbin	Calcula las posiciones de los bines para apilar puntos, basándose en el algoritmo de densidad de puntos (Wilkinson, 1999).
Filter	Genera un nuevo conjunto de datos filtrando los elementos del conjunto de datos de origen a partir de una expresión dada.
Formula	Computa un nuevo campo en cada uno de los elementos del conjunto de datos a partir de una fórmula (expresión).
Identifier	Asigna un identificador único (índice) a cada uno de los elementos del conjunto de datos.
Link	Genera un camino visual (*path*) entre dos nodos.
Range	Computa el valor mínimo y máximo de un campo del conjunto de datos.
Rename	Esta transformación permite seleccionar campos en los elementos de un conjunto de datos y guardar su valor asociado en otro campo con un nombre diferente.

Sort	Ordena el conjunto de datos a partir del valor de uno o varios campos.
Spacing	Permite distribuir un conjunto de datos en el espacio de dibujo, aplicando diferentes métodos de distribución.
Stack	Genera grupos apilados a partir de los valores del conjunto de datos.
Stratify	Esta transformación construye un árbol jerárquico a partir del conjunto de datos en el que se aplica.
Summary	Esta transformación permite realizar cálculos sumarios al conjunto de datos, pudiendo añadir los cálculos realizados al propio conjunto de datos o generar uno nuevo.

En el **Código 3.7** se muestra un ejemplo en el que aplican dos transformaciones a diferentes elementos del esquema: la primera se está aplicando a un conjunto de datos mientras que la segunda se aplica a una *geom* de tipo rectángulo.

En la primera transformación al conjunto de datos con nombre "$table$", se está realizando un apilamiento de los mismos agrupados por el valor de uno de los campos (en el ejemplo el campo "key"). El resultado de esta transformación se añade a cada uno de los elementos del conjunto de datos, en un nuevo campo llamado "$value$". El conjunto de datos resultante de aplicar esta transformación es el

que se guardará con el nombre "$table$".

La segunda transformación está realizando un filtrado al mismo conjunto de datos, manteniendo sólo aquellos elementos cuyo campo "$visible$" tenga un valor booleano true. Esta transformación no modifica el conjunto de datos original, sino que el nuevo conjunto de datos que se genera con los elementos filtrados es el que se utiliza para generar los rectángulos.

```
{
    . . .
    "data": [{
        "name": "table",
        "url": "/data/table.json",
        "transform": [{
            "type": "stack",
            "align": "default",
            "groupby": "key",
            "field": "value"
        }]
    }],
    . . .
    "geoms": [{
        "type": "rectangle",
        "source": {"data": "table"},
        "transform": [{
            "type": "filter",
            "expr": "datum.visible == true"
        }],
        . . .
    }]
}
```

Código 3.7. *Ejemplo de dos transformaciones: la primera aplicada a un conjunto de datos y la segunda aplicada a una geom.*

Cabe destacar que se pueden especificar varias transformaciones sobre un mismo conjunto de datos. Las transformaciones se aplican de manera secuencial, esto es, el conjunto de datos resultante de aplicar una transformación es el conjunto de datos de entrada para la siguiente transformación.

Transformación bin

La transformación ***bin*** discretiza valores numéricos en un conjunto de *bines* o contenedores, utilizando para ello los parámetros mostrados en la **Tabla 3.4.21**. Un caso de uso común de esta transformación es la creación de histogramas.

Tabla 3.4.21. *Parámetros de la transformación* ***bin****.*

Nombre	Descripción
"$field$"	(**Obligatorio**) Campo del conjunto de datos que utilizará la transformación para generar los *bines*.
"$domain$"	Intervalo con el dominio en el que se encuentran los valores. Si no se proporciona, se generará automáticamente tomando el menor y el mayor de todos los valores del campo especificado en el parámetro "$field$".

"$step$"	Tamaño exacto de cada uno de los *bines* generados. Si se utiliza, se ignorarán otros parámetros como "$bins$","$maxbins$" o "$minbins$".
"$bins$"	Establece el número exacto de *bines* en los que dividir el dominio. Si se utiliza, se ignorarán otros parámetros como "$step$","$maxbins$" o "$minbins$".
"$minbins$"	Establece el número mínimo de *bines* que se crearán.
"$maxbins$"	Establece el número máximo de *bines* que se crearán.
"as"	Listado con los nombres de los campos que se crearán en cada uno de los elementos del conjunto de datos para almacenar la información de los bines generados, y cuyo valor por defecto es ["$binIndex$","$binStart$","$binEnd$"].

Transformación dotbin

La transformación ***dotbin*** calcula las posiciones de los *bines* para apilar puntos, comúnmente utilizado para la generación de diagramas de puntos. Esta transformación está basada en el algoritmo de "densidad de puntos" (Wilkinson, 1999). El listado de parámetros para poder aplicar esta transformación se muestra en la **Tabla 3.4.22**.

Tabla 3.4.22. *Parámetros de la transformación* ***dotbin****.*

Nombre	Descripción
"$field$"	(**Obligatorio**) Campo del conjunto de datos que utilizará la transformación para generar los *bines*.
"$groupby$"	Nombres de los campos de datos para generar una agrupación por su valor. Si no se especifica, se utilizará un solo grupo que contiene todos los elementos del conjunto de datos.
"$step$"	Tamaño exacto de cada uno de los *bines* generados.
"as"	Nombre del campo en el que se guardará el índice del *bin* al que pertenece cada uno de los elementos del conjunto de datos.

Transformación filter

Esta transformación filtra los elementos del conjunto de datos de origen a partir de una expresión dada. Cada elemento (fila) del conjunto de datos es evaluado mediante dicha expresión, y si el resultado devuelto es un valor $true$, entonces dicho elemento se mantiene en el conjunto de datos. En la **Tabla 3.4.23** se muestra el listado de parámetros que admite esta transformación.

Tabla 3.4.23. *Parámetros de la transformación* ***filter****.*

Nombre	Descripción

"$expr$"	(**Obligatorio**) Expresión que debe cumplir cada uno de los elementos del conjunto de datos para ser incluidos en el nuevo conjunto de datos que se genere como resultado de aplicar esta transformación.

Transformación fórmula

Esta transformación permite computar un nuevo campo para cada elemento del conjunto de datos. El cómputo del nuevo campo se realiza a partir de una expresión data: el resultado de dicha expresión es guardado en un nuevo campo de cada elemento del conjunto de datos. La **Tabla 3.4.24** lista los parámetros requeridos para poder aplicar esta transformación.

Tabla 3.4.24. *Parámetros de la transformación* ***formula****.*

Nombre	Descripción
"$expr$"	(**Obligatorio**) Expresión con la que se generará el valor del nuevo campo del conjunto de datos.
"as"	(**Obligatorio**) Establece el nombre del campo en el que se guardará el nuevo valor obtenido a partir de la evaluación de la expresión dada para cada elemento del conjunto de datos. Si el campo no existe en cada elemento del conjunto de datos, se

creará, y si ya existe, se sobrescribirá su contenido.

Transformación identifier

Esta transformación extiende del conjunto de datos de origen asignando un identificador único para cada elemento del conjunto de datos basado en el índice que ocupa dicho elemento. La **Tabla 3.4.25** proporciona los parámetros que esta transformación necesita para poder ser aplicada.

Tabla 3.4.25. *Parámetros de la transformación* ***identifier****.*

Nombre	Descripción
"as"	(**Obligatorio**) Nombre del campo en el cual se guardará el identificador.

Transformación range

Esta transformación computa los valores máximo y mínimo de un campo del conjunto de datos. La tupla generada se almacena en una variable de estado, especificada en los parámetros de la transformación. Los parámetros mostrados en la **Tabla 3.4.26** son necesarios para poder aplicar esta transformación.

Tabla 3.4.26. *Parámetros de la transformación* ***range****.*

Nombre	Descripción
"$field$"	(**Obligatorio**) Campo del conjunto de datos al que se desea computar los valores mínimo y máximo.
"$state$"	(**Obligatorio**) Nombre de la variable de estado en la que se almacenará el rango generado.

Transformación rename

Esta transformación permite seleccionar campos en los elementos de un conjunto de datos y guardar su valor asociado en otro campo con un nombre diferente. Esta transformación necesita los parámetros mostrados en la **Tabla 3.4.27**.

Tabla 3.4.27. *Parámetros de la transformación* ***rename****.*

Nombre	Descripción
"$fields$"	(**Obligatorio**) Permite definir los campos de cada elemento del conjunto de datos cuyos valores serán copiados en otro campo con diferente nombre. Esta opción admite un listado de nombres, en el que cada nombre debe ser un campo existente en cada elemento del conjunto de datos.
"as"	(**Obligatorio**) Listado con los nombres de los nuevos campos en los que se guardarán los valores que

tenían los campos definidos a través de la opción "$fields$".

Transformación sequence

Esta transformación construye un nuevo conjunto de datos a partir de una secuencia de valores numéricos ordenados. Dicha secuencia se construye a partir de tres valores: valor inicial de la secuencia (*start*), valor final de la secuencia (*end*) y paso (*step*). La **Tabla 3.4.28** muestra los parámetros que admite esta transformación.

Tabla 3.4.28. *Parámetros de la transformación* ***sequence****.*

Nombre	Descripción
"as"	Nombre del campo en el que cada valor de la secuencia se guardará.
"$start$"	Valor inicial de la secuencia. Si no se especifica se tomará 0 como valor inicial.
"end"	(**Obligatorio**) Valor final de la secuencia.
"$step$"	Distancia entre cada valor de la secuencia. Si no se especifica se tomará 1 como paso.

Notar que la secuencia siempre empieza en el valor inicial, pero el valor final puede no estar incluido en la secuencia de valores

generados. Por ejemplo, si $start = 0$, $end = 10$ y $step = 3$, el listado de valores sería $[0,3,6,9]$, que no incluye al valor final 10.

Transformación sort

Esta transformación ordena los elementos del conjunto de datos a partir del valor de uno o varios campos, de forma alfanumérica. Así por ejemplo, si un elemento del conjunto de datos tiene el valor "$hola$" en el campo especificado para ordenar, y otro elemento tiene el valor "$mundo$" en dicho campo, al comparar ambas cadenas se tendría que "$hola$" $<$ "$mundo$", por lo que el primer elemento iría delante del segundo. De momento no se controla en el ordenamiento la presencia de mayúsculas y minúsculas y se delega su valor en el ordenamiento al locale usado por el lenguaje de programación que implemente estas especificaciones.

La **Tabla 3.4.29** muestra los parámetros necesarios para poder aplicar esta transformación.

Tabla 3.4.29. *Parámetros de la transformación* ***sort****.*

Nombre	Descripción
"$fields$"	(**Obligatorio**) Listado de campos a utilizar para computar el nuevo orden de los elementos en el conjunto de datos.
"$order$"	Listado con el orden para cada uno de los campos. Los valores admitidos son: "asc": orden ascendente (por defecto).

"$desc$": orden descendiente.

Este listado debe tener la misma longitud que el listado de campos especificados en el parámetro $fields$.

Este campo es opcional, por lo que de no especificarse se generará automáticamente un listado de "asc", tantos como campos se hayan especificado en el parámetro $fields$.

Transformación spacing

Esta transformación permite distribuir un conjunto de datos en el espacio de dibujo, aplicando diferentes métodos de distribución. Un ejemplo práctico en la cual esta transformación sería necesaria es para realizar la distribución de valores en un gráfico de lollipops, en que las circunferencias se distribuyen en una dimensión sin que se solapen unas con otras, y de forma que su distancia a su valor real sea el mínimo posible. En la **Tabla 3.4.30** se muestran los parámetros aceptados para esta transformación.

Tabla 3.4.30. *Parámetros de la transformación* ***spacing****.*

Nombre	Descripción
"$method$"	(**Obligatorio**) Permite establecer el método de distribución de los valores en el espacio disponible.

	Existen dos métodos de distribuciones posibles: de forma proporcional o de forma no solapante.
"$scale$"	(**Obligatorio**) Permite establecer la escala que se aplica a los valores previamente a su distribución. El espacio en el que se distribuirán los valores viene determinado por el rango de la escala.
"$fields$"	(**Obligatorio**) Permite establecer el campo del conjunto de datos del cual se deben extraer los valores para distribuir.
"as"	(**Obligatorio**) Permite establecer el nombre del nuevo campo en el que se guardará el nuevo valor una vez aplicada la distribución.
"$separation$"	Permite definir el valor mínimo de separación entre los valores. Este parámetro es obligatorio para la distribución no solapante.

Transformación *summary*

Esta transformación permite realizar cálculos sumarios al conjunto de datos, pudiendo añadir los cálculos realizados al propio conjunto de datos o generar uno nuevo. La **Tabla 3.4.31** muestra los parámetros que necesita esta transformación para poder ser aplicada, mientras que en la **Tabla 3.4.32** muestra las operaciones soportadas en esta transformación.

Esta transformación puede utilizarse por ejemplo para calcular los valores máximos y mínimos de un campo, calcular percentiles, etc…

Tabla 3.4.31. *Parámetros de la transformación* ***summary****.*

Nombre	Descripción
"$groupby$"	Nombres de los campos de datos para generar una agrupación por su valor. Si no se especifica, se utilizará un solo grupo que contiene todos los elementos del conjunto de datos.
"$join$"	Permite especificar si, tras realizar las agrupaciones y las operaciones sumarias, los resultados formarán un nuevo conjunto de datos independiente, o si por el contrario estos resultados serán añadidos a los elementos del conjunto de datos ya existente (es decir, se extenderá el conjunto de datos con los resultados sumarios). Por defecto tiene el valor $false$.
"$fields$"	(**Obligatorio**) Permite especificar los campos para los cuales se va a realizar las operaciones sumarias.
"op"	(**Obligatorio**) Permite especificar las operaciones sumarias a realizar para cada uno de los campos
"as"	(**Obligatorio**) Permite especificar el nombre del campo en el que guardará el resultado de aplicar las operaciones sumarias

*Tabla 3.4.32. Operaciones admitidas para la transformación **summary**.*

Nombre	Descripción
"$first$"	Devuelve el primer valor de la agregación.
"$last$"	Devuelve el último valor de la agregación.
"min"	Calcula el valor mínimo de todos los valores de la agregación.
"max"	Calcula el valor máximo de todos los valores de la agregación.
"$q1$"	Calcula el cuantil 0.25 a partir de los valores de la agregación.
"$q3$"	Calcula el cuantil 0.75 a partir de los valores de la agregación.
"$median$"	Calcula el cuantil 0.5 a partir de los valores de la agregación.
"$mean$"	Calcula el promedio de los valores de la agregación.
"sum"	Calcula la suma de todos los valores de la agregación.
"$count$"	Devuelve el número de valores de la agregación.

Paneles

Los **paneles** permiten la división del área de representación del gráfico en paneles independientes, siguiendo una distribución proporcionada en el esquema. Los paneles son especialmente útiles por ejemplo para permitir la representación de varios gráficos en uno solo, de forma que comparten los elementos que lo componen (datos, escalas, etc..), pero son representados en compartimentos de la imagen diferentes e independientes.

La distribución de paneles se puede definir de dos formas: **(1)** utilizando una matriz, en la que se realiza la distribución de los paneles contiguos; **(2)** proporcionando el número de filas y columnas en los que se debe hacer la partición en paneles del gráfico. En la **Tabla 3.4.33** se muestra los parámetros para definir la distribución de paneles en el esquema.

Tabla 3.4.33. *Parámetros para definir los paneles del gráfico.*

Nombre	Descripción
"$matrix$"	Permite definir la matriz con la distribución de los paneles. Este parámetro es incompatible con los parámetros "$rows$" y "$columns$"
"$rows$"	Permite definir el número de filas para realizar la distribución de paneles. Debe utilizarse junto con el parámetro "$columns$".

"$columns$"	Permite definir el número de columnas para realizar la distribución de paneles. Debe utilizarse junto con el parámetro "$rows$".

En la definición de cada geom y de cada eje se puede especificar en qué panel o conjunto de paneles se debe representar dicho elemento, por medio de un **selector** (que está inspirado en el selector ***:nth-child*** de CSS[39]). Este selector permite seleccionar uno o más paneles basándose en su índice, admitiendo el índice exacto al que aplicar o una fórmula del tipo $an + b$, que especifica el patrón que debe cumplir el índice para que el panel sea seleccionado.

Escalas

Una **escala** transforma un valor *abstracto* procedente de los datos a una propiedad visual del gráfico (por ejemplo, a un tamaño o posición en píxeles, o un color). Las escalas son vitales para proporcionar herramientas que ayudan a una mejor lectura del gráfico, como son los ejes y las leyendas.

Las escalas se pueden dividir en dos grupos, dependiendo de la transformación que se vaya a aplicar y del tipo de datos del dominio:

[39] Especificación del selector **:nth-child()** de CSS: https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child

escalas **continuas** y escalas **discretas**. En la **Tabla 3.4.34** se muestran todas las escalas definidas en **jviz**.

Tabla 3.4.34. *Escalas definidas en* ***jviz****.*

Nombre	Tipo	Descripción
Lineal	Continua	Transforma un valor del dominio en otro del rango de forma lineal (proporcional).
Logarítmica	Continua	Aplica una transformación logarítmica a los valores del dominio antes de que sean transformados a los valores del rango.
Exponencial	Continua	Aplica una transformación exponencial a los valores del dominio antes de que sean transformados a los valores del rango.
Color	Continua	Transforma los valores del dominio a un rango de colores.
Categórica	Discreta	Transforma cada uno de los valores del dominio a un valor del rango basándose en la posición.
Intervalo	Discreta	Realiza una división del rango en intervalos de igual tamaño y asigna a

		cada valor del dominio a un único intervalo del rango.
Punto	Discreta	Similar a la escala intervalo con la diferencia de que el rango es dividido en puntos equidistantes.

La **Tabla 3.4.35** muestra los parámetros necesarios para poder declarar una escala en la gramática de **jviz**.

Tabla 3.4.35. *Parámetros para la definición de una escala en* ***jviz****.*

Nombre	Descripción
"$type$"	(**Obligatorio**) Permite especificar el tipo de escala.
"$name$"	(**Obligatorio**) Permite asignar un nombre a la escala. Este parámetro es necesario para poder instanciar y utilizar la escala en transformaciones, geoms y ejes.
"$domain$"	(**Obligatorio**) Permite especificar el dominio de la escala.
"$range$"	(**Obligatorio**) Permite especificar el rango de la escala.
"$zero$"	Este parámetro acepta un valor booleano, y obliga a que el dominio siempre incluya el valor 0, modificándolo de forma necesaria para que esta

	condición se cumpla. Solo funciona para las escalas del grupo de las continuas. El valor por defecto de este parámetro es $false$.

Ejes

Los **ejes** son una forma de representar gráficamente una escala, permitiendo asociar categorías o cantidades con los elementos que se muestran en un gráfico. El eje de un gráfico está compuesto por los siguientes elementos: **(1)** una línea que permite conocer la envergadura del eje; **(2)** los valores o *labels* que se muestran en los ejes; **(3)** una marca o *tick* en la línea por cada uno de los valores que se representan en el eje; **(4)** una rejilla o *grid*.

En la **Tabla 3.4.36** se muestran los parámetros generales que deben ser proporcionados para poder declarar un eje.

Tabla 3.4.36. *Parámetros generales de un eje.*

Nombre	Descripción
"$scale$"	Escala a la cual el eje se ajustará, de forma que se utilizará los valores del rango de la escala para conocer el inicio y el final del eje.
"$position$"	Posición del eje con respecto a la zona de dibujo. Admite uno de los siguientes valores: "$left$", "$right$", "top" o "$bottom$".

"$values$"	Listado de valores a representar en el eje. Si no se proporciona, se obtendrá a partir del dominio de la escala a la que está asignado este eje.
"$panel$"	Panel en el que el eje debe ser representado.

La **Tabla 3.4.37** muestra los parámetros para poder configurar la línea que delimita el eje.

***Tabla 3.4.37**. Parámetros que permiten configurar la línea de un eje.*

Nombre	Descripción
"$line$"	*Booleano* indicando si la línea debe formar parte del eje o no.
"$lineColor$"	Color de la línea.
"$lineWidth$"	Anchura de la línea.
"$lineOpacity$"	Opacidad de la línea (0 totalmente transparente, 1 totalmente opaca).

En la **Tabla 3.4.38** se muestran los parámetros que permiten configurar el texto de los valores representados como parte del eje.

***Tabla 3.4.38.** Parámetros que permiten configurar el texto de cada uno de los valores mostrados en el eje.*

Nombre	Descripción
"$label$"	*Booleano* indicando si los valores deben ser incluidos como parte del eje.
"$labelColor$"	Color del texto asociado a cada valor mostrado en el eje.
"$labelSize$"	Tamaño de la fuente del texto asociado a cada valor mostrado en el eje.
"$labelOpacity$"	Opacidad del texto asociado a cada valor mostrado en el eje.
"$labelRotation$"	Ángulo de rotación del texto asociado a cada valor mostrado en el eje, en grados.
"$labelBaseline$"	Posición del texto en relación con la línea base sobre la que se asientan la mayoría de las letras.
"$labelAnchor$"	Alineación del texto.
"$labelOffset$"	*Booleano* para indicar si se debe añadir una pequeña muesca entre el texto del tick y su posición en la línea del eje.

En la **Tabla 3.4.39** se muestran los parámetros que permiten configurar la gradilla asociada al eje.

Tabla 3.4.39. *Parámetros que permiten configurar la gradilla asociada al eje.*

Nombre	Descripción
"$grid$"	*Booleano* indicando si la gradilla debe formar parte del eje.
"$gridColor$"	Color de cada una de las líneas de la gradilla.
"$gridWidth$"	Anchura de cada una de las líneas de la gradilla.
"$gridOpacity$"	Opacidad de cada una de las líneas de la gradilla (0 totalmente transparente, 1 totalmente opaca).

Leyendas

Al igual que un eje es una representación gráfica de una escala, las **leyendas** permiten visualizar e identificar las asignaciones realizadas por una escala a diferentes valores visuales, generalmente a colores, tamaños o formas. Comúnmente las leyendas suelen situarse en la parte derecha del gráfico, aunque también pueden situarse en la parte izquierda o inferior.

Las leyendas pueden categorizarse en dos tipos, dependiendo del tipo de escala a la que están asignadas: leyendas continuas y leyendas discretas. Las **leyendas continuas** permiten visualizar gráficamente la asignación de un intervalo a un gradiente de color, aplicada por una escala de color (**Figura 3.65**, leyenda 1). Por lo tanto, las leyendas continuas únicamente pueden ser asignadas a las escalas que sean de tipo color. Por otro lado, **las leyendas discretas** se utilizan en la

asignación de valores discretos, generalmente el realizado por una escala de tipo categórica (**Figura 3.65**, leyendas 2, 3 y 4).

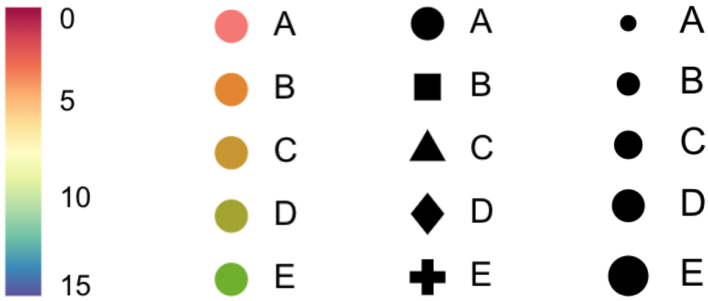


Figura 3.65. *Diferentes tipos de leyendas. De izquierda a derecha: (1) leyenda continua utilizando la paleta de colores espectral; (2) leyenda discreta a la que se le asigna a cada categoría un color; (3) leyenda continua a la que a cada categoría se le asigna un glifo diferente; (4) leyenda continua a la que a cada categoría se le asigna un tamaño.*

En la **Tabla 3.4.40** se muestran los parámetros comunes para poder definir una leyenda, ya sea continua o discreta.

Tabla 3.4.40. *Parámetros comunes para cualquier tipo de leyenda.*

Nombre	Descripción
"$position$"	(Obligatorio) Posición de la leyenda en el gráfico.
"$type$"	(Obligatorio) Tipo de la leyenda: "$glyph$"para leyendas de tipo discreta o "$gradient$" para leyendas de tipo continua.

"$scale$"	(Obligatorio) Escala a la que está asociada esta leyenda.
"$orientation$"	Orientación de la leyenda (horizontal o vertical).
"$fill$"	Color de fondo del recuadro de la leyenda.
"$stroke$"	Color del borde del recuadro de la leyenda.
"$strokeWidth$"	Tamaño del borde del recuadro de la leyenda.
"$padding$"	Relleno interno del recuadro de la leyenda. Es el espacio entre el borde y el contenido de la leyenda.
"$radius$"	Radio del borde del recuadro de la leyenda.
"$title$"	Texto a mostrar como título de la leyenda.
"$titleSize$"	Tamaño del título de la leyenda.
"$titleFill$"	Color del título de la leyenda.
"$labelSize$"	Tamaño del texto que acompaña a cada uno de los valores mostrados en la leyenda.
"$labelFill$"	Color del texto que acompaña a cada uno de los valores mostrados en la leyenda.

En la **Tabla 3.4.41** se muestran los parámetros necesarios para definir una leyenda de tipo continua.

Tabla 3.4.41. *Parámetros para una leyenda de tipo continua.*

Nombre	Descripción
"$gradientSize$"	Anchura del área del gradiente.
"$gridColor$"	Color de cada una de las líneas de la gradilla.
"$gridWidth$"	Anchura de cada una de las líneas de la gradilla.
"$gridOpacity$"	Opacidad de cada una de las líneas de la gradilla (0 totalmente transparente, 1 totalmente opaca).

Por otro lado, para una leyenda de tipo discreta es necesario utilizar los parámetros listados en la **Tabla 3.4.42**.

Tabla 3.4.42. *Parámetros para una leyenda de tipo discreta.*

Nombre	Descripción
"$glyph$"	Tipo de glifo a mostrar en cada elemento de la leyenda
"$glyphFill$"	Color de cada uno de los glifos mostrados en la leyenda.
"$gliphSize$"	Tamaño de cada uno de los glifos mostrados en la leyenda.
"$gliphOpacity$"	Opacidad de cada uno de los glifos mostrados en la leyenda (0 totalmente transparente, 1 totalmente opaco).

Fuerzas

Los ***force-directed layout*** son una clase de técnicas de visualización, utilizadas muy comúnmente, que se basan en el cálculo de las posiciones de un conjunto de nodos simulando una serie de fuerzas físicas que actúan sobre cada uno de los nodos. Dicho con otras palabras, se trata de una forma de representar gráficamente un conjunto de elementos (**nodos**) y las relaciones entre ellos (**ejes**) utilizando algoritmos que simulan su comportamiento en el mundo real bajo la acción de una serie de fuerzas físicas. La **Tabla 3.4.43** muestra los tipos de fuerzas que **jviz** permite aplicar.

Tabla 3.4.43. *Listado de fuerzas disponibles en* ***jviz****.*

Fuerza	Descripción
Repulsión	Permite aplicar una fuerza que hace que dos nodos se repelen, utilizando para ello la ley de Coulomb.
Atracción	Permite aplicar una fuerza que hace que dos nodos se atraigan, utilizando para ello la ley de Hooke.
Centro de masas	Permite aplicar una corrección a la posición de los nodos de forma que el centro de masas de todo el sistema esté lo más cerca posible de un punto (x, y) proporcionado.

El **simulador** es la maquinaria principal de este tipo de modelos. El simulador es el que decide cuando el movimiento de los nodos se inicia y cuando finaliza. El simulador también lleva el control de la posición y la dirección de cada nodo en cada momento, así como del cálculo de todas las fuerzas que actúan sobre cada uno de ellos. La simulación se ejecuta por **iteraciones**, de forma que en cada iteración se recalculan las fuerzas que actúan sobre cada nodo, tras lo cual se utilizan las fuerzas recalculadas para modificar su velocidad y por consiguiente la posición de cada uno de ellos.

El simulador utiliza una serie de variables internas para gestionar el estado del *layout* en cada momento. Dentro de estas variables se incluye la **energía** de la simulación, también llamado ***alpha***. Este *alpha* es un valor real que se encuentra siempre dentro del intervalo (0, 1):

- Cuando el valor de *alpha* se aproxima a 1, la simulación tiene la mayor energía posible, por lo que el valor de las fuerzas aplicadas es el mayor posible y los nodos tienen mayor movimiento.
- Cuando el valor de *alpha* se aproxima a 0, las fuerzas que se aplican sobre cada nodo se ven reducidas. Cuando dicho valor de *alpha* sea menor que un cierto valor umbral (denominado "**alphaMin"**), la simulación se detiene.

Este valor de *alpha* se modifica en cada iteración de la simulación, utilizando una función que hace que el valor de *alpha* converja a un valor ***alphaTarget*** definido por el usuario. Además, en cada iteración se evalúa si el valor de *alpha* es menor que el valor umbral ***alphaMin***,

en cuyo caso la simulación se detiene y el movimiento de los nodos cesa.

Las fuerzas se definen dentro del atributo "$forces$" del esquema, que admite los parámetros mostrados en la **Tabla 3.4.44**.

***Tabla 3.4.44**. Parámetros para configurar el simulador de fuerzas.*

Nombre	Descripción
"$nodes$"	(**Obligatorio**) Nombre del conjunto de datos que contiene los nodos a los cuales se le aplicarán las fuerzas.
"$links$"	Nombre del conjunto de datos que define las relaciones entre los nodos. Cada elemento del conjunto de datos deberá tener un campo "$source$", indicando el ID del nodo de origen, y un campo "$target$" indicando el ID del nodo final de la relación.
"$alphaTarget$"	Valor al que convergerá la variable "$alpha$" del simulador. Este valor debe estar en el intervalo $[0, 1]$, donde $alphaTarget = 0$ indica que la energía del simulador decaerá hasta detenerse totalmente, y $alphaTarget = 1$ indica que el simulador no se detendrá nunca. El valor por defecto es 0.

"$alphaMin$"	Valor umbral de *alpha*. Por defecto su valor es 0.05.
"$restart$"	Bandera que indica si la simulación se debe reiniciar cuando alguno de los parámetros o datos ha cambiado. Por defecto tiene un valor $false$.
"$static$"	Bandera que indica si la simulación se tiene que ejecutar de forma síncrona para generar un *layout* estático ($true$), o por el contrario se genera un *layout* animado ($false$). En el caso de que se opte por un layout estático, la energía del simulador no será tenida en cuenta para decidir cuándo se finalizará la simulación, sino que se ejecutarán tantas iteraciones como se indiquen en el parámetro "$iterations$". Por defecto tiene un valor $false$.
"$iterations$"	Número de iteraciones que el simulador ejecutará en el caso de que la opción de *layout* estático haya sido activada. Por defecto 100.
"$forces$"	(**Obligatorio**) Listado con las fuerzas que se desean aplicar en cada iteración de la simulación a cada nodo.

En cada iteración el simulador calcula las fuerzas que actúan sobre cada uno de los nodos del sistema. Dichas fuerzas se definen a través del parámetro "$forces$", y son las siguientes:

- Fuerza de atracción entre dos nodos que estén relacionados (como si de un muelle que une ambos nodos se tratase) y que se calcula utilizando la ley de **Hooke**.
- Fuerza de repulsión que hace que dos nodos que no estén relacionados se alejen uno del otro, en este caso aplicando la ley de **Coulomb** entre dichos nodos.
- Fuerza de atracción del centro de masas del sistema a un punto concreto.

Tras el cómputo de todas las fuerzas que actúan sobre cada nodo, el simulador modifica la velocidad del nodo, y posteriormente modifica su posición.

En el caso de que se desee fijar la posición de un nodo, y por lo tanto que no se aplique ninguna fuerza sobre él, el elemento del conjunto de datos referente a dicho nodo deberá tener dos campos con el nombre "fx" y "fy", conteniendo un valor numérico con la posición fija del nodo. La posición de dicho nodo en cada iteración será en su lugar reemplazado por los valores almacenados en estos campos.

Tras realizar cada iteración de la simulación (o en el caso de que haya seleccionado la opción de que el *layout* sea estático, tras haber realizado todas las iteraciones), se generará un conjunto de datos especial que contendrá el mismo listado de elementos que el conjunto

de nodos especificado. En cada elemento de este nuevo conjunto de datos se computarán los campos mostrados en la **Tabla 3.4.45**.

Tabla 3.4.45. *Campos generados en cada uno de los elementos del conjunto de datos tras la aplicación de las fuerzas.*

Nombre	Descripción
"x"	Componente x de la posición actual del nodo.
"y"	Componente y de la posición actual del nodo.
"vx"	Componente x de la velocidad actual del nodo.
"vy"	Componente y de la velocidad actual del nodo.

Los campos "x" e "y" de cada nodo pueden ser utilizados por las *geoms* para representar cada uno de los nodos.

En el caso de que se haya proporcionado también un conjunto de datos con los enlaces entre los nodos, también se generará un conjunto de datos especial que contendrá dichos enlaces, pero con los valores de cada uno de los nodos sustituidos. Así, por ejemplo, los campos "$source.x$" y "$source.y$" contendrán las componentes x e y del nodo de origen del enlace, mientras que los campos "$target.x$" y "$target.y$" contendrán las componentes x e y del nodo final del enlace.

Fuerza de repulsión (ley de Coulomb)

Esta fuerza se basa en la ley de Coulomb, en la que la fuerza eléctrica con la que se repelen dos cargas puntuales en reposo es directamente

proporcional al producto de las mismas, inversamente proporcional al cuadrado de la distancia que las separa y actúa en la dirección de la recta que las une. Esto es:

$$F = K\frac{q_1 * q_2}{r^2}$$

Donde:

- F es la fuerza eléctrica de repulsión (o de atracción).
- K es una constante de proporcionalidad llamada constante de la ley de Coulomb. No se trata de una constante universal y depende del medio en el que se encuentren las cargas.
- q_1 y q_2 son los valores de las dos cargas puntuales.
- r es la distancia que separa ambas cargas.

Para el cálculo de esta fuerza sobre cada pareja de nodos, supondremos que $K = 1$ y que el producto $q_1 * q_2$ será un valor positivo proporcionado por el usuario, que denominaremos **factor de repulsión**. A mayor factor de repulsión, los nodos más cercanos se repelen con mayor fuerza.

Los parámetros que permiten aplicar y configurar esta fuerza están mostrados en la **Tabla 3.4.46**.

Tabla 3.4.46. *Parámetros de la fuerza de repulsión.*

Nombre	Descripción

"$repulsion$"	Factor de repulsión entre dos nodos. Por defecto tendrá un valor de 10.
"$maxDistance$"	Distancia máxima entre dos nodos para que esta fuerza actúe. Nodos que estén a una distancia mayor a la proporcionada no se verán afectados por esta fuerza.

Fuerza de atracción (ley de Hooke)

Esta fuerza está basada en la ley propuesta por el físico Robert Hooke, en la que establece que la fuerza requerida para estirar un objeto elástico, como un resorte o muelle de metal, es directamente proporcional a la extensión del resorte. Esto es:

$$F = -kx$$

Donde:

- F es la fuerza. La dirección de esta fuerza generalmente tendrá signo negativo, para indicar que la fuerza de restauración debida al resorte está en dirección opuesta a la fuerza que causó el desplazamiento.
- k es una constante de proporcionalidad, generalmente conocida como constante de resorte.
- x la longitud de la extensión o compresión.

Para aplicar esta ley sobre cada pareja de nodos, utilizaremos x como la distancia entre los dos nodos, y k deberá ser proporcionado por el usuario, que denominaremos como el **factor de atracción**. Así pues, esta fuerza necesitará únicamente que se indique el parámetro mostrado en la **Tabla 3.4.47**.

***Tabla 3.4.47**. Parámetros de la fuerza de atracción.*

Nombre	Descripción
"$attraction$"	Factor de atracción entre dos nodos. Se trata de un valor real dentro del intervalo $[0, 1]$. Por defecto vale 0.5.

Centro de masas

A diferencia de las fuerzas de atracción y de repulsión en las que no modifican directamente las propiedades intrínsecas de los nodos, esta fuerza se basa en la corrección de las posiciones de los nodos de forma que el centro de masas de todo el sistema esté situado lo más cerca posible de un punto (x, y) definido. Los parámetros mostrados en la **Tabla 3.4.48** son necesarios para poder aplicar esta fuerza.

***Tabla 3.4.48.** Parámetros para el centro de masas.*

Nombre	Descripción
"cx"	(**Obligatorio**) Componente x de la posición central a la que se desea atraer todo el sistema.

"cy"	(**Obligatorio**) Componente y de la posición central a la que se desea atraer todo el sistema.

Geoms

Las ***geoms*** son los elementos más básicos de visualización, utilizados para codificar los datos de forma visual. Las *geoms* permiten representar gráficamente figuras geométricas primitivas (como por ejemplo rectángulos, líneas o texto), cuyas propiedades de posición, tamaño y apariencia se pueden construir a partir de los valores de los datos, de forma pura o transformados mediante el uso de escalas. En la **Tabla 3.4.49** se muestra un listado con las *geoms* disponibles en **jviz**.

***Tabla 3.4.49**. Listado de geoms definidas en **jviz**.*

Nombre	Descripción
Arc	Permite representar sectores circulares o sectores de coronas circulares.
Area	Permite representar áreas.
Circle	Permite representar círculos cerrados de un radio dado.
Curve	Permite representar curvas.

Glyph	Permite representar glifos (tanto glifos por defecto como los definidos por el usuario).
Line	Permite representar segmentos que unen dos puntos.
Path	Permite representar caminos (*paths*) siguiendo la sintaxis de SVG.
Polyline	Permite representar un conjunto de segmentos conectados.
Rectangle	Permite representar rectángulos.
Text	Permite representar texto, utilizado normalmente para anotar los datos.

En general, para cada uno de los elementos del conjunto de datos asignado a una geom se genera una figura del tipo de la geom. Como excepción están las *geoms* de tipo curva o área, en las que cada uno de los elementos del conjunto de datos son utilizados para definir un punto de la curva o área.

Cada *geom* requiere de una serie de parámetros, los cuales se muestran en la **Tabla 3.4.50**.

Tabla 3.4.50. *Parámetros globales de una geom.*

Nombre	Descripción

"$type$"	(**Obligatorio**) Tipo de geom a representar.
"$name$"	Permite asignar un nombre a la geom. Imprescindible si se desea asociar eventos a esta geom.
"$source$"	Permite asociar esta geom a un conjunto de datos, de forma que por cada elemento que contenga se representará una geom de este tipo, a excepción de las geoms "$curve$" y "$area$", en las que cada punto de la curva o del área es un elemento del conjunto de datos.
"$panel$"	Permite definir el panel en el que esta geom será representada. Este parámetro admite un número entero positivo con el índice del panel, o una expresión de la forma "$an + b$" (donde a y b son números enteros definidos por el usuario), y por la que la geom será representada en todos los paneles cuyo índice cumpla dicha expresión. En el caso de no especificarse ningún valor, la geom será representada en el primer panel.
"$render$"	(**Obligatorio**) Este parámetro está disponible para cualquier geom a excepción de las que sean de tipo "$group$". Permite establecer las propiedades visuales (tanto de posición como estéticas) de la geom en cada uno de los tres estados de su ciclo de vida.

"$geoms$"	(**Obligatorio**) Este parámetro está solo disponible para las geoms de tipo "$group$", permitiendo especificar un listado de geoms que se representarán dentro del grupo.
"$transform$"	Permite definir un listado de transformaciones que se aplicarán al conjunto de datos definido en el parámetro "$source$", previo a la representación de la geom.
"$tooltip$"	Permite definir el texto que se mostrará cuando el usuario sitúe el ratón sobre la geom.

Datos de entrada

Utilizando el parámetro "$source$" se puede especificar el conjunto de datos que estará asociado a la *geom*. En términos generales, por cada elemento o fila del conjunto de datos se generará una *geom* del tipo especificado, a excepción de las *geoms* de tipo $curve$ y $area$, en las que se utilizará todo el conjunto de datos para representar una única curva o área (de esta forma, cada elemento del conjunto de datos será un punto de la curva o del área). Por otro lado, en el caso de que no se proporcione ningún dato de entrada, solo se generará una única *geom* del tipo especificado.

Este parámetro debe ir acompañado de un objeto en el que se defina la fuente del conjunto de datos. Dicha fuente puede ser especificada utilizando los parámetros mostrados en la **Tabla 3.4.51**.

Tabla 3.4.51. *Parámetros para definir el conjunto de datos asociado a una geom.*

Nombre	Descripción
"$data$"	Permite especificar el nombre del conjunto de datos que se utilizará para representar la *geom*. Este parámetro es incompatible con el parámetro "$force$".
"$groupby$"	Permite especificar el nombre del campo a utilizar para realizar agrupaciones de los datos por el valor de dicho campo. Solo puede utilizarse junto con el parámetro "$data$", y para las geoms "$curve$" y "$area$",
"$force$"	Permite definir si el conjunto de datos que se utilizará para representar la *geom* debe ser extraído del conjunto de nodos o enlaces que resultan de haber aplicado las fuerzas definidas en el apartado "$forces$" del esquema. Este parámetro solo acepta los valores "$nodes$" (para utilizar los datos de los nodos) o "$links$" (para utilizar los datos de los ejes). Este parámetro es incompatible con los parámetros "$data$" y "$groupby$".

Transformaciones

Dentro de la configuración de la *geom* se puede especificar un listado de transformaciones que se aplicarán al conjunto de datos de entrada antes de que se represente la *geom* en cuestión. El conjunto resultante de aplicar estas transformaciones sólo será utilizado para representar la *geom* en la que se hayan definido.

Ciclo de vida de una geom

Dentro del parámetro "$render$" se establecen las propiedades visuales con las que cada *geom* se va a representar, incluyendo tanto las que especifican la posición o tamaño como las de estilo.

Dentro del ciclo de vida hay tres estados y las propiedades se deben asignar a uno de ellos según cuando se deban aplicar (**Figura 3.66**). Estos estados son:

- **Estado inicial** (campo "$\boldsymbol{init}$"): permite definir el conjunto de propiedades visuales iniciales de la *geom*. Normalmente es en este conjunto en el que se definirían las propiedades relacionadas con el tamaño y la posición de la *geom*.
- **Estado actualizado** (campo "$update$" : permite definir el conjunto de propiedades visuales que se aplicarán cada vez que se produzca una actualización en la *geom* (como por ejemplo las modificaciones de variables de estado como interacción del usuario con el gráfico). Generalmente en este conjunto se definen las propiedades cuyo valor depende de variables de estado. Este conjunto de propiedades también se

aplicaría junto con las propiedades definidas en "$init$" en el momento en el que la geom se representa por primera vez.

- **Estado de interacción** (campo "$hover$"): es un conjunto especial en el que se definen las propiedades que se desean aplicar cuando el usuario haga alguna acción sobre la *geom*. Por ejemplo, cuando el cursor entra en la *geom*, se aplican las propiedades de "$hover$" y en cuando el cursor abandone la *geom*, se aplicará el conjunto de propiedades definidos en el campo "$update$", por lo que en dicho conjunto se deberán definir las propiedades que sirvan de reinicio sobre las que se definan en el campo "$hover$".

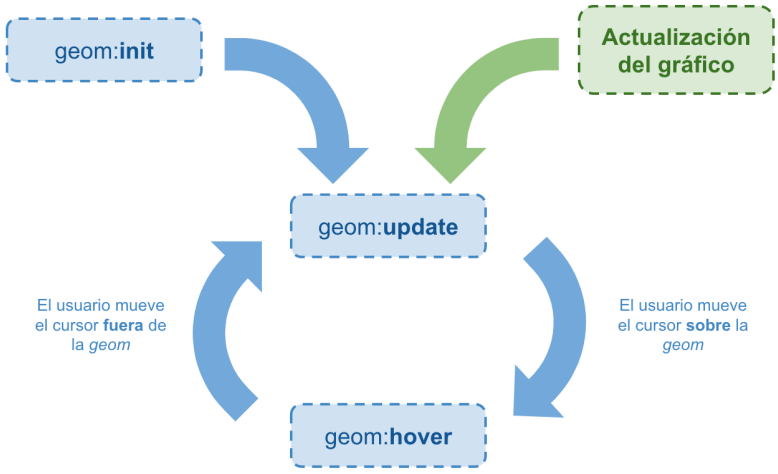


Figura 3.66. *Ciclo de vida de una geom. Cuando la geom se inicializa, se ejecutan las propiedades de los estados «init» y «update». Cuando el usuario mueve el cursor sobre la geom, se ejecutan las propiedades del estado «hover», y cuando mueve el cursor fuera de la geom, se ejecutan de nuevo las propiedades del estado «update». Por último, cuando el gráfico se actualiza (por modificación de las variables de estado) se ejecutan únicamente las propiedades del estado «update».*

Un ejemplo de definición de estas propiedades sería la que se muestra en el **Código 3.8**.

```
{
    "type": "circle",
    "render": {
        "init": {
            "x": {"value": 100},
            "y": {"value": 100},
            "radius": {"value": 50}
        },
        "update": {
            "fill": {"color": "blue"}
        },
        "hover": {
            "fill": {"color": "red"}
        }
    }
    . . .
}
```

Código 3.8. *Ejemplo de definición de los parámetros de una geom de tipo círculo en cada uno de sus tres estados del ciclo de vida.*

En el campo "$init$" se está definiendo la posición del círculo (coordenadas "x" e "y") así como su radio, mientras que en los campos "$update$" y "$hover$" se está definiendo el color de fondo que tendrá dicha figura: inicialmente el círculo tendrá un color de fondo azul. mientras que si el usuario sitúa el ratón sobre el círculo el color de fondo cambiará a rojo; a su vez, cuando el usuario abandone el círculo este cambiará su color de fondo de nuevo a azul.

Tipos de parámetros

La posición, dimensiones o aspecto de una geom se establece a partir de un conjunto de **parámetros**, que se pueden catalogar en dos grupos: parámetros específicos y parámetros de estilo. Cada *geom* cuenta con una serie de **parámetros específicos,** mediante los cuales se puede configurar su tamaño, posición u otros comportamientos. Por otro lado, **los parámetros** de estilo permiten especificar el aspecto visual de la *geom* a representar, los cuales están detallados en la **Tabla 3.4.52**.

Tabla 3.4.52. *Parámetros de estilo disponibles en* ***jviz****.*

Atributo	Descripción
"$fill$"	Permite especificar el color de fondo de la *geom*.
"$stroke$"	Permite especificar el color del borde de la *geom*.
"$strokeWidth$"	Permite especificar el ancho del borde de la *geom*, en píxeles. Por defecto vale 0px.
"$strokeDash$"	Permite especificar el patrón para generar un borde punteado.
"$opacity$"	Permite especificar el nivel de transparencia de la geom, siendo 0 totalmente transparente y 1 totalmente opaca.

"$cursor$"	Permite especificar qué tipo de cursor se mostrará cuando el usuario sitúe el cursor sobre la *geom*. Los valores admitidos son los mismos que los admitidos en el atributo $cursor$ de CSS.
"$transform$"	Permite aplicar transformaciones de CSS sobre la *geom*.

Tipos de geoms

Arc

Permite representar un arco circular a partir del centro del arco, el radio de la corona exterior, el arco de la corona interior y los ángulos de inicio y fin del arco. Mediante este tipo de figuras se puede presentar tanto sectores circulares como coronas circulares. La **Tabla 3.4.53** muestra los parámetros que permiten la representación de arcos en **jviz**.

Tabla 3.4.53. *Parámetro para representar un arco en* ***jviz****.*

Atributo	Descripción
"x"	(**Obligatorio**) Permite especificar la componente x de la coordenada del centro del arco.
"y"	(**Obligatorio**) Permite especificar la componente y de la coordenada del centro del arco.

"$innerRadius$"	Establece el radio interno del sector corona circular.
"$outerRadius$"	Establece el radio externo de la corona o sector circular.
"$radius$"	Permite establecer el radio del sector circular. Utilizar este parámetro es equivalente a utilizar $innerRadius = 0$ y $outerRadius = radius$. Esta propiedad sobreescribe cualquier otro valor especificado en "$innerRadius$" y "$outerRadius$".
"$startAngle$"	Ángulo (en radianes) de inicio del sector o corona circular.
"$endAngle$"	Ángulo (en radianes) de final del sector o corona circular.

Area

Esta *geom* permite representar áreas en el gráfico. Los parámetros que permiten definir un área están detallados en la **Tabla 3.4.54**.

Tabla 3.4.54. *Parámetro para representar una geom de tipo área en* ***jviz****.*

Atributo	Descripción
"$x1$"	(**Obligatorio**) Permite especificar la componente x de cada uno de los puntos de la parte superior del área.

"$y1$"	(**Obligatorio**) Permite especificar la componente y de cada uno de los puntos de la parte superior del área.
"$x2$"	(**Obligatorio**) Permite especificar la componente x de cada uno de los puntos de la parte inferior del área.
"$y2$"	(**Obligatorio**) Permite especificar la componente y de cada uno de los puntos de la parte inferior del área.
"$curve$"	Permite especificar el tipo de interpolación a aplicar.

En un gráfico de jviz, una geom de tipo área está definida siempre por dos curvas: la curva superior, cuyos puntos están definidos por los parámetros $x1$ e $y1$, y la curva inferior, cuyos puntos están definidos por los parámetros $x2$ e $y2$.

Circle

Esta *geom* permite representar un círculo en la posición y radio indicados. Los parámetros mostrados en la **Tabla 3.4.55** son necesarios para poder definir una *geom* de tipo círculo.

Tabla 3.4.55. *Parámetros para representar un círculo en* ***jviz****.*

Atributo	Descripción
"x"	(**Obligatorio**) Permite especificar la componente x del centro del círculo.

"y"	(**Obligatorio**) Permite especificar la componente "y" del centro del círculo.
"$radius$"	(**Obligatorio**) Permite especificar el radio del círculo.

Curve

Esta geom permite representar una curva que une un conjunto de puntos (x, y) ordenados. Por defecto, cada uno de estos puntos se une utilizando una línea recta, aunque se puede especificar un tipo de interpolación para crear curvas suavizadas o definidas por pasos. En la **Tabla 3.4.56** se muestran los parámetros admitidos en esta *geom*.

Tabla 3.4.56. *Parámetro para representar una geom de tipo curve en* ***jviz****.*

Atributo	Descripción
"x"	(**Obligatorio**) Permite especificar la componente x de cada uno de los puntos de la línea.
"y"	(**Obligatorio**) Permite especificar la componente y de cada uno de los puntos de la línea.
"$curve$"	Permite especificar el tipo de interpolación a aplicar.

Path

Esta *geom* permite crear figuras complejas combinando múltiples líneas y curvas, utilizando una serie de comandos. El conjunto de todos los comandos que permite dibujar una figura se denomina

camino. Esta figura está basada en el elemento ***path*** de SVG, por lo que la mayoría de los comandos de SVG son válidos para esta figura (**Tabla 3.4.57**).

Los comandos consisten en órdenes de desplazamiento, y se especifica utilizando un único carácter. Por ejemplo, el comando "mover a" se especifica utilizando el carácter M, seguido de las coordenadas x e y del punto al que se desea trasladar. Así pues, si nos queremos trasladar al punto $(10, 10)$ del gráfico, tendremos que utilizar el comando $M10\ 10$.

Tabla 3.4.57. *Comandos soportados en* ***jviz*** *para la generación de caminos.*

Comando	Descripción
$Mx\ y$	Permite desplazarse hasta el punto (x, y) especificado. Generalmente todo camino debe empezar con este comando, para especificar el punto de inicio de la figura, a partir de la cual el resto de comandos se irán aplicando.
$L\ x\ y$	Este comando permite dibujar una línea recta desde el último punto especificado en el comando anterior, hasta el punto proporcionado por las coordenadas (x, y).
$H\ x$	Este comando es un alias del comando L, que permite dibujar una línea horizontal en la dirección del eje x, hasta el punto que tiene componente x la proporcionada en este

	comando y componente y la del último punto del comando anterior.
$V\ y$	Permite dibujar una línea vertical en la dirección del eje y, hasta el punto que tiene como componente x la del último punto del comando anterior, y componente y la proporcionada a través de este comando.
$C\ x1\ y1, x2\ y2, x\ y$	Permite dibujar una curva de *Bezier* cúbica hasta el punto (x, y), utilizando dos puntos de control especificados por los puntos $(x1, y1)$ y $(x2, y2)$.
$Q\ x1\ y1, x\ y$	Permite dibujar una curva de *Bezier* cuadrática hasta el punto (x, y), utilizando un punto de control determinado por $(x1, y1)$. Esta es una versión simplificada de las curvas de Bezier cúbicas.
$A\ rx\ ry\ r\ lf\ sf\ x\ y$	Permite dibujar un arco hasta el punto (x, y).
Z	Este comando permite cerrar el camino, dibujando automáticamente una línea recta que une el último punto del comando anterior con el primer punto del camino, y generando así un polígono en lugar de una línea poligonal. Este comando no tiene ningún argumento adicional.

A excepción del comando M, todos los comandos utilizan el último punto del comando anterior para realizar su acción.

Para poder utilizar esta *geom*, el parámetro que se muestra en la **Tabla 3.4.58** debe ser proporcionado en el esquema.

Tabla 3.4.58. *Parámetro para representar un* ***path*** *de SVG en* ***jviz****.*

Atributo	Descripción
$"path"$	(**Obligatorio**) Cadena de caracteres conteniendo el camino SVG a representar.

Rectangle

Este tipo de *geom* permite representar gráficamente un rectángulo, utilizado en una gran variedad de gráficos como boxplots o para resaltar ciertas zonas de un gráfico. Un rectángulo puede ser especificado de tres formas distintas: **(1)** proporcionando la posición de los dos vértices opuestos del rectángulo; **(2)** proporcionando la posición de uno de los vértices y la altura y la anchura del rectángulo; **(3)** proporcionando el punto medio y la altura y la anchura del rectángulo. En la **Tabla 3.4.59** se listan los parámetros que permiten representar un rectángulo en **jviz**.

Tabla 3.4.59. *Parámetros para poder representar texto.*

Atributo	Descripción

"$x1$"	Permite especificar la componente x del punto en el que situará la esquina superior izquierda del rectángulo.
"$y1$"	Permite especificar la componente y del punto en el que situará la esquina superior izquierda del rectángulo.
"$x2$"	Permite especificar la componente x del punto en el que se situará la esquina inferior derecha del rectángulo.
"$y2$"	Permite especificar la componente y del punto en el que situará la esquina inferior derecha del rectángulo.
"$width$"	Permite especificar la anchura del rectángulo.
"$height$"	Permite especificar la altura del rectángulo.
"$xCenter$"	Permite especificar la componente x del punto medio del rectángulo.
"$yCenter$"	Permite especificar la componente y del punto medio del rectángulo.
"$radius$"	Permite especificar el radio (en píxeles) de las esquinas, en el caso de que se desee dibujar un rectángulo redondeado. Por defecto este valor vale 0 (sin esquinas redondeadas).

Line

Esta *geom* permite dibujar un segmento que une dos puntos. Existen tres tipos de segmentos: **(1)** los segmentos que unen dos puntos dados; **(2)** los segmentos horizontales, que únicamente necesita la altura del segmento (y) y las componentes horizontales de inicio y final del segmento (x_1 y x_2); **(3)** los segmentos verticales, que únicamente necesitan la anchura del segmento (x) y las componentes verticales de inicio y final del segmento (y_1 e y_2). Los parámetros para poder representar una línea o segmento se muestran en la **Tabla 3.4.60**.

Tabla 3.4.60. *Parámetros para poder representar una geom de tipo línea.*

Atributo	Descripción
"$x1$"	Permite especificar la coordenada x del punto de inicio del segmento.
"$y1$"	Permite especificar la coordenada y del punto de inicio del segmento.
"$x2$"	Permite especificar la coordenada x del punto final del segmento.
"$y2$"	Permite especificar la coordenada y del punto final del segmento.
"x"	Atajo para poder especificar el mismo valor para $x1$ y $x2$ (por lo que se representaría un segmento de tipo vertical).

"y"	Atajo para poder especificar el mismo valor para $y1$ y $y2$ (por lo que se representaría un segmento de tipo horizontal).

Text

La geom de tipo texto se utiliza normalmente para anotar visualmente los datos. Los parámetros mostrados en la **Tabla 3.4.61** permiten definir las propiedades básicas del texto.

Tabla 3.4.61. *Parámetros para poder representar texto.*

Nombre	Descripción
"$text$"	(**Obligatorio**) Texto a representar.
"x"	(**Obligatorio**) Coordenada x del punto en el que se representará el texto.
"y"	(**Obligatorio**) Coordenada y del punto en el que se representará el texto.
"$rotation$"	Ángulo en radianes para la rotación del texto. Por defecto este atributo tiene valor 0.
"$textAnchor$"	Permite alinear horizontalmente el texto (Figura 3.67). Posibles valores: "$start$": el texto se alinea a la izquierda. "$middle$": el texto queda alineado en el centro. "end": el texto se alinea a la derecha.

"$baseline$"	Permite especificar la posición del texto con referencia a la línea base (línea sobre la que se asientan la mayoría de las letras) (Figura 3.68). Admite uno de los siguientes valores: "$hanging$": el texto se dibuja por debajo de la línea base. "$middle$": el texto se dibuja centrado en la línea base (por defecto). "$baseline$": el texto se dibuja sobre la línea base.
"$fill$"	Permite especificar el color del texto.
"$fontSize$"	Permite especificar el tamaño del texto.
"$fontFamily$"	Permite especificar la fuente del texto.
"$fontStyle$"	Permite especificar el estilo del texto: "$normal$", "$italic$" u "$oblique$".
"$fontWeight$"	Permite especificar la anchura del texto: "$normal$"para texto normal o "$bold$" para texto en negrita.

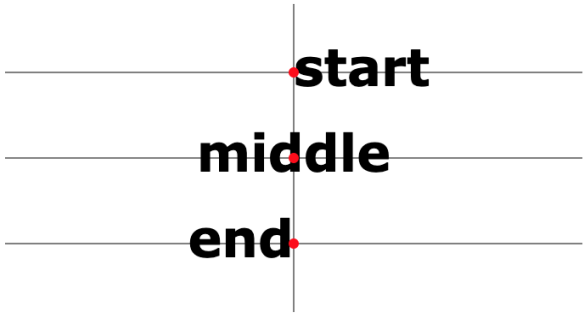


Figura 3.67. *Representación de las tres posibles alineaciones del texto: «start», «middle» y «end», respectivamente (código fuente de la imagen: https://jsfiddle.net/q6jn3hkf).*

Figura 3.68. *Diferentes posiciones del texto en referencia a la línea base. El punto rojo representa las coordenadas (x, y) en las que el texto se dibujaría (código fuente de la imagen: https://jsfiddle.net/9du28gym).*

Eventos

Los **eventos** son la capa más importante en la definición de un gráfico interactivo. Los eventos permiten traducir acciones del usuario sobre los elementos del gráfico (como por ejemplo podría ser pulsar sobre una *geom* o situar el ratón sobre una leyenda) en acciones sobre

variables de estado. Los eventos se definen en **jviz** utilizando un **selector** (**Figura 3.69**), inspirado en los selectores de CSS[40], que en reglas generales consta de tres partes: **(1)** la acción que lanzará el evento; **(2)** el elemento visual sobre el que se aplica (geom, leyenda, panel o todo el gráfico); **(3)** opcionalmente, un filtro basado en el nombre del elemento que permite distinguir sobre qué elementos se aplicará el evento.

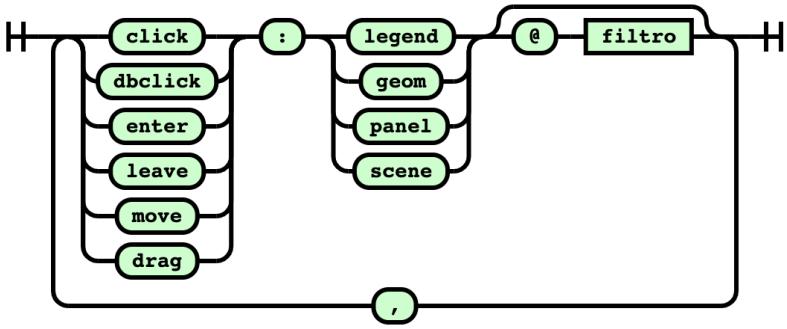


Figura 3.69. *Diagrama de railes del selector de eventos.*

Los eventos se deben definir en el campo "$events$" del esquema. Para cada evento que se desee registrar, se deberán proporcionar los parámetros expuestos en la **Tabla 3.4.62**.

Tabla 3.4.62. *Parámetros para registrar un evento.*

Parámetro	Descripción

[40] Selectores de CSS: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors

"on"	(**Obligatorio**) Selector del evento a aplicar, que debe seguir el patrón mostrado en la **Figura 3.69**.
"$update$"	Permite definir las variables de estado que se modificarán cuando este evento sea accionado.

3.4.3 **Ejemplo de visualización con jviz**

Un ***heatmap*** es una representación gráfica de datos donde los valores individuales contenidos en una matriz se representan como colores (**Figura 3.70**). Los *heatmaps* son útiles para mostrar la varianza entre múltiples variables, revelar patrones, mostrar si las variables son similares entre sí y para detectar si existe alguna correlación entre ellas.

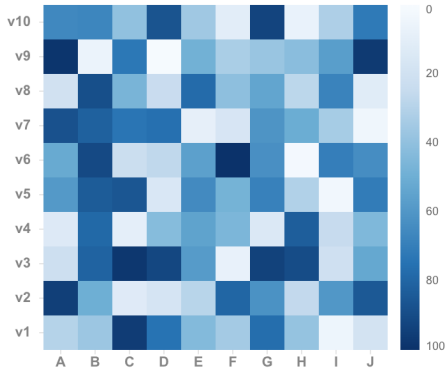


Figura 3.70. *Ejemplo de heatmap.*

En esta sección se va a mostrar como construir un *heatmap* utilizando **jviz**. Para ello, se ha dividido el proceso de construcción de este *heatmap* en tres pasos:

- **Primer pas**o: se definirá el **esquema** mínimo con el que se construirá un *heatmap* básico.
- **Segundo paso**: se añadirán los ejes al *heatmap*.
- **Tercer paso**: se añadirá una leyenda de color para ayudar en la interpretación del *heatmap*.

El esquema final que se obtendrá tras seguir los tres pasos que se acaban de especificar será uno similar al mostrado en el **Código 3.9**.

```
{
    "width": 350,
    "height": 250,
    "margin": {"left": 50, "bottom": 50, "right": 10},
    "outerMargin": {"right": 90},
    "data": [{
        "name": "table",
        "url": "/data/heatmap.json",
        "format": "json"
    }],
    "scales": [{
        "name": "x",
        "type": "interval",
        "domain": {"data": "table", "field": "group"},
        "range": {"draw": "width"}
    }, {
        "name": "y",
        "type": "interval",
        "domain": {"data": "table", "field": "variable"},
        "range": {"draw": "height"}
    }, {
```

```
        "name": "color",
        "type": "color",
        "domain": {"data": "table", "field": "value"},
        "range": {"palette": "blues"},
        "zero": true
    }],
    "axes": [{
        "position": "bottom",
        "scale": "x",
        "ticks": true,
        "tickOffset": 10
    }, {
        "position": "left",
        "scale": "y",
        "ticks": true,
        "tickOffset": 10,
        "tickAnchor": "end"
    }],
    "legends": [{
        "position": "right",
        "orientation": "vertical",
        "scale": "color",
        "type": "gradient",
        "gradientSize": {"value": 15},
        "gradientLength": {"draw": "height"},
        "gradientTicks": 5,
        "labelSize": {"value": "9px"},
        "labelOpacity": {"value": 0.6}
    }],
    "geoms": [{
        "type": "rectangle",
        "source": {"data": "table"},
        "render": {
            "init": {
                "x1": {"field": "group", "scale": "x", "interval": 0},
                "x2": {"field": "group", "scale": "x", "interval": 1},
                "y1": {"field": "variable", "scale": "y", "interval": 0},
                "y2": {"field": "variable", "scale": "y", "interval": 1}
            },
            "update": {
                "fill": {"field": "value", "scale": "color"}
```

```
            }
        }
    }]
}
```

***Código 3.9**. Esquema en **jviz** del heatmap.*

Estructura de los datos

Los datos que se desean representar como *heatmap* están almacenados en una estructura de datos en formato JSON, como en el mostrado en el **Código 3.10**.

```
[
    {"group": "A", "variable": "v1", "value": 30},
    {"group": "A", "variable": "v2", "value": 95},
    {"group": "A", "variable": "v3", "value": 22},
    . . .
]
```

***Código 3.10.** Primeras líneas de la estructura en JSON que contiene los datos a representar.*

En total esta estructura contiene 100 entradas (filas), cada una de ellas conteniendo los siguientes campos:

- Un campo llamado "$group$", que almacena un carácter dentro del conjunto $\{"A","B",\ldots,"J"\}$. Este campo será utilizado en el eje de las **x**.

- Un campo llamado "$variable$", que almacena una cadena de caracteres que tiene la estructura "$v\{i\}$", con $1 \leq i \leq 10$. Este campo será utilizado en el eje de las **y**.
- Por último, un campo llamado "$value$" con el valor numérico a representar.

Primer paso: *heatmap* básico

En este primer paso se va a realizar una representación básica del *heatmap*, a partir de los datos previamente introducidos. El resultado se puede observar en la **Figura 3.71**, y el esquema que permite generarlo en el **Código 3.11**.

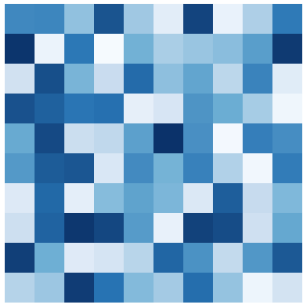


***Figura 3.71**. Representación básica del heatmap resultado de este primer paso.*

```
{
    "width": 200,
    "height": 200,
    "data": [{
        "name": "table",
        "url": "/data/heatmap.json",
        "format": "json"
```

```
    }],
    "scales": [{
        "name": "x",
        "type": "interval",
        "domain": {"data": "table", "field": "group"},
        "range": {"draw": "width"}
    }, {
        "name": "y",
        "type": "interval",
        "domain": {"data": "table", "field": "variable"},
        "range": {"draw": "height"}
    }, {
        "name": "color",
        "type": "color",
        "domain": {"data": "table", "field": "value"},
        "range": {"palette": "blues"},
        "zero": true
    }],
    "geoms": [{
        "type": "rectangle",
        "source": {"data": "table"},
        "render": {
            "init": {
                "x1": {"field": "group", "scale": "x", "interval": 0},
                "x2": {"field": "group", "scale": "x", "interval": 1},
                "y1": {"field": "variable", "scale": "y", "interval": 0},
                "y2": {"field": "variable", "scale": "y", "interval": 1}
            },
            "update": {
                "fill": {"field": "value", "scale": "color"}
            }
        }
    }]
}
```

***Código 3.11**. Esquema resultante del primer paso.*

Configuración del gráfico

En el primer nivel del esquema se define la configuración básica del gráfico. Para este primer paso, únicamente se necesitaría definir la anchura y la altura del gráfico (**Código 3.12**).

```
"width": 200,
"height": 200
```

***Código 3.12**. Dimensiones del gráfico.*

Los parámetros "$width$" y "$height$" permiten definir respectivamente la anchura y la altura total del gráfico. Al no haber definido ningún tipo de margen esta primera iteración, el área de dibujo que las *geoms* tendrán disponible será de $200px$ de altura por $200px$ de anchura.

Datos de entrada

La siguiente capa que se añadirá al esquema serán los datos de entrada del gráfico, dentro del parámetro "$data$". Este parámetro acepta un listado de objetos, en los que cada uno será un conjunto de datos que deberá tener un nombre único asociado, así como el origen o fuente de donde se extraerán dichos datos. En este ejemplo, se define un único conjunto de datos con el nombre "$table$" (**Código 3.13**).

```
"data": [{
    "name": "table",
    "url": "data/heatmap.json",
    "format": "json"
```

```
}],
```

Código 3.13. *Especificación de los datos de entrada.*

En **jviz**, los datos pueden ser cargados desde tres fuentes:

- Desde un archivo externo, especificando la dirección web donde se encuentra dicho archivo utilizando el parámetro "url". Esto permite cargar archivos de tipo JSON y CSV.
- A partir de otro conjunto de datos previamente definido, especificando su nombre en el parámetro "$source$".
- Desde unos valores especificados en el propio esquema, utilizando el parámetro "$value$". Esto también incluye la opción de que el conjunto de datos esté vacío (especificando "$value$": []), y sea completado de forma dinámica utilizando transformaciones.

Estos tres parámetros "url", "$value$" y "$source$" son mutuamente excluyentes entre sí, por lo que solo puede ser utilizado uno de ellos en cada conjunto de datos. En este caso en concreto, como los datos a utilizar están en un archivo JSON, se ha utilizado el parámetro "url" acompañado de la dirección en la que se encuentra.

Escalas

Las escalas permiten mapear valores de los datos a propiedades visuales. Las escalas se deben definir dentro del parámetro "$scales$" del esquema (**Código 3.14**).

```
"scales": [{
    "name": "x",
    "type": "interval",
    "domain": {"data": "table", "field": "group"},
    "range": {"draw": "width"}
}, {
    "name": "y",
    "type": "interval",
    "domain": {"data": "table", "field": "variable"},
    "range": {"draw": "height"}
}, {
    "name": "color",
    "type": "color",
    "domain": {"data": "table", "field": "value"},
    "range": {"palette": "blues"},
    "zero": true
}]
```

Código 3.14. *Definición de las escalas para el heatmap.*

Cada escala debe tener un nombre único asociado (proporcionado a través del parámetro "$name$") para que pueda ser referenciada y utilizada en otras capas, además de un tipo de escala (definido en el parámetro "$type$"). El tipo de la escala es el que determina cómo deben ser el dominio (valores de los datos a representar) y el rango (los valores visuales, como color o píxeles, a los que se transformará el dominio).

Para este ejemplo se han definido tres escalas. Las dos primeras escalas son de tipo intervalo (que representan el eje x y el eje y), por lo que el dominio de ellas debe ser un listado de valores únicos, mientras que el rango es un intervalo, el cual se dividirá en tantos

subintervalos (de igual tamaño) como elementos tenga el dominio. La tercera escala es de tipo color, por lo que el dominio será un intervalo de forma que cualquier valor dentro de dicho intervalo será mapeado a un color que esté dentro del rango de colores proporcionado, interpolando dicho color si fuera necesario.

En la primera escala (en el eje horizontal), al definir el dominio como $\{"data": "table", "field": "group"\}$, esto obtendrá todos los posibles valores que pueda tener el campo "$group$" en el conjunto de datos "$table$", para posteriormente dejar solo los valores únicos, que en este ejemplo concreto sería el listado $["A", "B", "C", \ldots, "J"]$. Por otro lado, el rango de la escala se ha definido como $\{"draw": "width"\}$, lo cual es traducido por jviz como el intervalo $[0, 200]$. Así pues, cada uno de los elementos del dominio será mapeado a uno de los diez subintervalos en los que haya sido dividido el rango, de forma que:

- El elemento "A" es mapeado al subintervalo $[0, 20]$.
- El elemento "B" es mapeado al subintervalo $[20, 40]$.
- Y así sucesivamente, hasta llegar al elemento "J" que es mapeado al último subintervalo $[180, 200]$.

La segunda escala (en el eje vertical) se generaría de forma similar a la primera, en la que al definir el dominio como $\{"data": "table", "field": "variable"\}$, este se traduciría como el listado $["v1", "v2", \ldots, "v10"]$. Por otro lado, como el rango se ha definido como $\{"draw": "height"\}$, este se traduciría al intervalo $[200, 0]$ (notar que en este caso el intervalo está en orden inverso). Así, cada uno de

los elementos del dominio será mapeado a un subintervalo del rango, de la siguiente forma:

- El elemento "$v1$" es mapeado al subintervalo $[200, 180]$.
- El elemento "$v2$" es mapeado al subintervalo $[180, 160]$.
- Y así sucesivamente, hasta llegar al elemento "$v10$" que es mapeado al subintervalo $[20, 0]$.

Para la tercera escala de tipo color, al definir el dominio como $\{"data": "table", "field": "value"\}$ este se construye tomando los valores mínimo y máximo del campo "$value$" en el conjunto de datos "$table$", mientras que el rango es un gradiente de azules, de forma que el azul más claro procederá del valor mínimo y el azul más oscuro del valor máximo.

Notar que en las dos primeras escalas, al especificar en el rango $\{"draw": "width"\}$ y $\{"draw": "height"\}$, esto se está traduciendo en los intervalos $[0, 200]$ y $[200, 0]$, respectivamente. El intervalo que se genera para el caso de la altura está invertido. Esto se debe a que en SVG el origen de coordenadas se encuentra situado en la esquina superior izquierda del gráfico, mientras que si se piensa en cualquier gráfica convencional, el origen de coordenadas se sitúa en la esquina inferior izquierda. De esta forma, al invertir el rango, el primer valor del dominio sería mapeado al valor de la altura, por lo que estaría situado en la parte inferior del gráfico, mientras que el último elemento sería mapeado al valor 0, por lo que estaría situado en la parte superior del gráfico.

Geoms

Las *geoms* son los elementos primarios de visualización en **jviz**, los cuales deben ser construidos a partir de los conjuntos de datos.

Para cada *geom* se debe especificar su tipo concreto (rectángulo, círculo, línea, etc..), mediante el parámetro "$type$". A continuación, se debe especificar utilizando el parámetro "$source$" el conjunto de datos al que estará asociado esta *geom*, de forma que para cada elemento del conjunto de datos se representará en el SVG una *geom* del tipo especificado. Las propiedades visuales de la *geom* se deben especificar en el parámetro "$render$", en el que además habrá que detallar en qué momento del ciclo de vida ("$init$", "$update$" y "$hover$") de la *geom* se deben aplicar dichas propiedades visuales (**Código 3.15**).

```
"geoms": [{
    "type": "rectangle",
    "source": {"data": "table"},
    "render": {
        "init": {
            "x1": {"field": "group", "scale": "x", "interval": 0},
            "x2": {"field": "group", "scale": "x", "interval": 1},
            "y1": {"field": "variable", "scale": "y", "interval": 0},
            "y2": {"field": "variable", "scale": "y", "interval": 1},
            "fill": {"field": "value", "scale": "color"}
        }
    }
}]
```

Código 3.15. *Definición de las escalas para el heatmap.*

En este ejemplo se está definiendo una única *geom* de tipo rectángulo, la cual utilizará el conjunto de datos "$table$". De esta forma, para cada uno de los elementos de dicho conjunto de datos se representará en sel SVG un rectángulo cuyas propiedades visuales están definidas en el parámetro "$render$".

Dentro del parámetro "$render$" se ha de dividir las propiedades visuales en tres grupos: "$init$", "$update$" y "$hover$".

- Dentro del grupo "$init$"se especifican las propiedades que se deben aplicar en el momento que cada *geom* se crea.
- Dentro del grupo "$update$" se especifican las propiedades que se deben aplicar cuando la *geom* se crea y cuando se actualiza.
- Por último, dentro del grupo "$hover$" se especifican las propiedades que se deben aplicar cuando el usuario sitúa el ratón sobre esta *geom*.

En este ejemplo en concreto solo se está utilizando el grupo "$init$", en el que se están definiendo las propiedades mostradas en el **Código 3.16**.

```
"x1": {"field": "group", "scale": "x", "interval": 0},
"x2": {"field": "group", "scale": "x", "interval": 1},
"y1": {"field": "variable", "scale": "y", "interval": 0},
"y2": {"field": "variable", "scale": "y", "interval": 1},
"fill": {"field": "value", "scale": "color"}
```

Código 3.16. *Propiedades visuales del heatmap.*

Las propiedades "$x1$", "$y1$", "$x2$" e "$y2$" permiten definir los dos vértices opuestos del rectángulo, imprescindibles para que pueda ser renderizado. Las propiedades "$x1$" y "$x2$" se extraen del campo "$group$" del elemento al que está asociado este rectángulo, tras lo cual se le aplica la escala "x". Esto haría que ambas propiedades tuvieran al final como valor el inicio del subintervalo al que están asociados por medio de la escala "x". Sin embargo, al utilizar el parámetro "$interval$" con valores "0" para "$x1$" y "1" para "$x2$", esto hace que "$x1$" mantenga el valor de inicio del subintervalo, pero "$x2$" pase a tener el valor final del subintervalo. Esto mismo ocurriría con las propiedades "$y1$" e "$y2$".

Por otro lado, la propiedad "$fill$" permite definir el color de fondo del rectángulo. Dicho color será extraído a partir del valor del campo "$value$" del elemento al que está asociado el rectángulo, tras lo cual se le aplica la escala "$color$" que lo transforma a un color dentro del esquema de azules.

Segundo paso: añadiendo ejes

Una vez se tiene el *heatmap* básico, el siguiente paso sería añadir los ejes izquierdo e inferior para que el usuario sepa a qué pares de variables pertenece cada uno de los valores mostrados en el *heatmap*. El gráfico que se obtendrá será similar al expuesto en la **Figura 3.72**, y el esquema que se obtendrá será el mostrado en el **Código 3.17**.

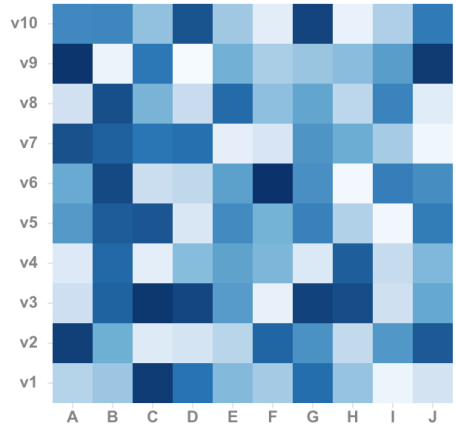


***Figura 3.72**. Resultado del heatmap tras el segundo paso.*

```
{
    "width": 250,
    "height": 250,
    "margin": {"left": 50, "bottom": 50},
    "data": [{
        "name": "table",
        "url": "/data/heatmap.json",
        "format": "json"
    }],
    "scales": [{
        "name": "x",
        "type": "interval",
        "domain": {"data": "table", "field": "group"},
        "range": {"draw": "width"}
    }, {
        "name": "y",
        "type": "interval",
        "domain": {"data": "table", "field": "variable"},
        "range": {"draw": "height"}
    }, {
        "name": "color",
        "type": "color",
        "domain": {"data": "table", "field": "value"},
        "range": {"palette": "blues"},
```

```
        "zero": true
    }],
    "axes": [{
        "position": "bottom",
        "scale": "x",
        "ticks": true,
        "tickOffset": 10
    }, {
        "position": "left",
        "scale": "y",
        "ticks": true,
        "tickOffset": 10,
        "tickAnchor": "end"
    }],
    "geoms": [{
        "type": "rectangle",
        "source": {"data": "table"},
        "render": {
            "init": {
                "x1": {"field": "group", "scale": "x", "interval": 0},
                "x2": {"field": "group", "scale": "x", "interval": 1},
                "y1": {"field": "variable", "scale": "y", "interval": 0},
                "y2": {"field": "variable", "scale": "y", "interval": 1}
            },
            "update": {
                "fill": {"field": "value", "scale": "color"}
            }
        }
    }]
}
```

***Código 3.17**. Esquema del heatmap para el segundo paso.*

Modificación de la configuración del gráfico

Para poder añadir los ejes, primero se debe modificar la configuración básica del gráfico para incrementar el tamaño del mismo y añadir los márgenes en los que se representarán los ejes (**Código 3.18**).

```
- "width": 200,
- "height": 200
+ "width": 250,
+ "height": 250,
+ "margin": {"left": 50, "bottom": 50}
```

***Código 3.18**. Diferencia entre la configuración del esquema del paso anterior (rojo) y de este paso (verde).*

En rojo se muestra la configuración que había del paso anterior, y en verde la parte que se modifica y que se añade. En primer lugar, se han añadido $50px$ a la anchura y altura del gráfico, que serán utilizados para los márgenes internos izquierdo e inferior. La configuración del margen interno se debe especificar a partir del parámetro "$margin$", el cual admite uno de los siguientes valores:

- Un valor numérico, de forma que los cuatro márgenes tendrán dicho valor.
- Un objeto en el que proporcione los valores específicos de los márgenes que se desean añadir. Los márgenes que no hayan sido especificados tendrán por defecto $0px$ de tamaño.

En este caso se ha utilizado un objeto en el que se especifican los valores de los márgenes izquierdo ("$left$") e inferior ("$bottom$"), ambos con un valor de $50px$. Como se acaba de comentar, como los otros dos márgenes no han sido especificados, ambos tendrán un valor de $0px$.

Ejes

Los ejes permiten representar gráficamente los valores de las escalas, para ayudar al usuario a interpretar mejor el gráfico (**Código 3.19**).

```
"axes": [{
    "position": "bottom",
    "scale": "x",
    "ticks": true,
    "tickOffset": 10
}, {
    "position": "left",
    "scale": "y",
    "ticks": true,
    "tickOffset": 10,
    "tickAnchor": "end"
}]
```

Código 3.19. *Esquema de los ejes del heatmap.*

El parámetro "$position$" permite especificar la posición de cada uno de los ejes, junto con la escala a la que está asociada especificada en el parámetro "$scale$". Mediante el parámetro "$ticks$": $true$ se obliga al eje a que muestre los valores del dominio de la escala, llamados $ticks$: al tratarse en ambos ejes de escalas discretas, todos los valores del dominio serán mostrados en el eje. Por último, el parámetro "$tickOffset$": 10 permite establecer el número de píxeles de separación entre el eje y cada uno de los $ticks$, mientras que en la escala de la izquierda el parámetro "$tickAnchor$": "end" permite alinear los "$ticks$" a la derecha.

Tercer paso: añadiendo la leyenda de degradado

En un *heatmap*, la leyenda es un caso especial en el que los valores representados son los valores continuos de una medida. Esto obliga a que deben visualizarse no como entradas discretas de agrupaciones, sino como una escala degradada de color que permita tener una idea aproximada del valor del campo según su aproximación visual en la leyenda.

Por ello, en este tercer y último paso del ejemplo se realizarán las modificaciones oportunas al esquema resultante del paso anterior para que se muestre la leyenda con el degradado (escala continua) en el lado derecho del gráfico. En la **Figura 3.73** se muestra el *heatmap* final de este tercer paso, y en el **Código 3.20** el esquema final de este tercer paso.

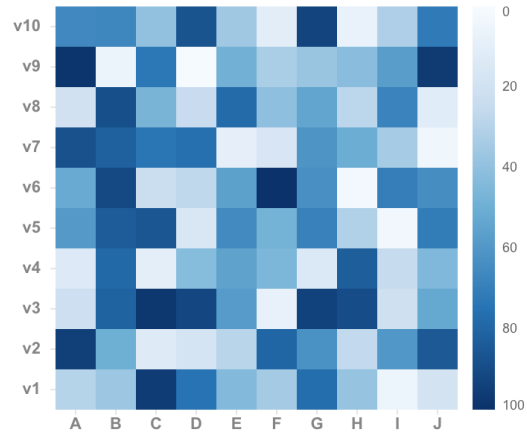


Figura 3.73. *Heatmap resultante del tercer y último paso.*

```
{
```

```
"width": 350,
"height": 250,
"margin": {"left": 50, "bottom": 50, "right": 10},
"outerMargin": {"right": 90},
"data": [{
    "name": "table",
    "url": "/data/heatmap.json",
    "format": "json"
}],
"scales": [{
    "name": "x",
    "type": "interval",
    "domain": {"data": "table", "field": "group"},
    "range": {"draw": "width"}
}, {
    "name": "y",
    "type": "interval",
    "domain": {"data": "table", "field": "variable"},
    "range": {"draw": "height"}
}, {
    "name": "color",
    "type": "color",
    "domain": {"data": "table", "field": "value"},
    "range": {"palette": "blues"},
    "zero": true
}],
"axes": [{
    "position": "bottom",
    "scale": "x",
    "ticks": true,
    "tickOffset": 10
}, {
    "position": "left",
    "scale": "y",
    "ticks": true,
    "tickOffset": 10,
    "tickAnchor": "end"
}],
"legends": [{
    "position": "right",
```

```
        "orientation": "vertical",
        "scale": "color",
        "type": "gradient",
        "gradientSize": {"value": 15},
        "gradientLength": {"draw": "height"},
        "gradientTicks": 5,
        "labelSize": {"value": "9px"},
        "labelOpacity": {"value": 0.6}
    }],
    "geoms": [{
        "type": "rectangle",
        "source": {"data": "table"},
        "render": {
            "init": {
                "x1": {"field": "group", "scale": "x", "interval": 0},
                "x2": {"field": "group", "scale": "x", "interval": 1},
                "y1": {"field": "variable", "scale": "y", "interval": 0},
                "y2": {"field": "variable", "scale": "y", "interval": 1}
            },
            "update": {
                "fill": {"field": "value", "scale": "color"}
            }
        }
    }]
}
```

Código 3.20. *Esquema resultante del tercer paso.*

Modificación de la configuración del gráfico

Es necesario realizar una nueva modificación en la configuración del gráfico para añadir los márgenes externos, sobre los que se representará la leyenda. Como la leyenda se desea mostrar en el lado derecho, únicamente sería necesario añadir el margen externo de dicho lado. Teniendo en cuenta que la leyenda ocupará todo el ancho del margen externo derecho, es recomendable añadir también un

margen interno derecho para que haya una separación entre el gráfico y la leyenda (**Código 3.21**).

```
- "width": 250,
+ "width": 350,
  "height": 250,
- "margin": {"left": 50, "bottom": 50}
+ "margin": {"left": 50, "bottom": 50, "right": 10},
+ "outerMargin": {"right": 90}
```

Código 3.21. *Diferencias en la configuración del gráfico entre el paso anterior (en rojo) y este paso (en verde).*

En rojo se muestra la configuración que había del paso anterior, y en verde la parte que se modifica y que se añade. En primer lugar, se han añadido $100px$ a la anchura del gráfico, que será utilizado para el margen externo derecho. Además, se ha añadido un margen interno derecho de $10px$, con la finalidad de añadir una separación entre el heatmap y la leyenda.

Por último, se ha añadido la configuración de los márgenes externos, especificado en el parámetro "$outerMargin$". Dicho parámetro admite los mismos valores que el parámetro "$margin$". Como la leyenda se va a situar en el lado derecho del gráfico, únicamente se ha añadido el margen derecho, de forma que el resto de márgenes tendrán $0px$ de tamaño.

Leyendas

Las leyendas permiten visualizar las transformaciones visuales realizadas por escalas, como por ejemplo colores, glifos y tamaños. En este caso en concreto se va a utilizar una leyenda de degradado, las cuales representan un dominio continuo utilizando un degradado de color continuo (**Código 3.22**).

```
"legends": [{
    "position": "right",
    "orientation": "vertical",
    "scale": "color",
    "type": "gradient",
    "gradientSize": {"value": 15},
    "gradientLength": {"draw": "height"},
    "gradientTicks": 5,
    "labelSize": {"value": "9px"},
    "labelOpacity": {"value": 0.6}
}]
```

***Código 3.22**. Configuración de la leyenda.*

Las leyendas tienen una serie de parámetros obligatorios: "$type$", "$position$" y "$scale$". El parámetro "$position$" permite establecer la posición de la leyenda con respecto al gráfico, lo cual en este ejemplo en concreto se ha posicionado en el lado derecho del gráfico. Por otro lado, los parámetros "$scale$" y "$type$" permiten establecer la escala a la que estará asociada la leyenda y el tipo de leyenda, respectivamente. Es importante tener en cuenta que no cualquier tipo de escala es válida para ser representada con una leyenda, ni

cualquier combinación de tipo y escala son válidas. Así pues, únicamente serían válidas las siguientes combinaciones:

- "$type$": "$gradient$" y una escala que sea de tipo color.
- "$type$": "$glyph$" y una escala que sea de tipo categórica.

En ese caso se está utilizando la escala que tiene como nombre "$color$", que es una escala de tipo color, por lo que la combinación con la leyenda de tipo degradado es válida. Los colores del degradado serán extraídos del rango de la escala, mientras que las etiquetas (los valores numéricos a los que están asociados ciertos colores del degradado) se extraerán del dominio de la escala.

La configuración del degradado se establece utilizando los siguientes parámetros:

- "$gradientSize$": {"$value$": 15} , para indicar la anchura que tendrá el degradado, el cual en este caso será de $15px$.
- "$gradientLength$": {"$draw$": "$height$"}, para indicar la altura del degradado, que en este caso será la misma que la altura del gráfico.
- "$gradientTicks$": 5, para indicar el número estimado de valores que deberán ser representadas junto con el degradado. Dichos valores estarán siempre dentro del dominio de la escala.

Por último, la configuración de los valores que se mostrarán junto con el degradado se define utilizando los siguientes parámetros:

- "$labelSize$": {"$value$": "$9px$"}, que permite definir el tamaño del texto de las etiquetas, siendo en este caso de $9px$ cada una.
- "$labelOpacity$": {"$value$": 0.6} , para indicar el nivel de transparencia del texto de las etiquetas.

3.4.4 **Reactividad**

Para que los gráficos construidos con **jviz** sean útiles en entornos de análisis exploratorio de datos interactivos, necesitamos dos condiciones: **(1)** el gráfico debe actualizarse cuando alguno de los elementos de entrada (variables de estado o datos) cambie de valor; **(2)** tras un evento de actualización, para poder ser eficiente y mantener la interactividad, únicamente se deben modificar aquellos elementos que sean estrictamente necesarios, para que la representación sea lo más eficiente posible. La primera condición es fácil de satisfacer (por ejemplo, borrando todo el gráfico y volviendo a regenerarlo). Sin embargo, conseguir que las dos condiciones se cumplan a la vez requiere un proceso de investigación e ingeniería complejo que no es apreciado a simple vista.

Para conseguir que estas dos condiciones se cumplan a la vez, se necesita la construcción de una estructura de datos especial que permita el control de esa reactividad (la actualización únicamente de las dependencias y no de todo el gráfico). Para ello se desarrolló un sistema que transforma el esquema JSON de **jviz** en un **grafo anotado** donde cada nodo almacena el estado y las propiedades del elemento en tiempo real, mientras que las aristas sirven para

establecer las relaciones de dependencia imprescindibles para poder llevar a cabo el proceso de actualización optimizada del gráfico. De forma que, como se muestra a continuación, con esta estructura se soluciona el problema de que el gráfico pueda ser actualizado modificando el mínimo número de elementos del mismo.

Construcción del grafo reactivo

Dado un problema de visualización a resolver, el primer paso sería definir los componentes gráficos a utilizar y sus eventos para posteriormente crear el esquema de **jviz** en el que se definen todos los elementos necesarios. Distinguimos aquí entre tres tipos de elementos:

1. **Elementos de entrada**, como por ejemplo las variables de estado o los conjuntos de datos.
2. **Elementos de transformación**, que serían las transformaciones o las escalas.
3. **Elementos de salida**, que serían las *geoms*, ejes y leyendas.

En lo sucesivo, a estos tres tipos de elementos se les asignará un glifo concreto, como se puede observar en la **Figura 3.74**.

Figura 3.74. Glifo asignado a cada uno de los tipos de elementos del grafo.

Durante la fase de evaluación del esquema, jviz construye un grafo en el que cada uno de los elementos que se han definido en el esquema constituye un nodo del grafo. Supongamos que, tras el procesado del esquema obtenemos un grafo como el mostrado en la **Figura 3.75**, en el que tenemos tres elementos de entrada (dos variables de estado y un conjunto de datos), cuatro elementos de transformación (dos transformaciones y dos escalas) y cuatro elementos de salida (dos ejes y dos *geoms*).

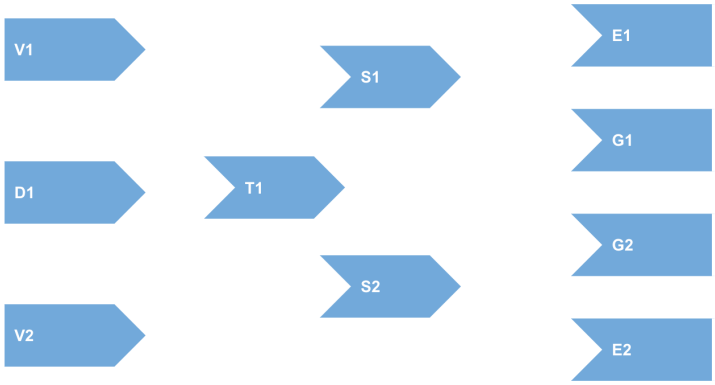


Figura 3.75. Nodos del grafo reactivo.

Estableciendo relaciones

Las relaciones entre los nodos de este grafo vienen dado por las relaciones de dependencia entre elementos: decimos que un elemento A depende de otro elemento B si en la especificación de alguno de los parámetros del elemento A utilizamos el elemento B. Por ejemplo, si en un atributo de una *geom* necesitamos transformar un valor a otro utilizando una escala definida, entonces esa geom depende de dicha escala, por lo que en el grafo habrá una relación entre la *geom* y la escala en cuestión.

Supongamos que, tras procesar el esquema de ejemplo, las relaciones en el grafo vienen dadas como las mostradas en la **Figura 3.76**.

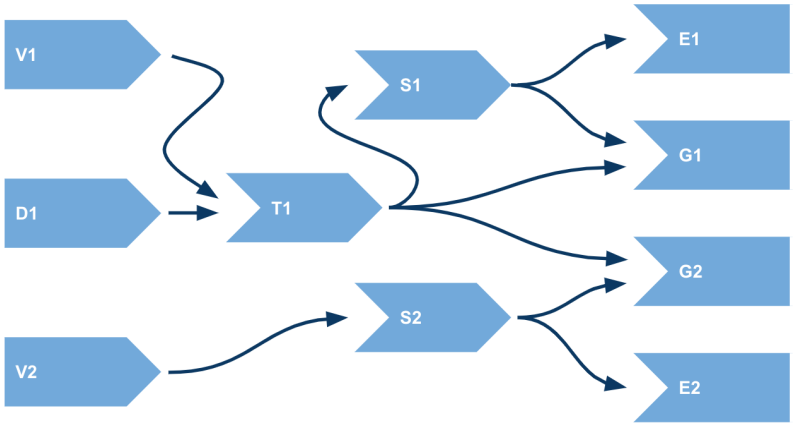


Figura 3.76. *Grafo con las relaciones entre cada uno de los nodos, simulando las relaciones de dependencia entre los elementos del gráfico. En este caso en concreto se tiene que: la transformación T1 depende del conjunto de datos D1 y de la variable de estado V1; la escala S1 depende únicamente del resultado de la transformación T1; la escala S2 depende del valor de la*

variable V2; la geom G1 depende del resultado de la transformación T1 y de la escala S1; la geom G2 depende del resultado de la transformación T1 y de la escala S2; el eje E1 depende de la escala S1; el eje E2 depende de la escala S2.

Aunque verbalmente se expresa que es el elemento A (por ejemplo, S2 en **Figura 3.76**) es el que depende del elemento B(V2), en el dibujo del flujo del esquema, la dirección de la relación entre los nodos del grafo (la punta de la flecha en la **Figura 3.76**) es al revés, ya que en el grafo la relación es que el valor del elemento B es el que está siendo utilizado para el cómputo del elemento A.

Para poder extraer todas las relaciones de dependencia entre los diferentes elementos del esquema, **jviz** incorpora un analizador que le permite detectar estas dependencias tanto en los atributos o parámetros de los elementos como en las expresiones. Este proceso de análisis únicamente se realiza la primera vez que el esquema es procesado.

Respondiendo a las modificaciones

En **jviz**, las modificaciones de las variables de estado o de los conjuntos de datos pueden producirse por dos circunstancias concretas:

1. En respuesta a la interacción del usuario con el gráfico, por medio de un evento declarado en el esquema del gráfico, las cuales ocasionan modificaciones en las variables de estado.
2. Por modificaciones externas a través de la API de **jviz**.

Cualquiera de estas dos circunstancias hace que se produzca un cambio en los valores almacenados en las variables de estado o en los conjuntos de estado, haciendo que el gráfico tenga que ser actualizado como respuesta a dichos cambios. Gracias al grafo de dependencias, se consigue optimizar el número de modificaciones que hay que realizar en los elementos del gráfico, evitando así el realizar cálculos innecesarios y reduciendo el tiempo que se necesita para actualizarlo.

El funcionamiento es el siguiente: supongamos que, por respuesta a una interacción del usuario con el gráfico, se lanza un evento interno en el gráfico que termina con la modificación de la variable de estado **V1** (**Figura 3.77**).

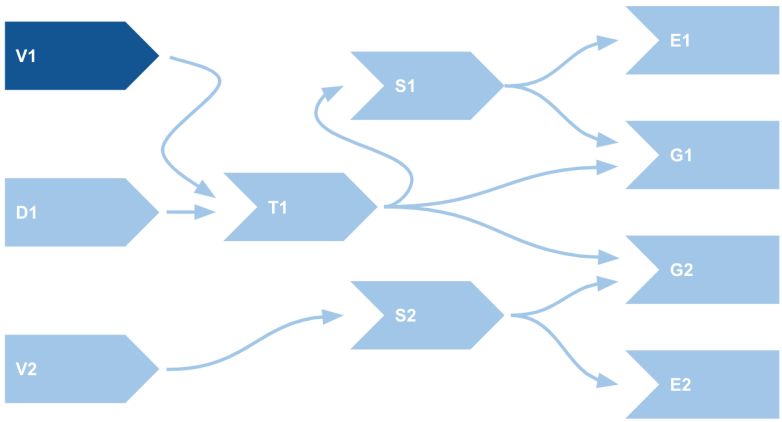


Figura 3.77. *Estado del grafo tras actualizar la variable de estado* ***V1****.*

Para poder conocer todos los elementos que están implicados en la modificación del valor de la variable de estado **V1**, **jviz** recorre este grafo, en este caso a partir del nodo de **V1**, modificando secuencialmente los elementos de transformación hasta que por último se modifican los elementos de salida implicados. El proceso sería el siguiente:

1. Como el nodo **V1** está relacionado con el nodo T1, **jviz** vuelve a aplicar la transformación **T1** (**Figura 3.78**) sobre el conjunto de datos **D1**.
2. Una vez esta transformación ha finalizado, **jviz** continúa recorriendo el grafo buscando los siguientes elementos de transformación que dependan de **T1**, que este caso sería la escala **S1** (**Figura 3.79**).
3. Puesto que ya no hay más elementos de transformación que dependan de los dos que han sido modificados (**T1** y **V1**), **jviz** procede a modificar los elementos de salida que estén relacionados con los elementos que han sido modificados, que según el grafo construido serían las geoms **G1** y **G2** y el eje **E1** (**Figura 3.80**).

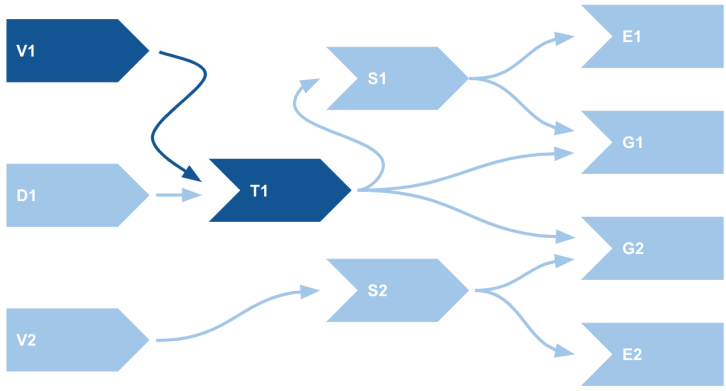


***Figura 3.78**. Modificación de la transformación **T1** por estar relacionada con la variable de estado **V1**.*

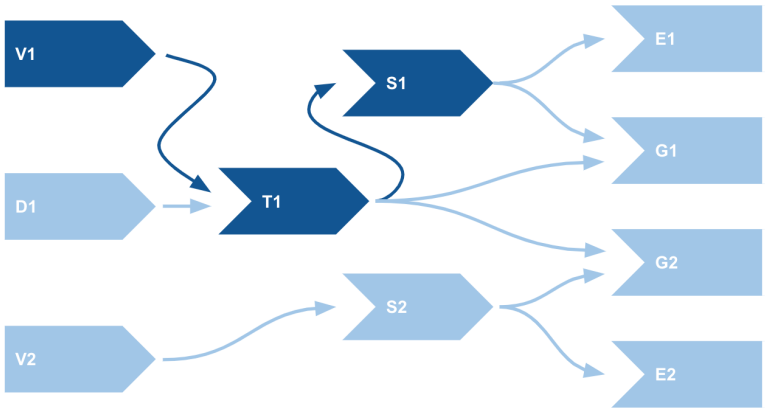


***Figura 3.79**. Actualización de la escala **S1** por depender del resultado de la transformación **T1**.*

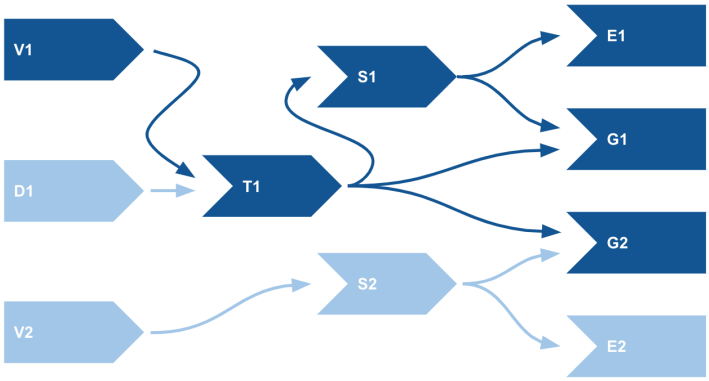


Figura 3.80. *Resultado final de la actualización de los elementos estrictamente necesarios tras la modificación del valor de la variable* ***V1****.*

3.4.5 **Formalizaciones matemáticas**

Escalas

En **jviz**, se ha definido una **escala** como una función s que transforma un valor abstracto procedente de los datos a una propiedad visual del gráfico (por ejemplo, a un tamaño o posición en píxeles o un color).

Dentro de una escala, el conjunto de partida se define como el **dominio** D de la escala, mientras que el conjunto de propiedades visuales a los que la escala mapea se define como el **rango** R de la escala. De esta forma, se puede formalizar una escala s de la siguiente forma:

$$s: D \rightarrow R$$

$$x \rightarrow s(x)$$

Las escalas se pueden dividir en dos grupos, dependiendo de la transformación que se vaya a aplicar y del tipo de datos del dominio: escalas **continuas** y escalas **discretas**.

Escalas continuas

Una escala continua realiza una transformación de un valor continuo de un dominio numérico en otro valor continuo del rango (numérico o color). De esta definición se puede extraer que tanto el dominio como el rango de una escala continua no es más que un intervalo, de forma que la escala continua transformará el inicio del dominio en el inicio del rango, y el final del dominio se transforma en el final del rango. En lo sucesivo, se considera $D = [x_0, x_1]$ como el dominio de la escala y $R = [y_0, y_1]$ como el rango de la escala, donde $x_0, x_1 \in \mathbb{R}, x_0 \leq x_1$.

Dentro de las escalas continuas, se ha definido cuatro tipos de escalas: **lineal, logarítmica, exponencial** y **color**.

Escala lineal

Una escala lineal s_{lineal} transforma un valor del dominio en otro del rango de forma proporcional, y viene dada por la siguiente expresión:

$$s_{lineal}(x) = m * x + n \quad m, n \in \mathbb{R}$$

Donde m y n son dos constantes dependientes del dominio y del rango en el que se aplique la escala.

Además, una escala lineal transformaría el valor inicial del dominio en el valor inicial del rango, y el valor final del dominio en el valor final del rango. Con la notación de dominio y rango que se ha definido anteriormente, la escala debería cumplir lo siguiente:

$$s_{lineal}(x_0) = y_0$$

$$s_{lineal}(x_1) = y_1$$

De esta forma, se puede despejar las constantes m y n para obtener una expresión en función de los valores iniciales y finales del dominio y del rango:

$$s_{lineal}(x) = y_0 + (y_1 - y_0) * (x - x_0) / (x_1 - x_0)$$

Escala logarítmica

Una escala logarítmica s_{log} es otro tipo de escala continua que se utiliza generalmente para transformar valores que tienen magnitudes diferentes, y que consiste en realizar una transformación logarítmica al valor del dominio antes de que sea transformado a un valor del rango. Así, una escala logarítmica para una base k dada se puede expresar de la siguiente forma:

$$s_{log}(x) = m * log_k(x) + n \quad m, n \in \mathbb{R}$$

De igual forma que con la escala lineal, se puede sustituir los valores de m y n para obtener la expresión de esta escala con la notación de dominio y rango que se ha definido anteriormente:

$$s_{log}(x) = y_0 + (y_1 - y_0) * (log_k(x) - log_k(x_0)) \,/\, (log_k(x_1) - log_k(x_0))$$

Escala exponencial

Una escala exponencial s_{exp} es un tipo de escala continua que se basa en realizar una transformación exponencial al valor del dominio antes de que sea transformado a un valor del rango de la escala. Dado un exponente k, una escala exponencial se puede expresar como sigue:

$$s_{exp}(x) = m * x^k + n \quad m, n \in \mathbb{R}, k \in \mathbb{Z}$$

Esta expresión se puede reescribir en función de los valores del dominio y rango de la forma siguiente:

$$s_{exp}(x) = y_0 + (y_1 - y_0) * (x^k - (x_0)^k) \,/\, ((x_1)^k - (x_0)^k)$$

Escala color

Una escala de color, s_{color}, es un tipo especial de escala continua que aplica una transformación lineal a un dominio continuo pero sobre un rango de colores. Esto significa que el rango de la escala no es un rango numérico, sino un rango de color.

Generalmente, el rango que se proporciona a esta escala son dos colores en formato hexadecimal. Para poder realizar la interpolación y devolver un nuevo color que esté dentro del rango de los dos colores proporcionados, esta escala realiza una transformación previa de los

colores a valores numéricos decimales en formato RGB (*Red*, *Green* y *Blue*).

Escalas discretas

Una escala discreta s_{dis} mapea cada uno de los valores discretos del dominio en un único valor del rango. En lo sucesivo, se considera como $D = \{x_0, \ldots, x_N\}$ y $R = \{y_0, \ldots, y_N\}$ el dominio y el rango de la escala respectivamente.

Escala intervalo

Se define como escala intervalo a una escala que, a partir de un rango continuo, realiza una división de dicho rango en intervalos de igual tamaño y asigna a cada valor del dominio a un único intervalo del rango. El número de intervalos vendrá dado por el cardinal del dominio (esto es, el número de valores que contiene el dominio).

Para realizar la división del rango en intervalos, se considera $N = |D|, N > 0$,$R = [y_0, y_1]$ el rango de la escala, y $L = |y_1 - y_0|$ la longitud del rango. Definimos β como la longitud de cada intervalo. Se considera lo siguiente:

- Entre cada intervalo se puede dejar una separación, cuya longitud será proporcional a la longitud que tenga cada intervalo. Así pues, dado $p \in [0, 1]$, se tiene que la separación entre cada intervalo tendrá tamaño $\beta * p$. Si el rango se divide en N intervalos, entonces tendremos $N - 1$ separaciones, por lo que la suma de todas las separaciones entre intervalos será $(N - 1) * (\beta * p)$.

- Antes del primer intervalo y después del último intervalo también puede haber una separación, que al igual que la separación entre intervalos será una cantidad proporcional a la longitud de cada intervalo. Dado entonces $m \in [0,1]$, se tiene que la separación exterior será de $2 * (\beta * m)$.

Con todas estas consideraciones, se tiene que la longitud β de cada intervalo deberá cumplir la siguiente expresión:

$$2 * (\beta * m) \ + \ (N-1) * (\beta * p) \ + \ \beta * N \ = \ L$$

De donde despejando β se puede obtener que la longitud de cada intervalo es la siguiente:

$$\beta \ = \ L \ / \ (2 * m \ + \ (N-1) * p \ + \ N)$$

La escala intervalo s_{int} se puede expresar de la siguiente forma:

$$s_{int}(x_i) \ = \ \beta * m \ + \beta * i * (1 \ + \ p), \qquad 0 \leq i \leq N, \qquad x_i \in D$$

Escala punto

Una escala punto es una variante de la escala intervalo en la que la longitud de cada intervalo es nula.

La construcción de la escala punto es similar a la de la escala intervalo: sea $N = |D|, N > 0$,$R = [y_0, y_1]$ el rango de la escala, $L = |y_1 - y_0|$ la longitud del rango y $m \in [0,1]$ el margen del punto inicial y final.

Definimos η como la separación entre cada punto que debe cumplir la siguiente expresión:

$$2 * (m * \eta) + (N - 1) * \eta = L$$

Por lo tanto, la separación entre puntos es la siguiente:

$$\eta = L / (2 * m + N - 1)$$

De esta forma, la escala punto s_{punto} se puede expresar como:

$$s_{punto}(x_i) = \eta * (m + i), \qquad 0 \leq i \leq N, \qquad x_i \in D$$

Interpolaciones

En el campo del análisis matemático, se conoce como **interpolación** al proceso de búsqueda de una función continua que permita obtener nuevos puntos dentro de un rango de puntos conocidos, denominados **puntos de control**. Existen diferentes formas de interpolación, dependiendo del tipo de función que se defina.

Las interpolaciones utilizando polinomios son las más fáciles y rápidas de calcular. Sin embargo, cuando el número de puntos que se dispone para realizar la interpolación es grande, la interpolación polinomial no es la más adecuada. Una alternativa para poder solucionar este inconveniente es utilizar funciones polinómicas definidas a trozos, conocidas como ***splines***.

Splines

Una ***spline*** (Ahlberg, 1967) es una forma matemática de representar una curva, a partir de una serie de puntos a lo largo de la curva y definiendo una función polinómica a trozos que permita calcular puntos adicionales dentro de cada intervalo definido por dichos puntos, satisfaciendo las condiciones de continuidad entre dichos puntos. Se define el **grado de una *spline*** como el grado de la función polinómica.

Consideremos una función $s\colon [a,b] \to \mathbb{R}$ definida dentro del intervalo $[a,b]$, y una partición de dicho intervalo en $N+1$puntos (que serán los puntos de control), es decir, $x_0 < x_1 < \ldots < x_{N-1} < x_N$, donde $x_0 = a$ y $x_N = b$. Definimos N intervalos utilizando dichos puntos de la forma que sigue:

$$A_i = [x_i, x_{i+1}], 0 \leq i < N$$

Construimos la función polinómica de grado k de la *spline* en el intervalo A_i de la forma siguiente:

$$s_i(x) = \sum_{j=0}^{k} a_{i,j}x^j = a_{i,0} + a_{i,1}x + a_{i,2}x^2 + \ldots + a_{i,k}x^k, x \in [x_i, x_{i+1}]$$

Donde $a_{i,j} \in \mathbb{R}$ son los coeficientes del polinomio. Así, podemos definir la función $s(x)$ de la siguiente forma:

$$s(x) = \qquad\qquad s_0(x) \quad si \quad x \in [x_0, x_1]$$

$$s_1(x) \quad si \quad x \in [x_1, x_2]$$

…

$$s_{N-1}(x) \quad si \quad x \in [x_{N-1}, x_N]$$

Esta función $s: [a, b] \rightarrow \mathbb{R}$ se considera ***spline* de grado k** y cumple las siguientes propiedades:

1. La restricción de dicha función en cada uno de sus intervalos A_i es un polinomio de grado k.
2. La función es de clase C^{k-1} en el intervalo $[a, b]$ (es decir, es k-1 veces continuamente diferenciable en dicho intervalo).

Esta última condición implica que cada una de las funciones polinómicas de la ***spline*** deben cumplir lo siguiente:

$$s_i\,(x_{i+1}) \;=\; s_{i+1}\,(x_{i+1})$$

$$s'_i\,(x_{i+1}) \;=\; s'_{i+1}\,(x_{i+1})$$

…

$$s_i^{(k-1)}\,(x_{i+1}) \;=\; s_{i+1}^{(k-1)}\,(x_{i+1})$$

Para cualquier $0 \leq i < N - 1$.

Por ejemplo, en una *spline* de tercer grado el polinomio en la que se expresaría sería de tercer grado. Para poder expresar esta *spline*, es

necesario encontrar $4N$ coeficientes, que denotaremos por $(a_i, b_i, c_i, d_i),\ 0 \leq i < N$. Por la propiedad (1), podemos obtener $N + 1$ condiciones, mientras que la propiedad (2) nos proporciona $3(N - 1)$ condiciones para obtener estos coeficientes. Las otras dos condiciones restantes deberán ser impuestas al sistema para poder asegurar que cada *spline* sea única.

Las *splines* se utilizan comúnmente para realizar interpolaciones ya que simplifican los cálculos y ofrecen una gran precisión y estabilidad de los resultados.

Interpolación lineal

La interpolación **lineal** utiliza *splines* de primer grado, que se basan en segmentos que unen puntos de control consecutivos dos a dos.

Consideremos el siguiente conjunto $\{(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)\}$ como el conjunto de puntos de control sobre los que queremos realizar la interpolación. Al tratarse de una *spline* de primer grado, la ecuación de cada uno de los segmentos en cada intervalo $[x_i, x_{i+1}]$ tendrá la siguiente estructura:

$$l_i(x) = a_i + b_i x, \quad x \in [x_i, x_{i+1}] \quad 0 \leq i < N$$

Como este tipo de interpolación debe pasar por cada punto de control, se tiene que:

$$y_i = l_i(x_i) = a_i + b_i x_i$$

$$y_{i+1} = l_i(x_{i+1}) = a_i + b_i x_{i+1}$$

De aquí podemos extraer que:

$$a_i = \frac{y_i x_{i+1} - y_{i+1} x_i}{x_{i+1} - x_i}, \qquad b_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Interpolación Catmull-Rom

El gran inconveniente que se presenta al realizar una interpolación lineal a trozos es que la función que se obtiene no es derivable en los puntos de control x_i. Para obtener curvas suaves (es decir, que sean derivables en los puntos de control), debemos recurrir a *splines* de un mayor grado. Las *splines* que más se utilizan son las *splines* de tercer grado, denominadas también como ***splines* cúbicas**.

Las ***splines de Catmull-Rom*** (Catmull y Rom, 1974) pertenecen a la familia de interpolaciones utilizando *splines* cúbicas, y vienen definidas de forma que la tangente de cada punto de control se calcula utilizando el punto siguiente y el anterior. Además, la curva pasa por cada uno de los puntos de control, lo cual no ocurre en todos los tipos de *splines*.

Para poder calcular cualquier punto en la curva, necesitamos cuatro puntos de control: dos a ambos lados del punto que queremos calcular. Supongamos que estos cuatro puntos de control son x_{i-2}, x_{i-1}, x_i y x_{i+1}, por lo que el punto de la curva que queremos calcular

está dentro del intervalo $[x_{i-1}, x_i]$. Sea $\tau \in [0,1]$ la **tensión** de la curva, y definimos la tangente en los puntos de control x_{i-1} y x_i como $\tau(x_i - x_{i-2})$ y $\tau(x_{i+1} - x_{i-1})$ respectivamente (**Figura 3.81**).

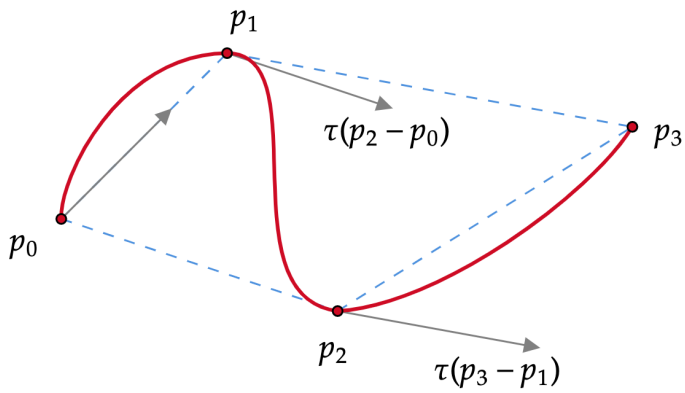


Figura 3.81. *Representación gráfica de las tangentes en los puntos p_1 y p_2.*

Se define una parametrización de la curva que pasa por x_{i-1} y x_i de la siguiente forma:

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Como la curva pasa por los puntos de control, se tiene que:

$$p(0) = a_0 = x_{i-1}$$

$$p(1) = a_0 + a_1 + a_2 + a_3 = x_i$$

Por otro lado, derivando esta parametrización y sustituyendo por la tangente en los dos puntos de control, se obtienen las dos expresiones siguientes:

$$p'(0) \;=\; a_1 = \tau(x_i - x_{i-2})$$

$$p'(1) = a_1 + 2a_2 + 3a_3 = \tau(x_{i+1} - x_{i-1})$$

Se obtiene así el siguiente sistema de cuatro ecuaciones con cuatro incógnitas:

$$a_0 \;= x_{i-1}$$

$$a_0 + a_1 + a_2 + a_3 \;=\; x_i$$

$$a_1 = \tau(x_i - x_{i-2})$$

$$a_1 + 2a_2 + 3a_3 = \tau(x_{i+1} - x_{i-1})$$

Resolviendo este sistema se obtienen los siguientes valores para las constantes a_i:

$$a_0 = x_{i-1}$$

$$a_1 = \tau(x_i - x_{i-2})$$

$$a_2 = 3(x_i - x_{i-1}) - \tau(x_{i+1} - x_{i-1}) - 2\tau(x_i - x_{i-2})$$

$$a_3 = -2(x_i - x_{i-1}) + \tau(x_{i+1} - x_{i-1} + x_i - x_{i-2})$$

Estos valores pueden ser expresados de una forma más ordenada:

$$a_0 = (1)x_{i-1}$$

$$a_1 = (-\tau)x_{i-2} \;+\; (\tau)x_i$$

$$a_2 = (-2\tau)x_{i-2} + (3 - \tau)x_{i-1} + (3 - 2\tau)x_i + (-\tau)x_{i+1}$$

$$a_3 = (-\tau)x_{i-2} + (2 - \tau)x_{i-1} + (\tau - 2)x_i + (\tau)x_{i+1}$$

De aquí, se puede expresar la parametrización de la curva en forma matricial, como sigue:

$$\mathbf{p}(s) = \mathbf{u}^T \mathbf{M} \mathbf{p} = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \end{bmatrix}$$

La **tensión** afecta a la curva en los puntos de control, definiendo la amplitud tangente a cada punto de control, pero no la dirección de la curva, por lo que se puede elegir cualquier valor razonable entre 0 y 1 para obtener la curvatura que mejor se ajuste a los puntos. De normal se suele escoger como valor $\tau = \frac{1}{2}$.

Las *splines* de *Catmull-Rom* presentan las siguientes propiedades matemáticas:

- La curva definida pasa por todos los puntos de control.
- Esta *spline* es de tipo C^1, ya que no hay discontinuidades en la dirección de la tangente ni en su magnitud.
- Sin embargo, no es de tipo C^2, ya que la segunda derivada se interpola linealmente dentro de cada intervalo, lo que hace que la curva varíe linealmente a lo largo del intervalo.
- Si se modifica la posición de uno de los puntos de control, solo afecta a la curva localmente.

Por otro lado, ¿qué ocurre con la curva entre los dos primeros puntos, x_0 y x_1, o con los dos últimos puntos, x_{N-1} y x_N? La tangente en los

puntos de control x_0 y x_{N-1} no está definida, por lo que muchas implementaciones optan por no definir la curva en los puntos extremos. Otras implementaciones sí que las definen utilizando la tangente entre x_0 y x_1 y la tangente entre x_{N-1} y x_N, que sería $\tau(x_1 - x_0)$ y $\tau(x_N - x_{N-1})$, respectivamente.

Escalonada

Una **función escalonada *h*** definida en un intervalo $[a, b]$ es una función definida a trozos dentro de dicho intervalo que tiene un número finito de discontinuidades $a = x_0 < x_1 < \ldots < x_N = b$ y que cumple que en cada subintervalo (x_i, x_{i+1}) la función es constante, teniendo una discontinuidad en cada punto x_i.

Con esta definición, se puede expresar una función escalonada como una función h: $[a, b] \rightarrow \mathbb{R}$, donde:

$$h(x) = \sum_{i=0}^{N-1} \alpha_i \chi_{A_i}(x)$$

Donde $A_i = [x_i, x_{i+1})$, para $0 \leq i < N - 1$, $A_{N-1} = [x_{N-1}, x_N]$, $\alpha_i \in \Re$ es el valor de la función escalonada en el intervalo A_i, y χ_{A_i} es la función característica del subconjunto A_i y que está definida de la siguiente forma:

$$\chi_{A_i}(x) = 1 \text{ si } x \in A_i$$

$$\chi_{A_i}(x) = 0 \text{ si } x \notin A_i$$

Los intervalos A_i que se acaban de definir cumplen también las siguientes propiedades:

- La intersección entre cada uno de ellos es vacía, esto es, $A_i \cap A_j = \{\}, \forall i \neq j$.
- La unión de todos los intervalos forma el intervalo original en el que está definida la función escalonada, esto es $\cup_{i=0}^{N-1} A_i = [a, b]$.

Para poder construir una función escalonada, se define $L = \{(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)\}$ como el conjunto de N puntos de la línea a representar utilizando la interpolación escalonada. Existen tres formas de definir los intervalos en los que se definirá la función escalada (**Figura 3.82**).

Primera opción (*step*): cada punto de la línea es el centro del intervalo. Así, la partición quedaría formada por lo siguientes puntos:

$$P = \{x_0, \frac{x_1 + x_0}{2}, \frac{x_2 + x_1}{2}, \ldots, \frac{x_N + x_{N-1}}{2}, x_N\}$$

En total la partición tendrá $N + 2$ puntos, por lo que los intervalos de la función escalonada serían:

$$A_0 = [x_0, (x_1 + x_0)/2)$$

$$A_i = [(x_i + x_{i-1})/2, (x_{i+1} + x_i)/2), 1 \leq i < N$$

$$A_N = [(x_N + x_{N-1})/2, x_N]$$

En total habrían $N + 1$ intervalos.

Segunda opción (*step-after*): cada punto de la línea es el inicio del intervalo. En este caso, los intervalos que se considerarían son:

$$A_i = [x_i, x_{i+1}), 0 \leq i < N$$

Además, se tendría un intervalo extra, $A_N = \{x_N\}$, ya que si no se perdería el valor de y_N, por lo que en total habrían $N + 1$ intervalos.

Tercera opción (*step-before*): cada punto de la línea es el final del intervalo. Esto obliga que los intervalos a considerar para la función escalonada fueran:

$$A_i = (x_{i-1}, x_i], 0 < i \leq N$$

Como ocurre en el caso anterior, para no perder el valor del primer punto de la línea, se tendría que añadir un intervalo extra, $A_0 = \{x_0\}$, por lo que en total habrían $N + 1$ intervalos para esta opción.

En los tres casos, la función escalonada vendría definida por:

$$h(x) = \sum_{i=0}^{N} y_i \chi_{A_i}(x)$$

Donde A_i serían los intervalos para una de las tres opciones que se acaban de especificar.

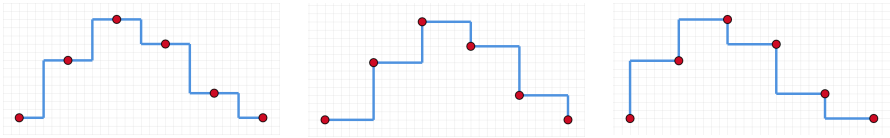


***Figura 3.82**. Ejemplo de los tres tipos de interpolaciones escalonadas, de izquierda a derecha: «step», «step-after» y «step-before».*

3.4.6 **Editor online**

Una parte importante de todo lenguaje de gráficos o módulos de visualización es la creación de herramientas para su prueba en tiempo real, donde se puedan mostrar sus capacidades reactivas de visualización interactiva. Dentro de este trabajo se ha investigado la creación de este tipo de herramientas y qué especificaciones deberían tener para cumplir con una experiencia de usuario completa, y funcional.

Como prueba de concepto se ha creado un editor online con los mínimos elementos necesarios para poder probar y editar los esquemas de forma interactiva, emulando su encapsulación en componentes y dándoles un entorno interactivo con una arquitectura similar a las aplicaciones reales para la interacción de eventos entre la visualización y los manejadores de actualización de estado.

Al ser **jviz** una biblioteca de visualización interactiva es importante que se puedan mostrar sus funcionalidades desde un entorno donde el

usuario solo se tenga que preocupar de la construcción del esquema. Por otro lado, también es importante el poder tener una herramienta que permita el testado y evaluación de los nuevos desarrollos de la biblioteca de software. Es por esto por lo que una parte del trabajo de la tesis se basa en la formalización, descripción y modelado de este tipo de herramientas y su importancia.

El editor, llamado **jvizlab**, es accesible a través de la siguiente url: https://viz.mgviz.org.

Estructura de la aplicación

La unidad de trabajo en el editor es el ***sandbox***. Un *sandbox* es un contenedor que permite almacenar datos (en archivos JSON) y esquema. Los *sandboxes* se guardan en el navegador del usuario, utilizando una base de datos llamada ***IndexedDB*** [41], que está disponible en todos los navegadores web modernos.

La aplicación se divide en dos vistas funcionales: la vista principal, en la que se muestran los *sandboxes* creados por el usuario, y la vista del editor, en la que el usuario puede visualizar e interaccionar con el gráfico generado a partir de un esquema.

[41] Documentación de ***IndexedDB***: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

Vista principal

La vista principal del editor (**Figura 3.83**) muestra un listado de todos los *sandboxes* creados por el usuario, así como un listado de ejemplos que se pueden utilizar para probar **jviz**. Para cada uno de los *sandboxes* listados se muestra: **(1)** una miniatura del gráfico generado; **(2)** título del gráfico; **(3)** una breve descripción del gráfico; **(4)** un botón para cargar ese *sandbox*.

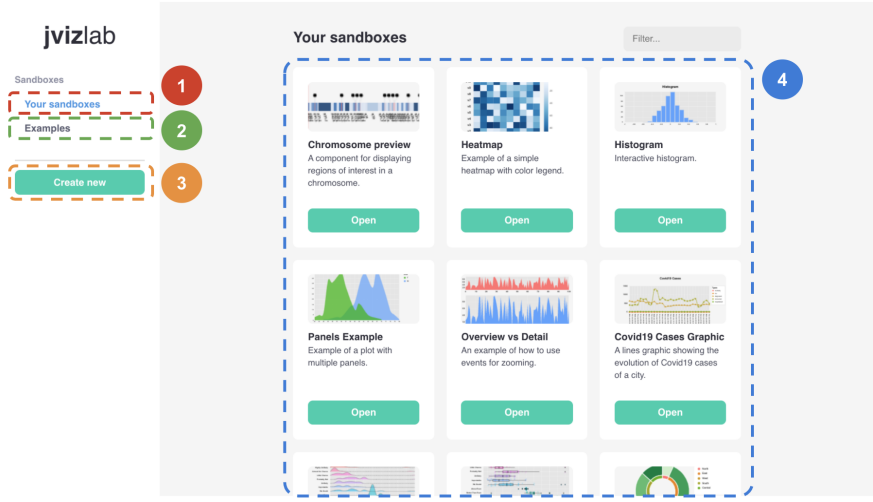


Figura 3.83. *Vista principal del editor. Se pueden distinguir cuatro partes:* ***(1)*** *opción para mostrar los «sandboxes» del usuario;* ***(2)*** *opción para mostrar los «sandboxes» de ejemplo;* ***(3)*** *botón para crear un nuevo «sandbox» vacío;* ***(4)*** *listado de los «sandboxes» para cada una de las dos opciones anteriormente indicadas.*

Vista editor

La **vista del editor** (**Figura 3.84**) se accede cuando el usuario selecciona un *sandbox* desde la vista principal, bien de los que tenga creados localmente o de los ejemplos. Esta vista está formada por cinco elementos o paneles, los cuales están conectados internamente para simular la experiencia de aplicación funcional al usuario y poder experimentar en tiempo real los resultados del esquema.

***Figura 3.84**. Vista del editor, formado por cinco paneles que se comunican entre ellos: **(1)** cabecera; **(2)** explorador de archivos del «sandbox»; **(3)** panel para la edición del esquema o de los datos; **(4)** panel para la visualización de los gráficos; **(5)** panel de interacción con el gráfico.*

En la parte superior del editor se encuentra la **cabecera**, la cual está dividida en tres secciones: **(1) el logo del editor** en la parte izquierda, de forma que al pulsar sobre él se cerraría el editor y se volvería a mostrar la vista principal; **(2) el título del *sandbox***, en la parte central

de la cabecera; y **(3)** una serie de botones para realizar acciones concretas sobre el *sandbox* que se está editando o visualizando (guardar, descargar, configurar o borrar el *sandbox*).

En la parte izquierda del editor se encuentra el **explorador de archivos** contenidos en el *sandbox*. En la parte superior de este panel se encuentra un botón que permite crear un nuevo archivo y añadirlo al *sandbox*.

En la parte central del editor se muestra el **panel de edición**, que permite explorar y modificar tanto los esquemas de **jviz** como los archivos con los datos. Para facilitar la escritura, este panel incorpora un editor interactivo que permite tanto el coloreado del contenido como el autocompletado. Para realizar la ejecución de los esquemas del *sandbox*, en este panel se muestra un botón en la esquina superior derecha del mismo, que se conecta con el panel derecho para realizar la visualización del esquema que se esté editando.

En la parte derecha se encuentran **dos paneles para la exploración y gestión de la visualizació**n. El primer panel, situado en la parte superior, permite explorar cada una de las visualizaciones generadas en orden de ejecución. El segundo panel, situado en la parte inferior, permite obtener información del gráfico y su manipulación. Este último panel consta de tres pestañas: **(1)** un visor de la información del proceso de generación del gráfico (*logs*), así como de los posibles errores en el esquema; **(2)** los manejadores del gráfico, que permiten modificar las variables de estado definidas en el esquema; **(3)** el esquema final.

En el panel de exploración de visualizaciones se muestran todos los gráficos que hayan sido generados, en orden de generación. Este panel superior está conectado con el panel inferior, de forma que al pulsar sobre cada uno de los gráficos el panel inferior muestra la información de dicho gráfico y los manejadores para manipularlo. En la pestaña de los manejadores se puede simular lo que cualquier componente de visualización haría, que es modificar los valores de las variables de estado para que el gráfico se actualize en base a esos cambios (**Figura 3.85**).

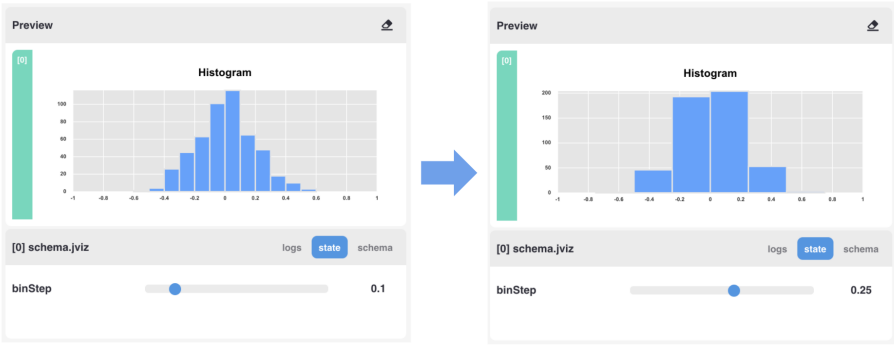


Figura 3.85. *Modificación de la variable de estado «binStep» en el histograma, utilizando un manejador de tipo deslizador. El panel está conectado con el gráfico por medio de la API de* ***jviz****, de forma que, al modificar la posición del deslizador, este se comunica con el gráfico para darle el nuevo valor de esta variable, haciendo que el gráfico se actualize.*

3.5 MGvizApps

3.5.1 **Introducción**

Para validar que los resultados de esta investigación sobre metodologías reactivas de visualización sirven para resolver necesidades de exploración de datos y por tanto cumplen el objetivo traslacional promovido por la actual política científica, se ha creado una plataforma de prototipos de aplicaciones focalizadas en biomedicina genómica usando los conceptos obtenidos como fruto de esta investigación.

Con ello se demuestra que este estudio **no se queda solo en resultados** técnicamente innovadores, sino que pueden dar como resultado aplicaciones adaptables y rápidamente configurables para dar respuesta a las necesidades de los laboratorios de investigación en biomedicina.

Necesidad biomédica

Para ello hemos recogido necesidades en diferentes campos de la biomedicina y propuesto una solución interactiva para mejorar el análisis exploratorio de datos en esos campos.

Durante el desarrollo de esta investigación se realizó un trabajo de campo para ver qué necesidades existían en los laboratorios que trabajaban con NGS. En este estudio se detectó, por un lado, la necesidad por parte de algunos usuarios de entornos clínicos de que las aplicaciones bioinformáticas utilicen datos únicamente accesibles en los ordenadores locales y, por otro lado, de una alta configuración para cada laboratorio que implica mantener diferentes versiones del lado cliente y del lado servidor sincronizadas.

Objetivo y solución bioinformática propuesta

Las necesidades expuestas anteriormente se alinean dentro del enfoque de uno de los objetivos de este trabajo, centrado en la usabilidad y desarrollo de visualizaciones y aplicaciones web reactivas que permitan al usuario la exploración interactiva de sus datos.

Para cubrir las necesidades mencionadas, se ha enfocado el desarrollo de estas aplicaciones de la siguiente manera:

- Estas aplicaciones deben estar centradas en el lado cliente. El uso de servicios REST se reduce al acceso de bases de datos públicas y sistemas de información biomédica que ayuden a

completar las anotaciones de las aplicaciones que se van a desarrollar.

- Las herramientas desarrolladas usarán elementos modulares, y se dará la máxima importancia a la configurabilidad de la aplicación para cada necesidad local. Esto reducirá el tiempo de desarrollo y permite seguir una filosofía de *Lean Development* y *Fast Prototyping* (Poppendieck y Poppendieck, 2003).
- El sistema a desarrollar deberá cumplir un objetivo doble. Por un lado, debe dar una solución rápida y eficaz a las necesidades de los investigadores en biomedicina. Por otro, ha de obtener una rápida retroalimentación del usuario final, con tal de generar aplicaciones más complejas siguiendo un modelo de *Lean Development*.

Como solución, se desarrolló una plataforma para la creación e integración de aplicaciones bioinformáticas para hacer análisis interactivo de datos (*Exploratory Data Analysis*, EDA) en NGS y que fueran versátiles, adaptables, de rápido desarrollo y que además se ejecuten en el lado cliente. A esta plataforma se le ha denominado **MGvizApps**.

3.5.2 **Resultados**

Este apartado describe los resultados de la creación de una plataforma bioinformática centrada en la visualización y el análisis exploratorio de los datos. Para ello, se muestra un conjunto de

aplicaciones que están orientadas a la resolución de problemas biomédicos muy concretos. Estas sirven de ejemplo para mostrar la importancia que tiene un diseño modular en entornos heterogéneos, donde los desarrollos requieren procedimientos extremadamente adaptables.

Con este fin, se han diseñado las aplicaciones como prototipos o pruebas de concepto. Su propósito es ayudar a definir el proceso a seguir a la hora de desarrollar aplicaciones que puedan ser interoperables con grandes sistemas *back end*, a través de protocolos REST o HL7, para su integración en grandes redes de sistemas de salud.

En resumen, **MGvizApps** ha sido desarrollada basándose en la idea de la modularidad, apoyándose en los componentes mostrados en este capítulo y en el sistema de visualización reactiva (**jviz**) desarrollado en el capítulo 3.4. Esta plataforma prima la rapidez de desarrollo, la personalización y la configuración rápida según las necesidades del usuario, permitiendo además el uso de los datos locales para evitar problemas de custodia legal o burocráticos por los reglamentos de protección de datos, como, por ejemplo, el RGPD (Reglamento General de Protección de Datos).

Este apartado de resultados se divide en dos partes: (1) los desarrollos tecnológicos necesarios para la creación de la plataforma y sus funcionalidades; (2) las aplicaciones propiamente dichas.

Desarrollos tecnológicos

Como se ha mencionado en la introducción, la plataforma **MGvizApps** está basada, por un lado, en el desarrollo del sistema modular para la visualización y UX mostrado en s anteriores. Pero, por otro lado, también ha necesitado de desarrollos tecnológicos propios para permitir un uso local sin mediación de un servidor. Los principales desarrollos tecnológicos de este apartado han sido: **(1)** la lectura de archivos por fragmentos e indexación para el acceso directamente más tarde; **(2)** el uso de procesado paralelo en segundo plano (*workers*); **(3)** el almacenamiento en el navegador web; **(4)** la generación de informes editables desde el lado cliente sin transferencia de datos a un servidor externo.

Lectura de archivos por fragmentos

Para permitir la lectura de archivos grandes a través de las aplicaciones web, se ha desarrollado un módulo de lectura propio basado en la fragmentación del mismo en bloques o *chunks*. Este módulo minimiza el uso de memoria del navegador, ya que permite la lectura de grandes archivos sin tener que cargarlos en su totalidad en la memoria (evitando así el enlentecimiento o bloqueo del navegador). Con este módulo, se han podido satisfacer dos necesidades imprescindibles: la lectura de archivos línea a línea y la lectura de archivos TAR.

Lectura de archivos línea a línea

La mayoría de archivos utilizados en bioinformática (VCF, SAM, GFF, etc.), pese a tener un formato diferente, comparten la misma filosofía de que cada uno de los elementos que contienen está almacenado por líneas. Por ello, la mejor forma de leer y procesar este tipo de archivos es realizando la lectura línea a línea.

Lectura de archivos TAR

El formato de archivos TAR (acrónimo de *Tape Archiver*) recibe su nombre por su uso original, el cual consistía en la unificación de múltiples archivos en uno sólo para simplificar el proceso de almacenamiento de archivos en cintas magnéticas. Los archivos TAR se basan en la escritura por bloques de 512 bytes: en el primer bloque, cada archivo contiene una cabecera, en la que se almacena información del mismo (nombre, tamaño, tipo de archivo, etc.); los bloques a continuación corresponden al contenido del archivo, cuyo tamaño se ajusta a un múltiplo de 512 bytes mediante un último bloque con relleno si hiciera falta (**Figura 3.86**).

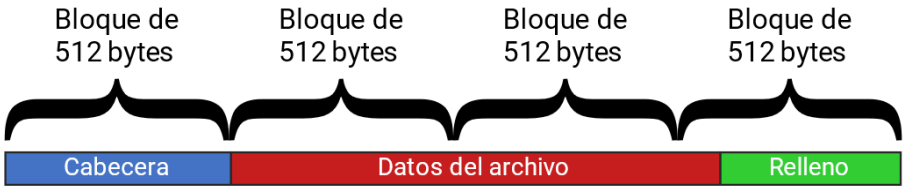


***Figura 3.86**. Estructura por bloques de un archivo en un **TAR**. El bloque de los primeros 512 bytes se corresponde con la cabecera, mientras que el resto de bloques se utilizan para almacenar el contenido del archivo, añadiendo bytes extra para que el tamaño total sea un múltiplo de 512.*

Mediante el módulo de lectura por fragmentos, se puede realizar la indexación de todos los archivos contenidos en el TAR de la siguiente manera:

1. Se leen los primeros 512 bytes del TAR (la cabecera del primer archivo contenido).
    a. De la cabecera, se extrae el nombre del archivo, su tipo y su tamaño.
    b. El nombre y el tamaño, así como el índice del bloque del TAR, se utilizan para generar el índice.
2. Se ignoran los siguientes bloques de 512 bytes del TAR, tantos como corresponda según el tamaño del archivo.
3. Si no se ha llegado al final del archivo TAR, se vuelve al primer paso continuando desde el bloque donde termina el archivo.

De esta forma, obtendremos un listado de todos los archivos contenidos en el TAR sin haber leído todo el fichero. Cada uno de los elementos del índice generado almacena la información mínima necesaria para que, cuando se requiera, se pueda extraer el contenido de un archivo concreto sin tener que volver a buscarlo en todo el TAR.

Las aplicaciones de la plataforma **MGvizApps** saben navegar y acceder a ficheros en un árbol de directorio de un análisis archivado en un TAR (**Figura 3.87**). Este procedimiento facilita al usuario el manejo de los datos como un único fichero manteniendo toda la potencia y expresividad de una estructura de árbol de directorio para su organización y categorización dentro de la aplicación, de forma inmediata y transparente.

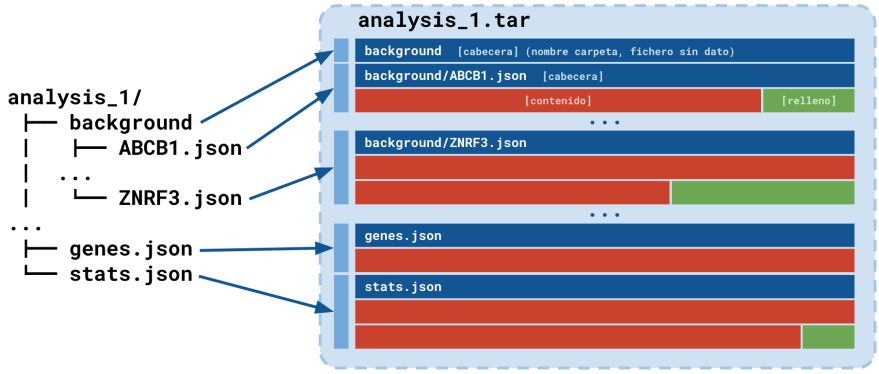


Figura 3.87. *Almacenamiento de una estructura de carpetas en un TAR. Cualquier software de análisis compatible con* ***MGvizApps*** *puede crear directorios con los resultados, y archivar toda esa estructura en un TAR. Esto permite a las aplicaciones de* ***MGvizApps*** *el acceso a los diferentes archivos bajo demanda para su visualización una vez vinculado el TAR a dicha aplicación.*

Procesado en segundo plano en paralelo (Web Workers)

Un obstáculo todavía presente en el desarrollo de aplicaciones web es la ejecución de JavaScript en un único subproceso del navegador. Esto significa que no se pueden ejecutar varios scripts al mismo tiempo. Pongamos como ejemplo una aplicación web que necesite manipular y manejar eventos del UI, consultar y procesar grandes cantidades de datos de API y, además, manipular el DOM. Deberá realizar todas estas tareas en un único subproceso, lo cual no puede ser simultáneo debido a limitaciones en el tiempo de ejecución de JavaScript de los navegadores.

Para solventar este problema se ha desarrollado un módulo que, utilizando la tecnología de los *Web Workers* (**Figura 3.88**), permite

ejecutar tareas en un segundo plano. Los *Web Workers* se ejecutan en un subproceso aislado y están diseñados para efectuar tareas más costosas, sin que esto afecte al rendimiento del código JavaScript ejecutado en el subproceso principal.

***Figura 3.88**. Flujo del funcionamiento de los Web Workers. El proceso principal (aplicación) crea un nuevo worker, al cual se le asigna una tarea a realizar (script). Los resultados de la ejecución de dicha tarea en el worker son comunicados a la aplicación, que, tras recibir el último mensaje de finalización de la tarea, destruye el worker.*

Almacenamiento en el navegador web

Toda aplicación de análisis necesita una forma de poder almacenar información en el sistema de forma permanente, para poder retomar los análisis ya realizados o para mantener los estados de las configuraciones. En casos así conviene crear un servicio REST que provea de dicha capa de acceso a estos datos, con acceso a una base de datos para almacenarlos en el servidor. Para ciertas aplicaciones

que requieren realizar análisis rápidos y una sola vez con los mismos datos, se puede reducir la complejidad de la aplicación centrándose solo en la parte del cliente y delegar las tareas típicas del servidor a módulos del cliente que almacenen datos de forma local en el navegador, así como accediendo a los archivos locales del ordenador del usuario que está utilizando la aplicación.

Para el uso de estos sistemas de almacenamiento local del navegador, se ha desarrollado un módulo [42]que facilita el acceso y la manipulación de los datos contenidos en estos sistemas de almacenamiento (aunque de momento solo permite trabajar con la base de datos *IndexedDB*, el objetivo es ampliarlo para que se pueda utilizar con otros sistemas como, por ejemplo, *Local Storage*).

Generación de informes editables desde el cliente

La plataforma **MGvizApps** posee un módulo de apoyo a la toma de decisiones (*Decision Support System*, DSS) para la creación de informes. Este módulo consta de: **(1)** tablas resumen para selección de evidencias y valores susceptibles de ser informados (**Figura 3.89**:1); **(2)** un sistema de entrada de texto libre con etiquetación automática de palabras clave, además de la realizada por el usuario (**Figura 3.89**:2); **(3)** una previsualización del resultado previo a la descarga en formato MS WORD (**Figura 3.89**:3).

[42] Código del módulo de gestión del almacenamiento en el navegador: https://github.com/MGvizPro/MGvizApps/blob/develop/packages/common/src/utils/database.js

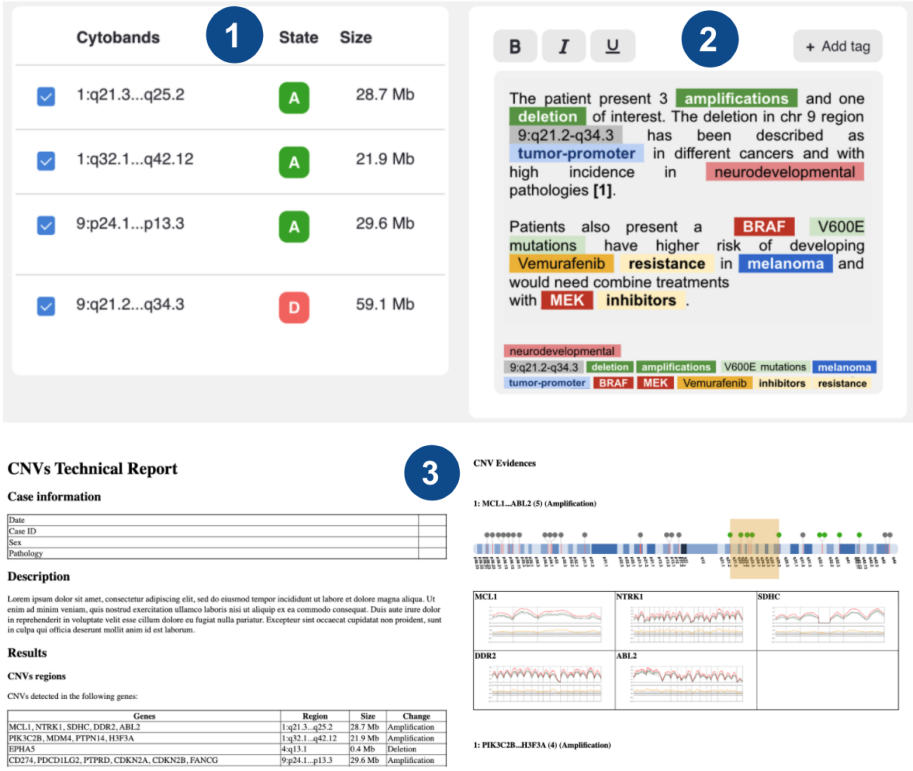


*Figura 3.89. Módulo de Decisión Support System (DSS) en la plataforma **MGvizApps**, formado por: **(1)** tablas resumen; **(2)** sistema de entrada con etiquetación; **(3)** generación del informe final.*

VISMapper

La aplicación **VISMapper** de MGvizApps es una reimplementación del **VISMapper** original utilizando toda la tecnología y componentes desarrollados como fruto de este trabajo. Esta aplicación permite realizar la exploración de los lugares de inserción vírica a partir de un

archivo SAM resultante de todo el proceso de alineamiento, lo que en la aplicación original se hacía automáticamente en el servidor.

El código de **VISMapper** está libremente accesible en *GitHub*[43] y se puede acceder desde la web de **MGvizApps**[44].

Novedades tecnológicas

Esta reimplementación de **VISMapper** se fundamenta en dos ejes principales: (1) el acceso a los datos de forma local y (2) la creación de la plataforma interactiva basándose en los componentes de **MGviz** y gráficos reactivos de **jviz.**

Para el desarrollo de esta aplicación se ha creado un sistema de lectura e indexación de archivos SAM desde el lado del cliente y se ha desarrollado un nuevo componente de **MGviz** para visualización y exploración de regiones genómicas.

En esta aplicación se ha usado el sistema de lectura e indexado de ficheros locales para desarrollar un lector de ficheros SAM que permite un rápido acceso al fichero local para obtener la selección de lecturas de secuenciación que cumplan los requisitos marcados por la

[43] Código de la aplicación **VISMapper** en el repositorio de *GitHub*: https://github.com/MGvizPro/MGvizApps/tree/develop/packages/vismapper

[44] Aplicación **VISMapper**: https://dev.apps.mgviz.org/#!/apps/vismapper

aplicación. Se ha diseñado un indexador de SAM[45] que se ejecuta dentro de un *worker* del navegador, permitiendo así el procesado en paralelo de dicho archivo (**Figura 3.88**). Esto facilita que la aplicación pueda computar todos los lugares de inserción a partir de las lecturas del archivo SAM sin tener que cargarlo íntegramente en memoria. El flujo utilizado por este indexador de SAM es el siguiente (**Figura 3.90**):

1. En primer lugar, el archivo SAM es leído línea a línea por el *worker* de la aplicación.
2. Cada una de las líneas es procesada y utilizada para generar los lugares de inserción y el índice.
3. Una vez el archivo SAM ha sido leído y procesado por completo, el listado de lugares de inserción final y el índice generado son devueltos a la aplicación web.
4. Cuando en la aplicación se selecciona un lugar de inserción a visualizar, se utiliza el índice para obtener las posiciones del archivo SAM en las que se encuentran las lecturas correspondientes a dicho lugar de inserción.
5. Tras acceder al fragmento del archivo SAM que contiene dichas secuencias, se visualiza el lugar de inserción y las lecturas en la aplicación.

[45] Código con la lógica del procesado del SAM en **VISMapper**: https://github.com/MGvizPro/MGvizApps/blob/develop/packages/vismapper/workers/findInsertionSitesSam.js

El otro de los elementos tecnológicos importantes desarrollados para esta reimplementación es la integración de los componentes que permiten la exploración de elementos genómicos (*Feature Track*, *ScaleLegend Track*) formando un visor genómico reactivo, ya que los servicios REST y componentes de terceros que se usaban para la visualización genómica en la aplicación original ya no están disponibles.

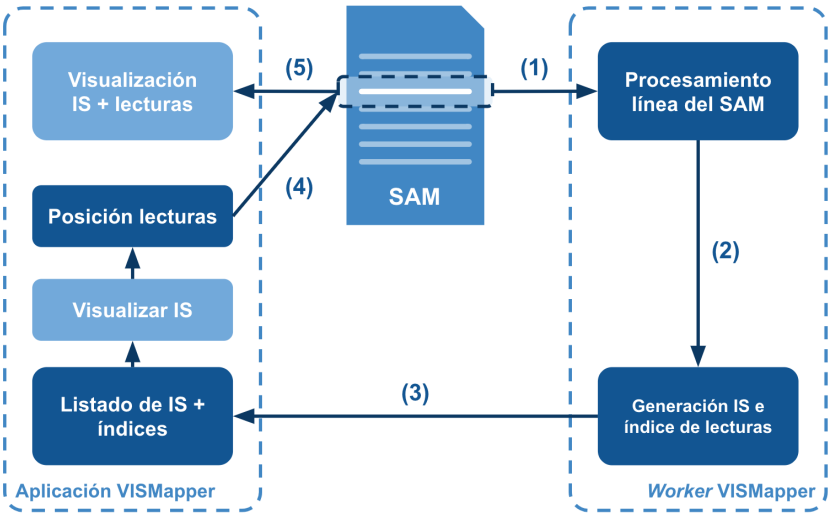


***Figura 3.90**. Esquema de flujo del procesado y acceso a los archivos locales indexados. **(1)** El archivo SAM es leído y procesado línea a línea por el «worker» de la aplicación. **(2)** Cada una de las líneas es utilizada para la generación de los lugares de inserción y del índice. **(3)** El listado de lugares de inserción y el índice generados son devueltos a la aplicación web. **(4)** Acceso mediante el índice generado a la posición correspondiente del IS en el SAM **(5)** Obtención de las lecturas para su visualización.*

Análisis exploratorio de inserción de secuencias víricas

La aplicación **VISMapper** contiene diferentes vistas con varios de los módulos para análisis exploratorio de datos genómicos (**Figura 3.91**):

1. Cariotipo con la distribución de los lugares de inserción detectados.
2. Ideograma de un cromosoma mostrando los hitos de elementos genéticos de interés.
3. Tabla de elementos genómicos con la información de los lugares de inserción.
4. Visor genómico para la previsualización del contexto en el que se sitúan los lugares de inserción en el genoma.

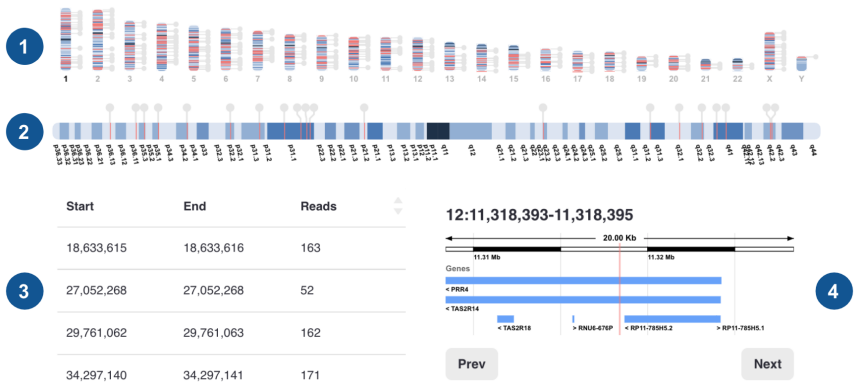


*Figura 3.91. Galería de componentes utilizados en **VISMapper**. **(1)** cariotipo; **(2)** ideograma de un cromosoma mostrando los hitos de elementos genéticos de interés; **(3)** tablas con anotación de regiones genómicas; **(4)** previsualización del contexto genómico de un elemento genómico.*

VISMapper está orientada como una herramienta para analizar visualmente de forma interactiva los datos producidos por datos de secuenciación NGS donde existan inserciones de virus.

La aplicación necesita como datos de entrada un archivo de alineamientos genómicos en formato SAM [46], el cual debe ser proporcionado en la pantalla inicial de la aplicación (**Figura 3.92**).

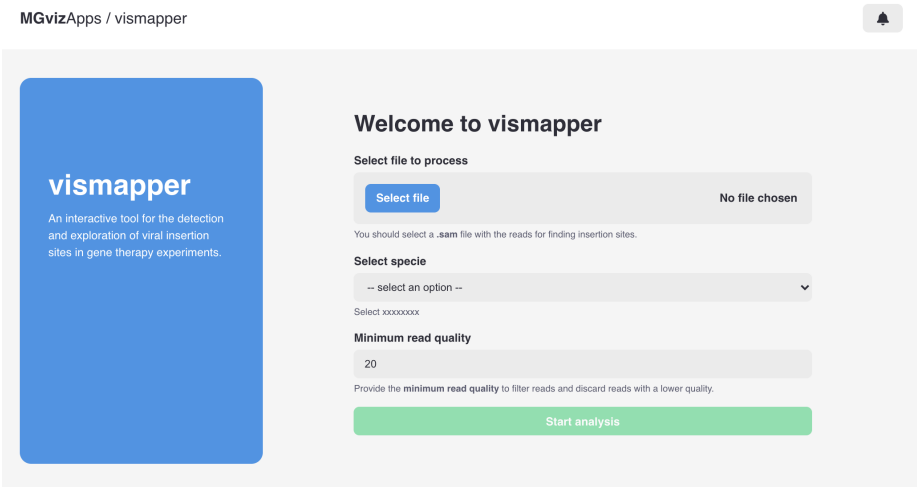


Figura 3.92. *Vista de entrada a la aplicación para cargar el archivo SAM con toda la información de NGS.*

Una vez se ha cargado el archivo SAM y se define el umbral de la calidad de lectura mínima, la aplicación procede a la búsqueda de los lugares de inserción de virus, utilizando el mismo algoritmo que se desarrolló para la publicación original (Juanes *et al.,* 2017). Como el

[46] Especificación del formato SAM: https://samtools.github.io/hts-specs/SAMtags.pdf

objetivo de estas aplicaciones es resaltar la usabilidad y la importancia del sistema modular que se ha desarrollado como parte de este trabajo, se ha dejado como trabajo futuro la mejora de estos algoritmos de detección y su extensión a otros dominios (por ejemplo, terapia génica con plásmidos en nematodos, inserción de transposones y *off-targets* de CRISPR).

Una vez generada toda la información relativa a los lugares de inserción a partir del archivo local proporcionado, estos datos se visualizan de forma interactiva para permitir la exploración de las inserciones por cromosoma y visualizar su contexto genómico (**Figura 3.93**).

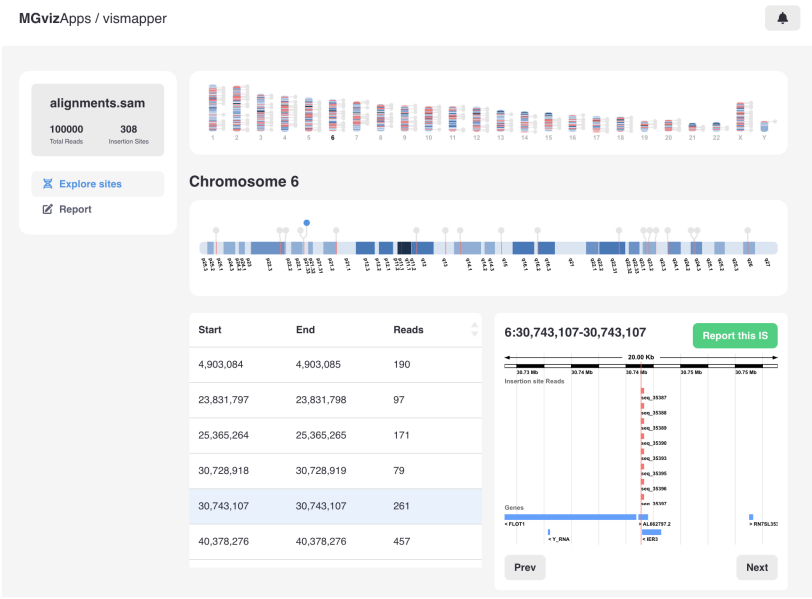


***Figura 3.93**. Cuadro de mandos de la nueva versión de **VISMapper**. El cariotipo superior muestra todos los lugares de inserción en cada uno de los cromosomas. Pulsando sobre cualquiera de los cromosomas del cariotipo, se*

muestra el cromosoma en cuestión con sus lugares de inserción, además de una tabla con información de cada uno de ellos. Al pulsar sobre un lugar de inserción, se mostrará en el visor genómico el contexto en el que se encuentra.

SeqMask

La aplicación **SeqMask** nace de la necesidad de en el laboratorio de enmascarar las bases con variantes comunes para evitar diseñar cebadores en esas zonas, ya que mermaría la eficacia de la amplificación de la Reacción en Cadena de la Polimerasa (*PCR*) del alelo con la variante.

El código de la aplicación **SeqMask** está libremente accesible en *GitHub*[47] y se puede acceder desde la web de **MGvizApps**[48].

Para la construcción de la aplicación **SeqMask** se utilizaron diversos componentes (**Figura 3.94**): **(1)** cariotipo para la localización de los genes solicitados; **(2)** ideograma con la localización específica de los genes; **(3)** componente para la representación en contexto genómico de los exones o variantes; **(4)** herramienta de selección de transcritos;

[47] Código de la aplicación **CNVReporter** en el repositorio de GitHub: https://github.com/MGvizPro/MGvizApps/tree/develop/packages/seqmask

[48] Acceso a **SeqMask**: https://dev.apps.mgviz.org/#!/apps/seqmask

(5) visualizador de secuencias; **(6)** exportación de las secuencias en WORD.

Figura 3.94. *Diferentes componentes utilizados en la aplicación* ***SeqMask****: **(1)** cariotipo; **(2)** ideograma; **(3)** visor genómico; **(4)** selector de transcritos; **(5)** visualizador de secuencias; **(6)** exportación a WORD.*

La aplicación permite la generación y enmascaramiento de secuencias, a partir de transcritos de genes o regiones específicas. Para facilitar el acceso a las secuencias ya generadas, la aplicación hace uso del almacenamiento local en el navegador para guardar todas estas secuencias, permitiendo el acceso posterior. En la primera vista de la aplicación (**Figura 3.95**) se muestra el listado de estas secuencias, agrupadas en análisis, junto con el estado en el que se encuentran (en cola, en ejecución, completadas o fallidas).

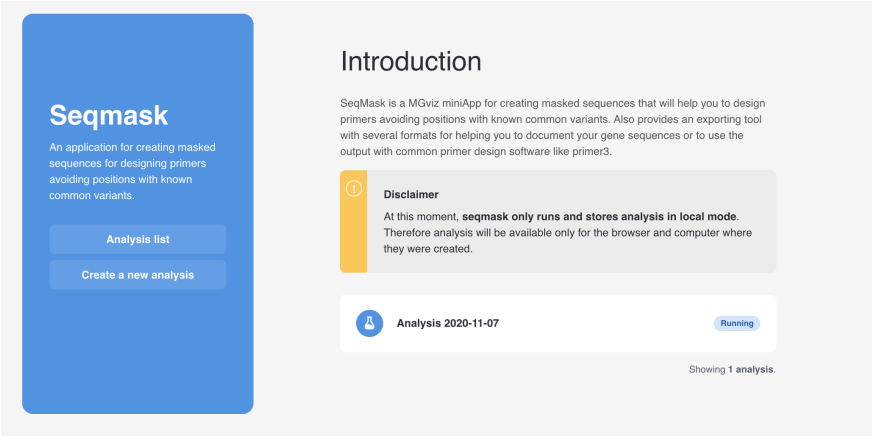


__Figura 3.95__. Vista inicial de la aplicación __SeqMask__. En el lateral izquierdo se muestran dos botones para acceder al listado de secuencias creadas (análisis) y para crear una nueva. En la parte derecha se muestra una breve descripción de la aplicación y el listado de secuencias creadas con su estado.

Para la generación de nuevas secuencias, la aplicación proporciona una vista (**Figura 3.96**) en la que se debe proporcionar cierta información: **(1)** nombre del análisis a crear, utilizado para distinguir entre diferentes análisis; **(2)** especie a utilizar para la obtención de las secuencias y de las variantes; **(3)** listado de exones o regiones de interés; **(4)** valor mínimo de frecuencia alélica, para enmascarar aquellas variantes cuya frecuencia alélica sea mayor o igual que la proporcionada; **(5)** distancia máxima en pares de bases que deben tener dos regiones para que se junten en una única.

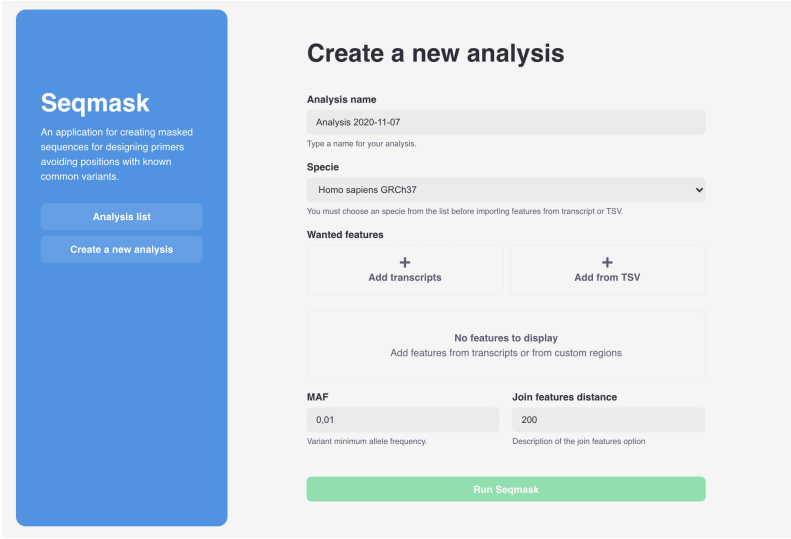


***Figura 3.96**. Vista de la pantalla de creación de un nuevo análisis en **SeqMask**.*

Para la selección de transcritos se ha desarrollado el componente *TranscriptSelector*, que permite la selección del transcrito (o transcritos) de interés a partir de un gen concreto (**Figura 3.96**). Una vez especificado el gen en el cuadro de búsqueda, el componente se conecta con el servicio Biomart de Ensembl (Yates *et al.,* 2019) para solicitar la estructura de transcritos y exones de dicho gen, los cuales son mostrados al usuario en la parte inferior del cuadro de búsqueda (**Figura 3.97**). Para cada transcrito del gen solicitado, se muestra: **(1)** su estructura de exones; **(2)** el identificador del transcrito; **(3)** en el caso de que se trate de un transcrito de referencia, se mostrará su identificador de RefSeq.

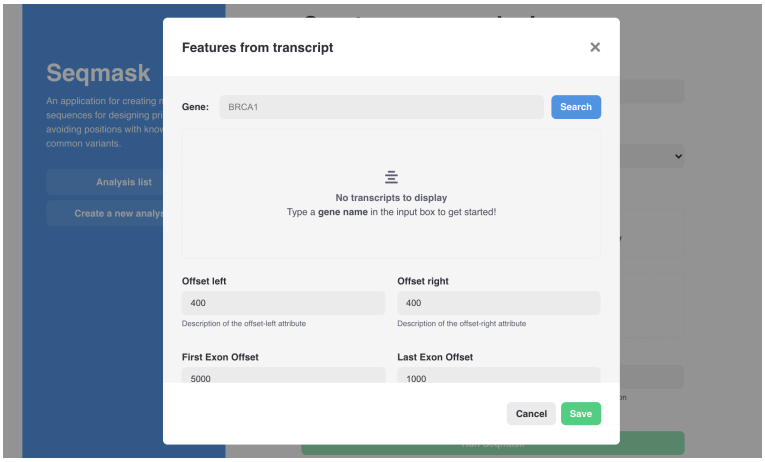


*Figura 3.97. Vista del componente **TranscriptSelector**.*

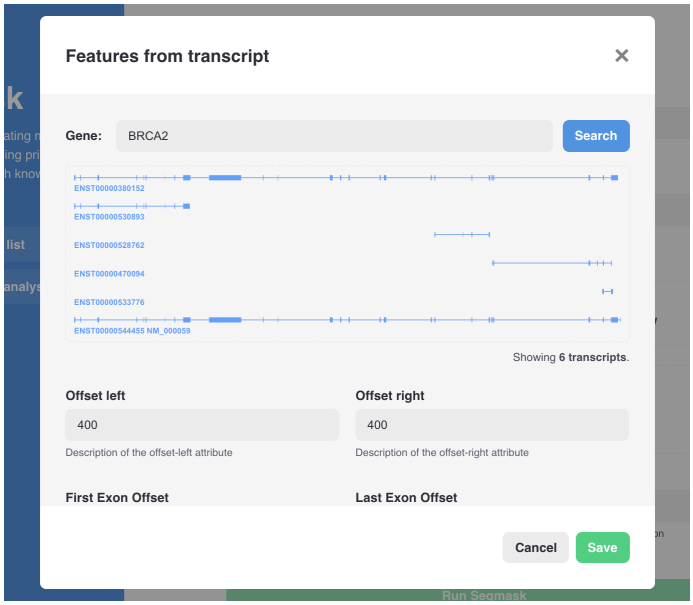


*Figura 3.98. Selección del transcrito de interés a partir del nombre de un gen, en este caso de **BRCA2**.*

El componente *TranscriptSelector* permite escoger entre uno o varios de los transcritos mostrados para el gen especificado. En el caso de que más de un transcrito sea seleccionado se generará el *metatranscrito*, formado por la unión de todos los exones de los exones de los transcritos seleccionados.

Una vez seleccionados los transcritos de interés, se mostrará en la pantalla de creación del análisis el listado con todos los exones del transcrito (o *metatranscrito*) del gen seleccionado (**Figura 3.99**).

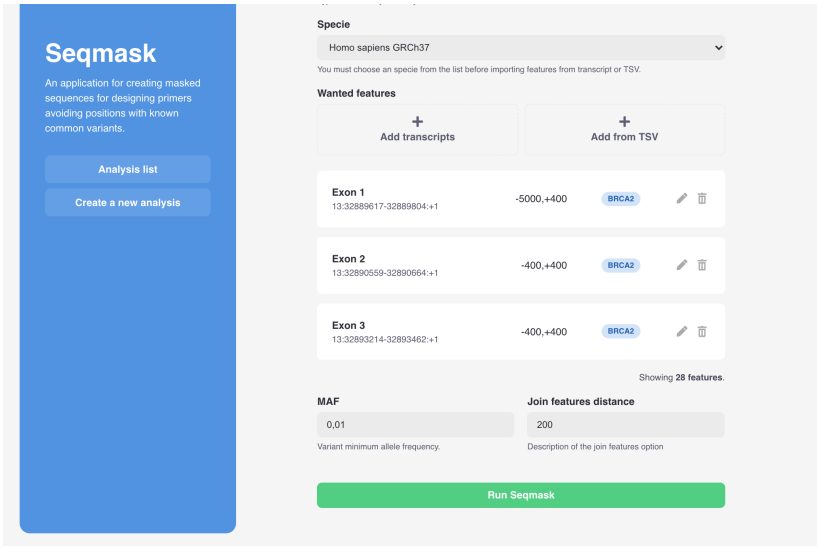


Figura 3.99. *Listado de exones del gen* ***BRCA2*** *obtenidos a partir de uno de sus transcritos.*

Tras completar la generación de las secuencias, **SeqMask** permite la exploración de todas ellas, agrupadas por exones (más un extra de secuencia a cada lado). Además, todas estas secuencias tendrán enmascaradas con una «N» las variantes que tengan una frecuencia

alélica mayor al valor determinado en la pantalla de configuración (0.01 por defecto) (**Figura 3.100**).

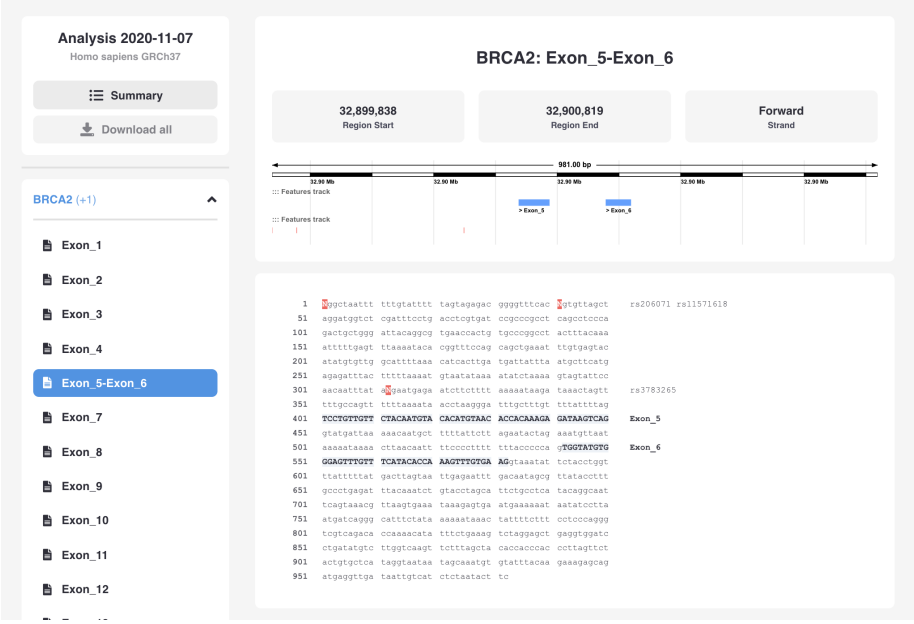


Figura 3.100. *Vista de los exones y secuencia con las variantes comunes enmascaradas.*

Con esta aplicación se introdujeron principalmente mejoras en la usabilidad. Por ejemplo, aunque la secuencia se visualiza en secciones separadas de 10 nucleótidos, internamente es una secuencia continua y por lo tanto, acciones básicas como la búsqueda de fragmentos mediante el diálogo de búsqueda del navegador (**Figura 3.101**) o el seleccionar y copiar fragmentos específicos (**Figura 3.102**) no se ven afectadas.

cctaagaactcatacaNccagNaccctggagtcgattgattagagcctagtccaggagaa
tgaattgacactaatctctgcttgtgttctNtgtctccagCAATTGGGCAGATGTGTGAG
GCACCTGTGGTGACCCGAGAGTGGGTGTTGGACAGTGTAGCACTCTACCAGTGCCAGGAG
CTGGACACCTACCTGATACC

Figura 3.101. *La selección de un fragmento de secuencia con el cursor es continua cuando se pega en otro documento.*

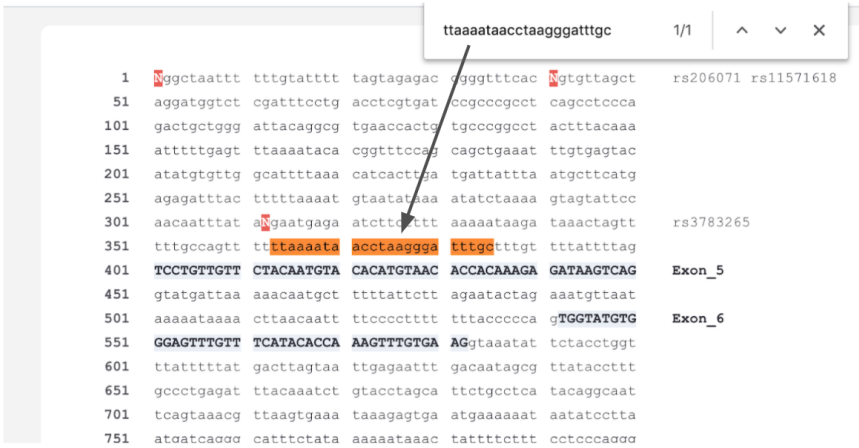


Figura 3.102*. Ejemplo de búsqueda de una secuencia de ADN en el visor de secuencias, donde estas son mostradas en bloques de 10 bases.*

Por último, la aplicación también permite la exportación de todas las secuencias de un gen a **FASTA** o a **MS WORD** (**Figura 3.103**).

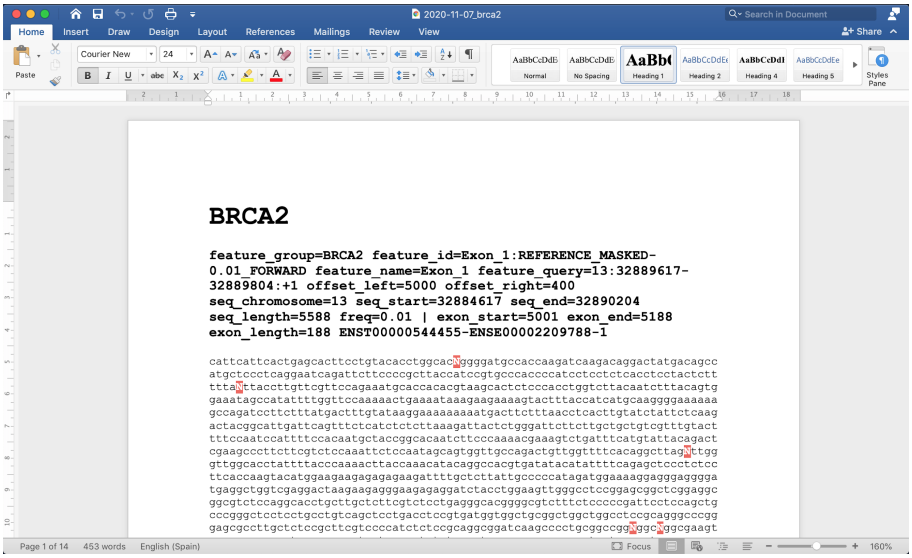


*Figura 3.103. Captura del archivo **MS WORD** generado en **SeqMask**.*

CNVReporter

CNVReporter es una aplicación de **MGvizApps** dedicada a la exploración reactiva y creación de informes para Variantes de Número de Copia (*CNVs*) de forma interactiva.

CNVReporter está diseñada para ayudar al genetista clínico a evaluar y anotar análisis de CNVs. La herramienta se basa en la detección de diferencias de cobertura entre una muestra germinal y una tumoral del

mismo individuo o de una muestra tumoral comparada contra un conjunto de muestras control.

Esta aplicación dispone de varias vistas que siguen un flujo de exploración de datos. En resumen, la aplicación permite determinar interactiva y gráficamente el estado de las regiones cromosómicas estudiadas y crear informes a los genetistas clínicos.

El código de **CNVReporter** está libremente accesible en *GitHub*[49] y se puede acceder desde la web de **MGvizApps**[50].

Este prototipo ha servido para el estudio y desarrollo de un sistema de acceso a ficheros TAR en el sistema de archivos local y para poder acceder a ese fichero posteriormente desde la aplicación en cualquier momento, de forma transparente y de igual forma que si se hubiera subido a un servidor.

Otros de los desarrollos importantes introducidos en esta aplicación fueron la revisión de datos interactivamente (modificación de los estados de los CNVs por parte del usuario) y la generación de informes editables desde el lado cliente.

La aplicación **CNVReporter** contiene diferentes vistas con varios de los módulos básicos para análisis exploratorio de datos genómicos (**Figura 3.104**): **(1)** cariotipo; **(2)** boxplot para el ratio de una señal; **(3)**

---

[49] Código de la aplicación **CNVReporter** en el repositorio de *GitHub*: https://github.com/MGvizPro/MGvizApps/tree/develop/packages/cnvreporter

[50] Acceso a **CNVReporter**: https://dev.apps.mgviz.org/#!/apps/cnvreporter

ideograma de un cromosoma mostrando los hitos de elementos genéticos de interés; **(6)** tablas con anotación de regiones genómicas; **(4)** previsualización de la cobertura de genes; **(5)** estructura del transcrito y exones; **(7)** sistema de generación de informes editables; **(8)** módulo de exportación a **MSWord** y **PDF**.

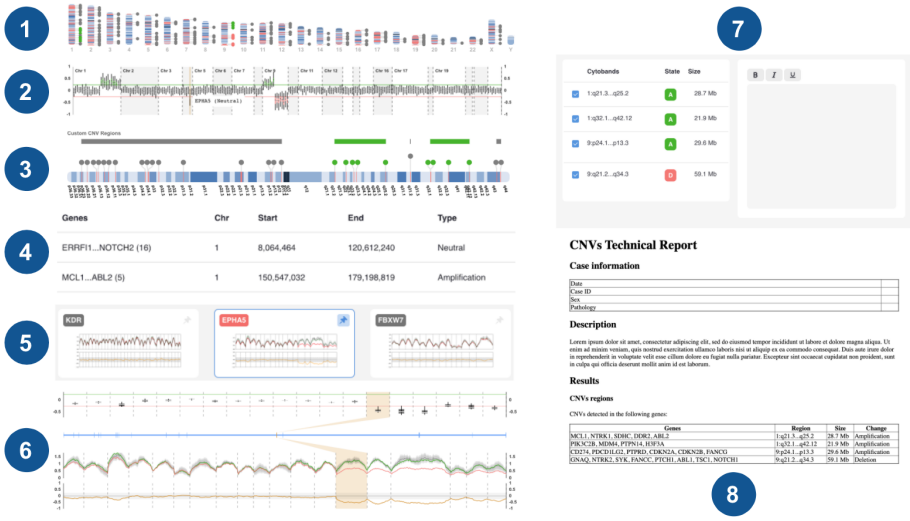


*Figura 3.104. Galería de componentes utilizados en **CNVReporter**. **(1)** cariotipo; **(2)** boxplot para el ratio de una señal; **(3)** ideograma de un cromosoma mostrando los hitos de elementos genéticos de interés; **(6)** tablas con anotación de regiones genómicas; **(4)** previsualización de la cobertura de genes; **(5)** estructura del transcrito y exones; **(7)** sistema de generación de informes editables **(8)** módulo de exportación a **MS Word** y **PDF**.*

CNVReporter está orientada como una herramienta para analizar visualmente de forma interactiva los datos producidos por la *pipeline*

de medicina de precisión oncológica **MGVizCNV**[51] o por cualquier aplicación de detección de CNVs tras aplicarle un normalizador de datos para convertir los datos al estándar de **MGviz**.

La aplicación necesita unos datos de entrada de valores de cobertura de las secuencias de estudio y de datos controles para las comparaciones. Estos datos se deben cargar en la pantalla inicial de la aplicación (**Figura 3.105**) mediante un archivo TAR, el cual contiene todos los archivos de coberturas y estadísticas. Este archivo TAR no será cargado por completo en memoria, sino que se generará un índice de los archivos que contiene, lo cual permitirá a la aplicación acceder a cada uno de estos archivos posteriormente conforme lo necesite.

[51] Poster de **MGvizCNV**: https://figshare.com/articles/poster/MGvizCNV_a_QC_Machine_Learning_approach_for_CNV_evidence_scoring/12895778

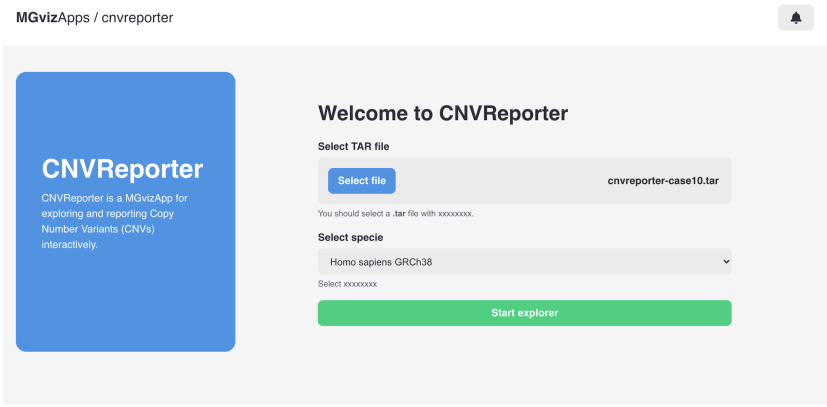


Figura 3.105. *Pantalla de entrada a la aplicación* ***CNVReporter****.*

Una vez cargados los datos, la aplicación, en ausencia de una interpretación de CNVs previa, se asigna el estado de CNV a cada uno de los genes automáticamente según el *log2ratio* de la cobertura del gen frente a su control germinal o el de una población control. Los umbrales para la clasificación del CNV se pueden cambiar de forma interactiva con los manejadores en el panel izquierdo. Una vez calculado el estado de cada gen, este se actualiza en todos los paneles (cariotipo, boxplot y tabla) (**Figura 3.106**)

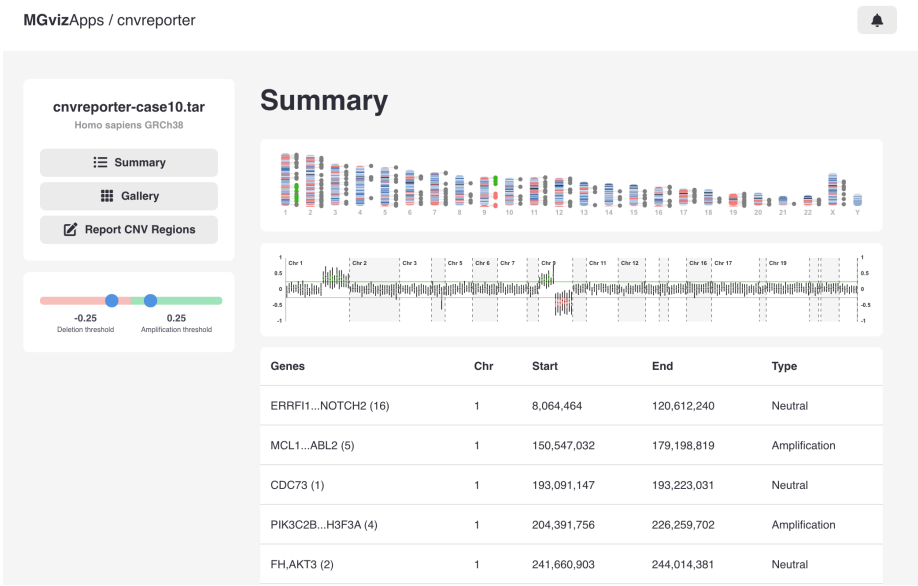


***Figura 3.106**. Pantalla principal de la aplicación **CNVReporter**. A la izquierda se muestra el nombre del archivo TAR cargado, así como la especie y los botones que permiten el acceso a cada una de las vistas de la aplicación. También se muestran los manejadores para ajustar los umbrales de detección de CNV. En la parte derecha se muestra el contenido de la vista resumen (cariotipo, boxplot y tabla de regiones con CNVs).*

Esta vista resumen muestra un panel de *boxplots* con los valores de los *log2ratio* de coberturas de cada base agrupados por gen para mostrar rápidamente cuál es la inestabilidad cromosómica del caso, y permite detectar rápidamente grandes amplificaciones (MYC), deleciones parciales (**Figura 3.107**:1) o isocromosomas (**Figura 3.107**:2).

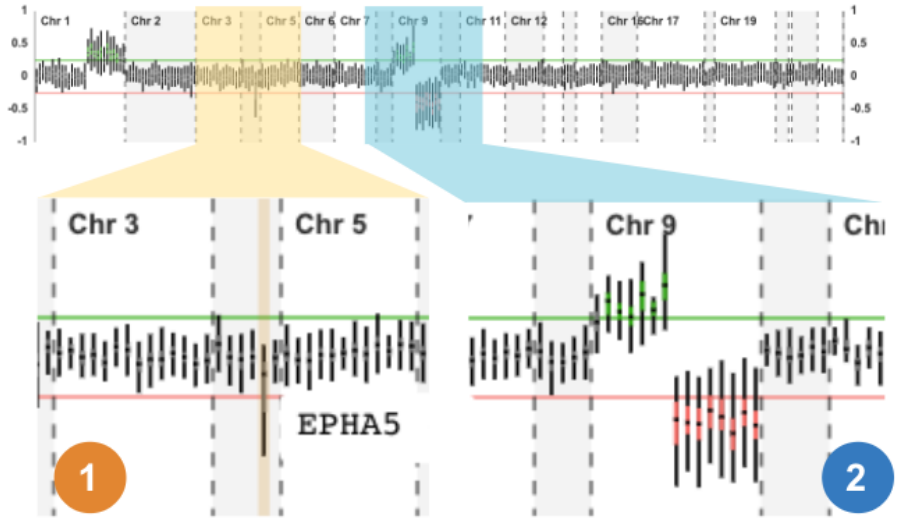


Figura 3.107. *Detalle de un gen* ***parcialmente delecionado*** *(EPHA5, en el cromosoma 4) y un* ***isocromosoma*** *(cromosoma 9).*

Una vez ajustados los niveles de umbral para detectar los CNVs según la pureza del tumor y variación técnica del experimento, queda ajustar el estado de CNV de los genes individualmente por cromosoma. En la **Figura 3.107**:2 se ve que el primer gen del cromosoma 9 no queda seleccionado y en la **Figura 3.107**:1 se aprecia que el gen EPHA5 del cromosoma 4 posiblemente esté parcialmente delecionado y necesita revisión. Pulsando en el cromosoma correspondiente del *boxplot*, cariotipo o región de la tabla, se accede a la vista del cromosoma para una exploración más detallada del análisis (**Figura 3.108**).

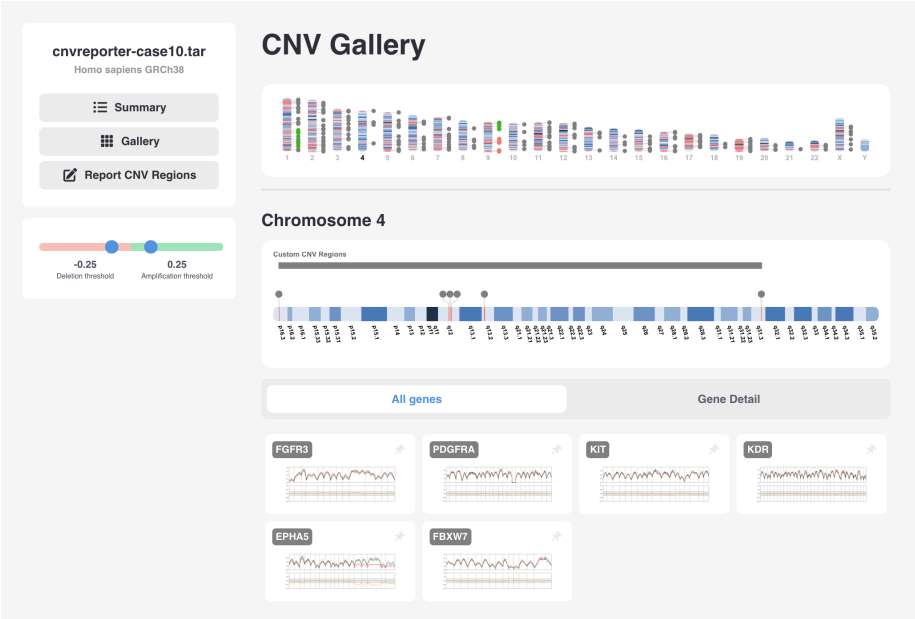


Figura 3.108. Vista en la que se muestra la distribución de los genes del cromosoma 4, junto con el perfil de cobertura para cada uno ellos

Si visualmente se observa que un gen no tiene un patrón de cobertura concordante al estado seleccionado, como en este caso el gen EPHA5 que se aprecia una posible deleción en la parte (**Figura 3.109**) se puede pasar a una vista más detallada de dicho gen (**Figura 3.110**).

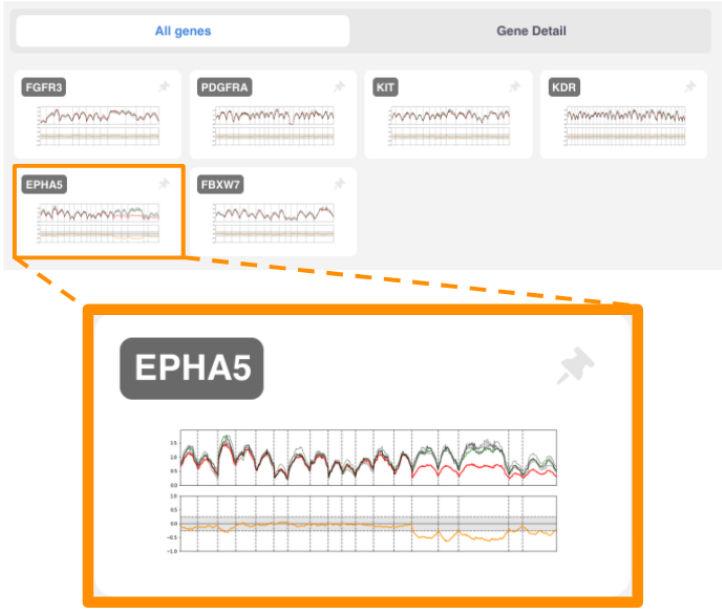


Figura 3.109. *Ampliación de la vista de la tarjeta del gen EPHA5 donde se aprecia su deleción parcial. La cobertura del tumor (rojo) por debajo del germinal (verde) y controles poblacionales (gris).*

La vista más detallada del gen usa un componente gráfico reactivo del **jviz** de tres paneles, donde el área del exón en el *boxplot*, transcrito y cobertura quedan resaltados a medida que el puntero pasa por encima de cada exón. En esta vista detallada se puede actualizar el valor del estado del CNV (**Figura 3.111**). Al modificar el valor del estado del gen se lanza un evento, capturado por el controlador principal que gestiona el estado de cada gen y las regiones. Una vez el estado del gen ha sido actualizado y las regiones de CNV han sido computadas de nuevo, este controlador lanza un nuevo evento que es escuchado por todos los paneles de esta vista, los cuales se actualizan mostrando el nuevo estado del gen y las nuevas regiones de CNV computadas (**Figura 3.112**).

*Figura 3.110. Vista detallada del gen **EPHA5**, en el cromosoma 4.*

***Figura 3.111**. Modificación manual del estado del gen **EPHA5** a delecionado, utilizando el menú desplegable en la vista detallada del gen.*

*Figura 3.112. Resultado de la modificación manual del estado del gen EPHA5*Una vez el genetista clínico ha finalizado la exploración de los genes de interés, pasará a la vista de generación del informe (**Figura 3.113**), que permite la selección de las regiones de CNVs a reportar y la redacción de algunas de las secciones del informe final. Además, y previo a la descarga del informe en MS WORD, podrá previsualizar el informe final y las evidencias de las regiones reportadas (**Figura 3.114** y **Figura 3.115**).

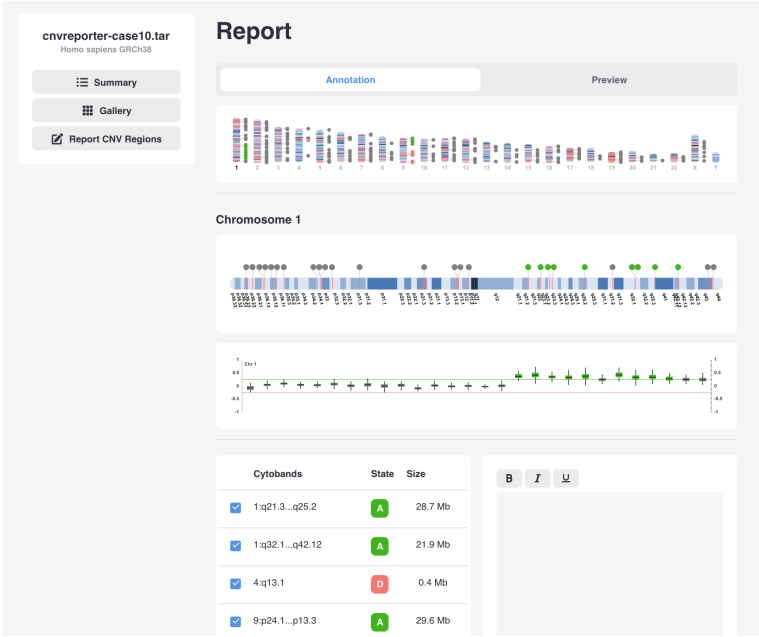


Figura 3.113. *Vista resumen y generación del informe final.*

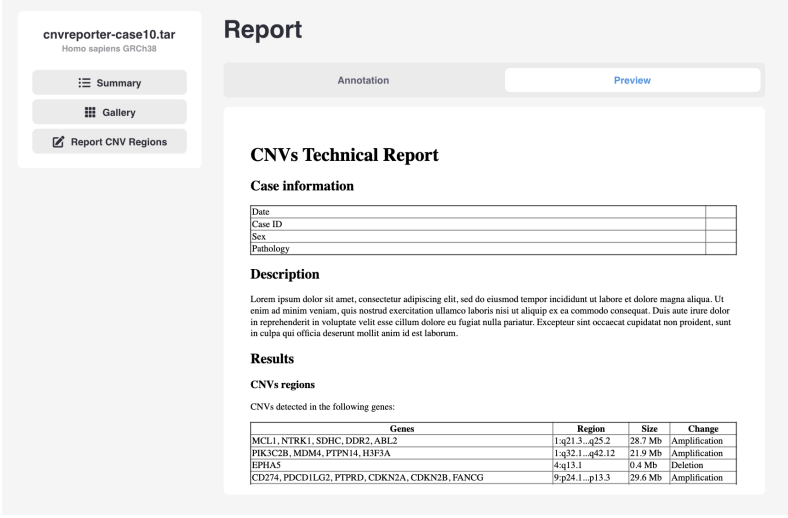


Figura 3.114. *Previsualización del informe generado, en el que puede observar una tabla en la que el usuario puede añadir información de la*

muestra estudiada, un bloque para la descripción del análisis y una tabla resumen con las regiones de CNV detectadas.

Figura 3.115. *Vista de las evidencias en el informe. Para cada región de CNV reportada, se muestra un ideograma con la zona que ocupa dicha región resaltada y la gráfica de los perfiles de cobertura para cada uno de los genes contenidos en dicha región (evidencias de la región).*

4

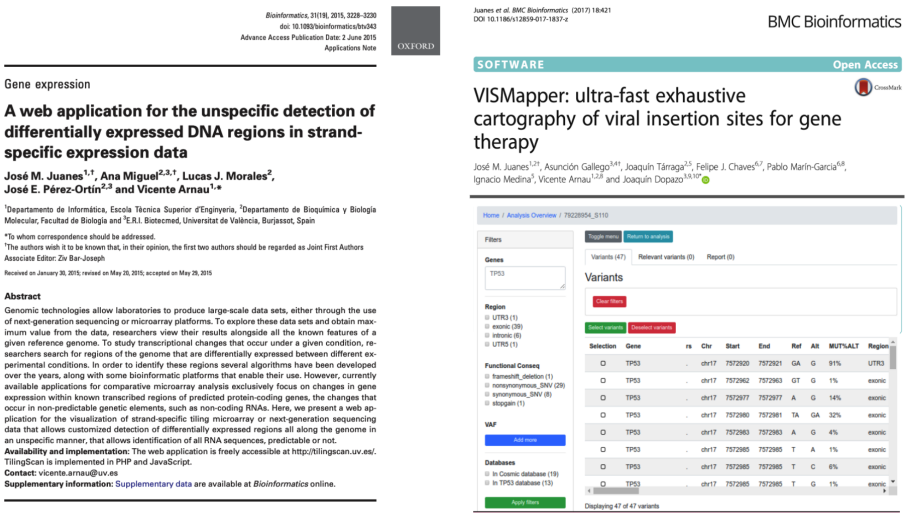
DISCUSIÓN

El desarrollo de este trabajo empieza con dos objetivos claros. Por un lado, realizar un estudio sobre el desarrollo de metodologías de visualización interactivas, con especial hincapié en su uso en el análisis exploratorio de datos. Por otro lado, modelizar y facilitar la integración de este tipo de visualizaciones en las aplicaciones bioinformáticas.

Con ello, se consigue facilitar el desarrollo de herramientas orientadas a una exploración interactiva y reactiva de datos genómicos en biomedicina. Sin embargo, esto conlleva solventar una dificultad

añadida: estas herramientas deben ser útiles en un conjunto de entornos heterogéneos, como son los laboratorios de investigación, los centros biotecnológicos y las áreas de diagnóstico en sistemas sanitarios.

En una primera fase de la investigación, se colaboró con varios laboratorios de biomedicina para resolver problemas concretos de su investigación, donde necesitaban plataformas interactivas que visualizaran los análisis. Como resultado de estas colaboraciones, se publicaron dos artículos sobre las aplicaciones creadas: **TilingScan** (Capítulo 3.1) y **VISMapper** (Capítulo 3.2). Más adelante, se desarrolló otra aplicación, llamada **SAP** (Capítulo 3.3) para la automatización del análisis de un kit diagnóstico de cáncer, que incluye un flujo de análisis bioinformático y una plataforma web para la priorización de variantes (**Figura 4.1**).



Bioinformatics, 31(19), 2015, 3228–3230
doi: 10.1093/bioinformatics/btv343
Advance Access Publication Date: 2 June 2015
Applications Note

Gene expression

A web application for the unspecific detection of differentially expressed DNA regions in strand-specific expression data

José M. Juanes[1,†], Ana Miguel[2,3,†], Lucas J. Morales[2], José E. Pérez-Ortín[2,3] and Vicente Arnau[1,*]

[1]Departamento de Informática, Escola Tècnica Superior d'Enginyeria, [2]Departamento de Bioquímica y Biología Molecular, Facultad de Biología and [3]E.R.I. Biotecmed, Universitat de València, Burjassot, Spain

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors
Associate Editor: Ziv Bar-Joseph

Received on January 30, 2015; revised on May 20, 2015; accepted on May 29, 2015

Abstract

Genomic technologies allow laboratories to produce large-scale data sets, either through the use of next-generation sequencing or microarray platforms. To explore these data sets and obtain maximum value from the data, researchers view their results alongside all the known features of a given reference genome. To study transcriptional changes that occur under a given condition, researchers search for regions of the genome that are differentially expressed between different experimental conditions. In order to identify these regions several algorithms have been developed over the years, along with some bioinformatic platforms that enable their use. However, currently available applications for comparative microarray analysis exclusively focus on changes in gene expression within known transcribed regions of predicted protein-coding genes, the changes that occur in non-predictable genetic elements, such as non-coding RNAs. Here, we present a web application for the visualization of strand-specific tiling microarray or next-generation sequencing data that allows customized detection of differentially expressed regions all along the genome in an unspecific manner, that allows identification of all RNA sequences, predictable or not.
Availability and implementation: The web application is freely accessible at http://tilingscan.uv.es/. TilingScan is implemented in PHP and JavaScript.
Contact: vicente.arnau@uv.es
Supplementary information: Supplementary data are available at *Bioinformatics* online.

Juanes et al. BMC Bioinformatics (2017) 18:421
DOI 10.1186/s12859-017-1837-z

BMC Bioinformatics

SOFTWARE Open Access

VISMapper: ultra-fast exhaustive cartography of viral insertion sites for gene therapy

José M. Juanes[1,21], Asunción Gallego[3,41], Joaquín Tárraga[2,5], Felipe J. Chaves[6,7], Pablo Marín-García[6,8], Ignacio Medina[5], Vicente Arnau[1,2,9] and Joaquín Dopazo[10,11,12*]

Figura 4.1. *Primeras tres aplicaciones desarrolladas, de las cuales dos (**VISMapper** y **TilingScan**) fueron publicadas en revistas con revisión por*

*pares, y la tercera (**SAP**) se puso en producción en una empresa de kits de análisis genético.*

Durante esta etapa, se solventaron las necesidades biológicas y de interactividad que se requerían en los laboratorios. A su vez, se detectaron las limitaciones existentes de muchas de las aplicaciones bioinformáticas al uso y las necesidades fundamentales que había que abordar para corregirlas.

La aplicación **TilingScan** estaba diseñada para la detección de expresión diferencial en experimentos de *tiling array* en levaduras. Pese a sus limitaciones, permitía la suficiente interactividad en las gráficas (ver **Figura 3.27** del Capítulo 3.1) como para ser útil, y seguía un flujo de trabajo bien definido e intuitivo (**Figura 4.2**).

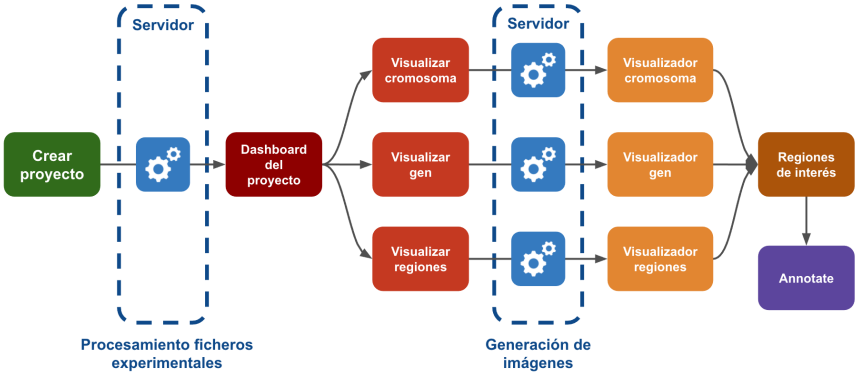


***Figura 4.2**. Flujo de uso de la aplicación **TilingScan**.*

Esta aplicación tenía una serie de puntos favorables desde un punto de vista de UX en bioinformática:

- Contenía interactividad con el movimiento del cursor e informaba del valor del punto para cada eje, permitiendo medidas precisas en las comparaciones.
- Tenía habilitado opciones de zoom por regiones. (**Figura 4.3**)
- Contenía ciertos elementos básicos de interacción con el usuario, como por ejemplo una subida de archivos a través de un formulario, un cuadro de mandos, se procesaban los datos en el servidor y que es además donde se generaban las imágenes.
- Contenía diferentes modos de visualización según el contexto que le interese al investigador (cromosoma, gen o búsqueda de regiones diferencialmente expresadas).
- Contenía un sistema de anotación integrado en el sistema para registrar las regiones de interés para el investigador.

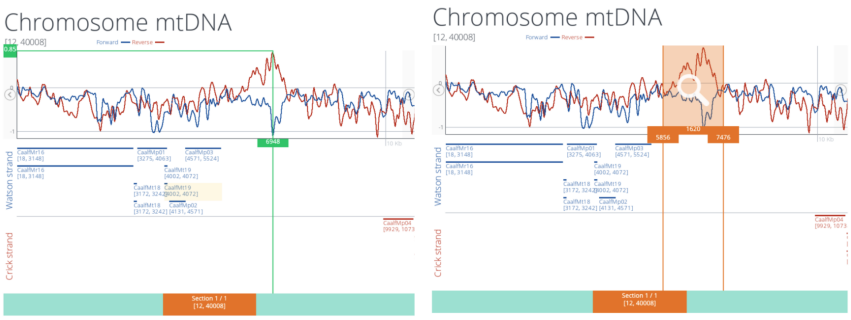


Figura 4.3. *Vista de la interactividad de* ***TilingScan****. Retroalimentación de las coordenadas del puntero y la selección de regiones para hacer vistas más detalladas.*

Aun así, la aplicación tenía ciertas limitaciones:

- La tecnología usada (2015) quedó rápidamente obsoleta. Se utilizó un servidor CGI con PHP, el cual generaba todas las imágenes que se visualizaban en la parte web. (**Figura 4.4**)
- Las imágenes generadas para la representación de la expresión de todo un cromosoma eran enormes, lo cual obligaba a dividirlas en imágenes de hasta un máximo de 5.000 píxeles.
- Los gráficos eran estáticos, se generaban en el servidor y en el cliente se permitía la navegando por cada uno de los trozos, pero como los fragmentos eran muy grandes implicaba que había que fragmentarse en varias secciones, cuando se llegaba al final de una sección se tenía que cargar la siguiente sección y volvía a empezar. Esto implicaba una merma en la calidad de la experiencia del usuario.
- La función de zoom era muy ineficiente pues había que volver a llamar al servidor para que generase de nuevo las imágenes y volverlas a traer.
- Debido a un flujo de trabajo monolítico las imágenes generadas eran específicas del tipo de visor (genómico, gen, región) y cada vez había que generar todas las imágenes, aunque los datos y la visualización fuera la misma.

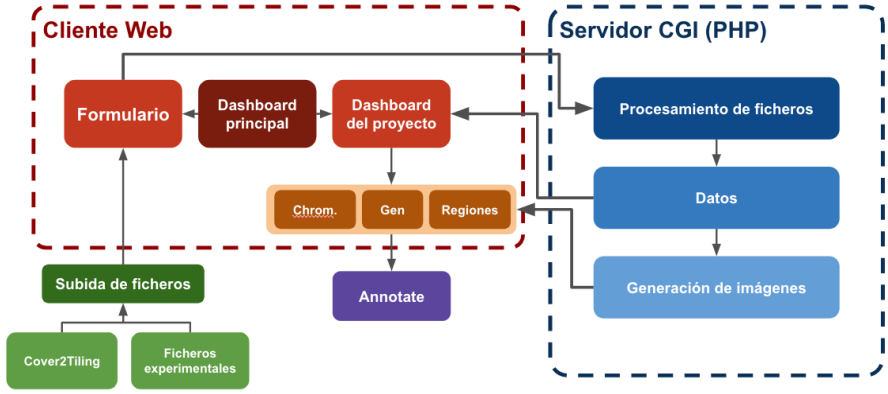


*Figura 4.4. Arquitectura de la aplicación **TilingScan.***

La siguiente solución de visualización interactiva desarrollada, **VISMapper**, fue diseñada para visualizar los lugares de inserción de virus usados en terapia génica. Las mejoras obtenidas fueron:

- La representación de las imágenes en SVG en el lado del cliente (usando una biblioteca de terceros) frente a la creación en el servidor del **TilingScan**.
- La incorporación de tablas y componentes gráficos intercomunicados mediante la utilización de eventos. Al presionar sobre una fila en la tabla se muestra la localización de la región en el ideograma.
- Incorporación de marcado de áreas interactivo en el ideograma a partir de un rango de coordenadas. (**Figura 4.5**).
- Desarrollo de una tabla con un informe automático.
- Se desarrolló un flujo de análisis bioinformático desde FASTQ a informe. El archivo FASTQ pasa por todo un flujo de procesamiento en el servidor para generar los datos de

inserción. El resultado de este procesamiento se guarda en una base de datos, facilitando el acceso posterior para su visualización desde la aplicación web (**Figura 4.6**).

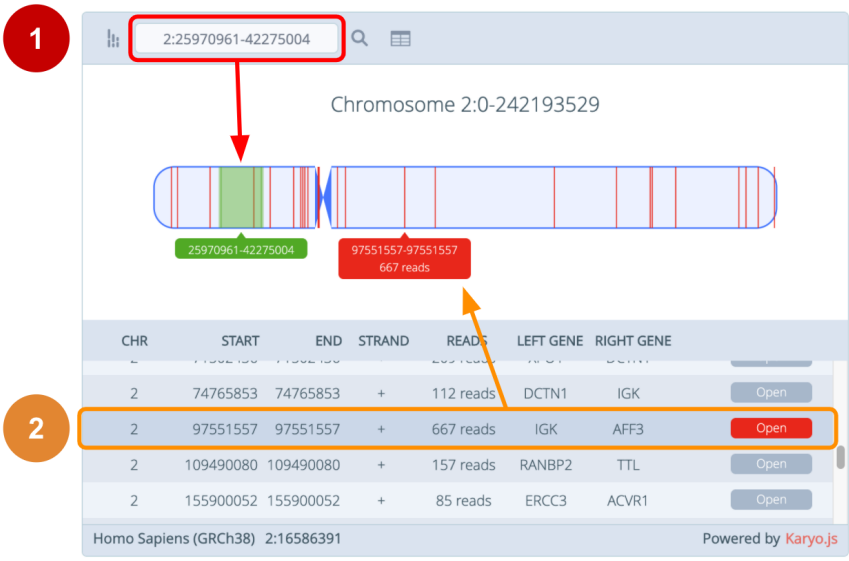


***Figura 4.5**. Ejemplo de interacción entre diferentes componentes en **VISMapper**.*

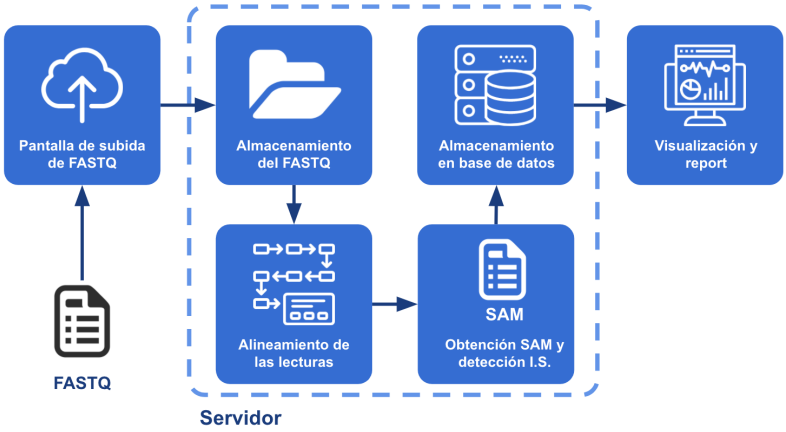


Figura 4.6. *Flujo de procesos desde que se sube el archivo hasta que se visualiza.*

Con estos dos estudios, donde se trabajaron las necesidades de visualización típicas de laboratorios de biología molecular, se sentaron las bases y se definieron los objetivos a cumplir para obtener aplicaciones realmente útiles y generadoras de conocimiento en el ámbito de la visualización bioinformática.

El siguiente paso fue la creación de una aplicación para el procesado de datos de NGS, su visualización y la realización de informes, donde se aplicó todo lo aprendido anteriormente para crear una plataforma moderna y funcional. En esta aplicación, la visualización se utilizó tanto para el control de calidad en los procesos de laboratorio como para la ayuda a la creación de informes de los análisis. Esto fue importante para cumplir el objetivo translacional de este proyecto de tesis.

La aplicación contenía un flujo completo de análisis de NGS, desde la subida de la muestra de secuenciación, pasando por su procesado en el servidor por medio de una cola de tareas y la visualización del estado de este procesado, así como del control de calidad de cada una de ellas (**Figura 4.7**).

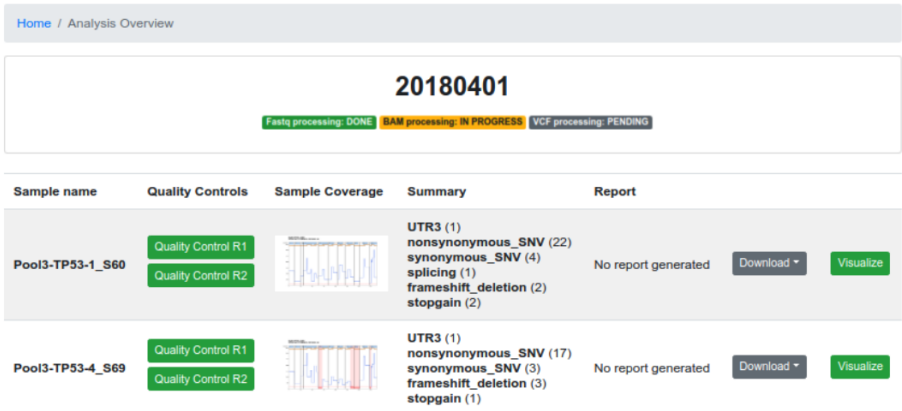


Figura 4.7. *Cuadro de mandos de muestras analizadas en la aplicación* ***SAP*** *y el estado de la cola de análisis.*

Para esta aplicación, se diseñó un filtrado y un sistema de priorización de variantes por pasos, introduciendo el concepto de «entornos de responsabilidad» en el contexto de las aplicaciones profesionales de salud, donde diferentes técnicos con diferentes niveles de responsabilidad pueden seleccionar las variantes para ser incluidas en el informe genético.

Se dio mucha importancia, en esta aplicación, a la facilidad de uso por parte del genetista clínico, y a la creación de una interfaz que esté focalizada en la creación de un informe genético donde queden bien anotadas las evidencias que dan apoyo al diagnóstico. Muchas otras

aplicaciones, como *Golden Helix* [52] , *Congenica* [53] o *SOPHiA Genetics*[54], han ido mejorando en este aspecto, pero siguen dejando de lado la importancia que hay en los laboratorios clínicos profesionales de implementar las diferentes capas y roles de responsabilidad y aprobación.

Una de las mejoras que presenta la aplicación SAP, en términos de UI, es la forma intuitiva que tiene de hacer los filtrados y agrupar los resultados de los diferentes niveles de certidumbre de las variantes. Para gestionar las selecciones de variantes, se crearon las pestañas de variantes relevantes (*Relevant variants*) y variantes para el informe (*Report variants*), que permite a los usuarios con diferentes roles centrarse en aquel conjunto de variantes de su interés (**Figura 4.8**).

[52] Software *VarSeq* de la compañía ***Golden Helix***:
https://www.goldenhelix.com/products/VarSeq/vsclinical.html

[53] Plataforma de la compañía **Congenica**:
https://www.congenica.com/solutions/our-platform/

[54] Plataforma de genómica de la compañía **Sophia Genetics**:
https://www.sophiagenetics.com/sophia-ai/genomics.html

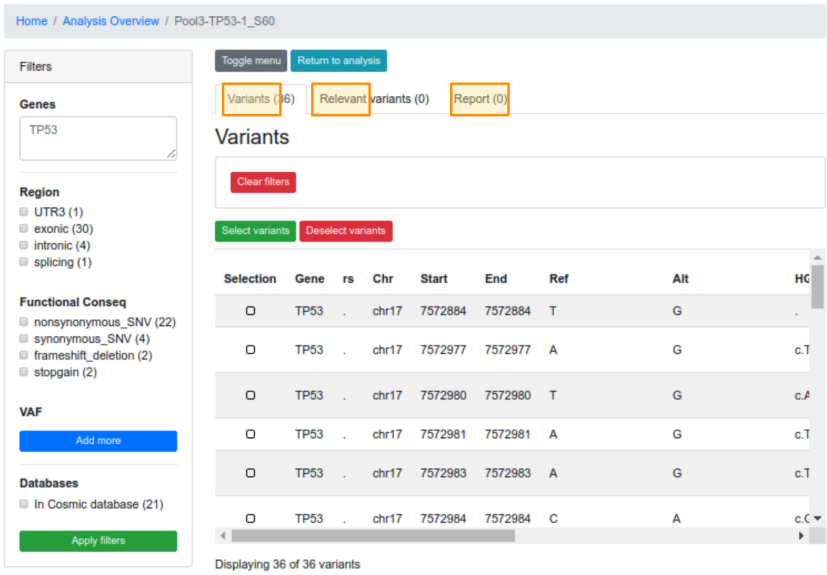


***Figura 4.8**. Vista de priorización de variantes donde se pueden ver todas las variantes filtradas y las pestañas «Relevant» y «Report».*

Otra mejora en el UI para la exploración interactiva es el rápido acceso que tiene el usuario a los datos de control de calidad y de anotación clínica de la variante (**Figura 4.9**). Entre estas pestañas de información detallada, las más destacables son la de anotación clínica (**Figura 4.9**), la de información de bases de datos específicas (**Figura 4.10**) y la de revisión de la variante (**Figura 4.11**). Esta última es la de más utilidad para la creación de una plataforma de análisis bioinformático, donde siempre debe haber un nivel de curación de datos para la mejora de análisis futuros y para permitir mantener las referencias actualizadas. Este módulo de revisión será refactorizado en las aplicaciones finales de **MGvizApps,** y es una de las piezas más importantes para el cumplimiento de los objetivos de este trabajo: la mejora de sistemas de DSS en bioinformática.

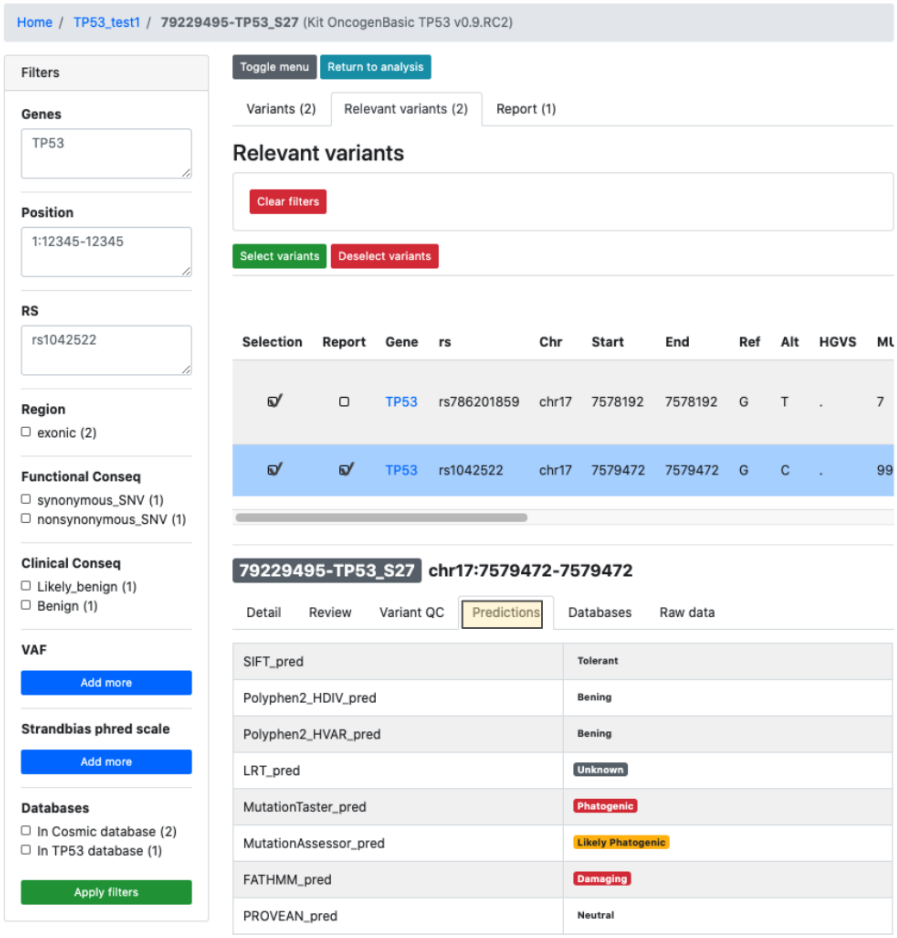


Figura 4.9. *Vista con los detalles de las anotaciones clínicas de la variante.*

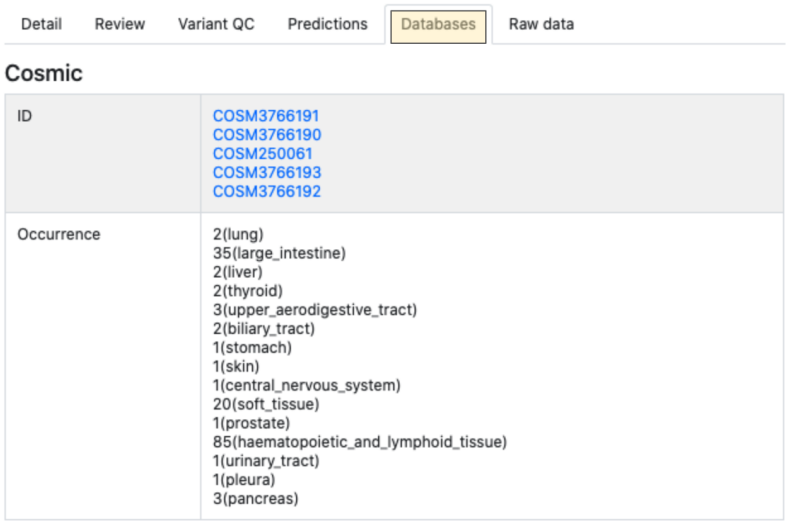


***Figura 4.10**. Detalle de la pestaña de información sobre la variante en bases de datos preseleccionadas.*

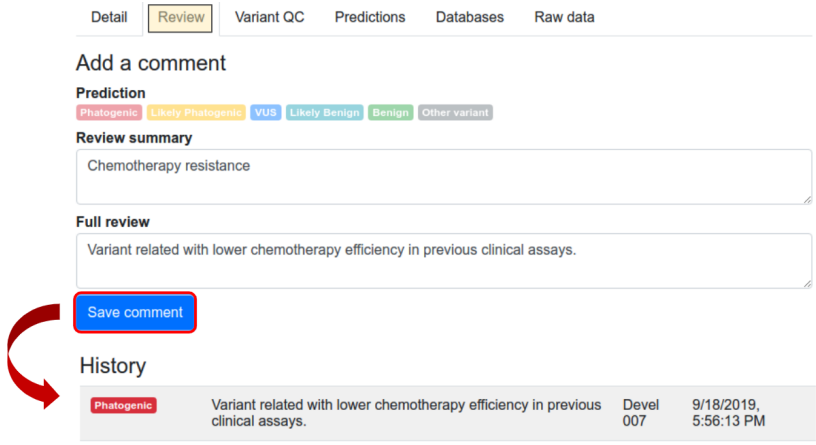


***Figura 4.11.** Vista de la pestaña de revisión de variantes donde se puede asignar la categoría clínica de la variante y añadir, para los informes, anotaciones relevantes sobre la evidencia de esta variante en el fenotipo.*

Durante el desarrollo de esta aplicación, se hicieron varios estudios sobre diferentes formas de confeccionar los informes clínicos y su automatización. Finalmente, se optó por crear una vista de informe en una pestaña resumen, donde los datos del caso se pueden rellenar automáticamente (mediante llamadas al HIS por HL7 en sistemas hospitalarios) o a mano. También se da una opción de revisar la selección y rellenar la interpretación del informe viendo los datos de las pestañas de detalle de la variante (**Figura 4.12**).

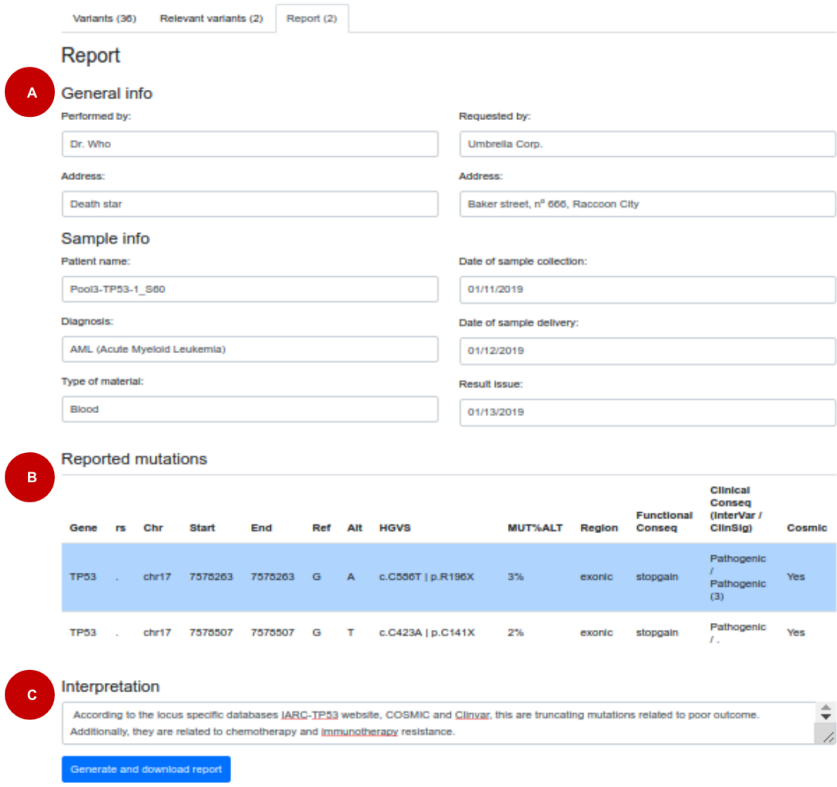


Figura 4.12. *Vista de la pantalla que permite la generación del informe, donde se puede anotar la información de la muestra y la interpretación, que posteriormente se puede descargar en formato WORD.*

Desarrollo de componentes y jviz

Con estas tres aplicaciones, se cerró una primera etapa de revisión tecnológica y de investigación sobre el desarrollo de aplicaciones interactivas, llegando a un entendimiento de las partes elementales de cualquier aplicación bioinformática que pueda dar una buena UX a partir de un UI moderno y eficaz.

A partir de aquí, se trabajó una serie de **requisitos necesarios** para poder mejorar la experiencia y la interfaz de usuario (UX y UI) de cualquier aplicación de bioinformática orientada a la exploración de datos biomédicos. Estos requisitos se detallan a continuación:

- Conseguir una interactividad más flexible, enfocada a la exploración de datos genómicos y la generación de hipótesis.
- Acceso a los datos de forma local, puesto que en muchos laboratorios existen restricciones que obligan a que datos sensibles no puedan ser sacados de la red de la institución.
- Rapidez en la construcción de los prototipos de las aplicaciones, para poder adquirir los requisitos de los investigadores.

A partir de estas necesidades, se plantearon las posibles **soluciones**, las cuales se pusieron a prueba en las nuevas aplicaciones desarrolladas. Fue necesario el planteamiento modular para el desarrollo de las aplicaciones bioinformáticas, el uso de gráficos reactivos para mejorar el análisis exploratorio de datos interactivo y por último desarrollar aplicaciones que evitaran el lado servidor. La investigación de estas soluciones se centró en los siguientes tres ejes:

- **Modularización**: diseño de un sistema de componentes y su interoperabilidad (gestión de eventos).
- **Representación gráfica**: formalización e implementación de una gramática de visualización reactiva.
- **Arquitectura** de aplicaciones biomédicas basadas en componentes modulares, y acceso local (haciendo uso del almacenamiento en el navegador y la lectura fraccionada de archivos locales).

Tras el estudio pormenorizado de la anatomía de las aplicaciones web y servicios web publicados en los últimos años[55], se vio que todas las aplicaciones seguían un esquema similar (**Figura 4.13**), formado por:

1. **Cabecera** con las diferentes acciones a realizar en la aplicación, como por ejemplo hacer inicio de sesión y subir archivos.
2. **Localizador genómico** global como un cariotipo.
3. **Tablas** donde se muestran los resultados, con una cierta interactividad y comunicación con las otras partes.
4. **Gráficos** con valores estadísticos del elemento que se está estudiando.
5. **Visualizador de regiones** concretas del genoma donde pueden verse los elementos genómicos con más detalle.

[55] Servicios web publicados en los últimos años en la revista **NAR:** https://academic.oup.com/nar/issue/47/W1.

Figura 4.13. *Estructura modular de cualquier aplicación web bioinformática.*

El resultado de aplicar este análisis a nuestra investigación fue la creación de un sistema basado en componentes reusables, que utilizan una visualización reactiva, la cual se muestra en los capítulos 3.4 (**jviz**) y 3.5 (**MGvizApps**). Estos componentes tienen como fin aislar cada uno de los diferentes elementos funcionales de la aplicación, manteniendo su independencia y proporcionando una API común. Con ello, se consigue diseñar rápidamente aplicaciones modulares mediante componentes interoperables.

El camino para conseguir esta interoperabilidad ha pasado por varias aproximaciones y desarrollos hasta llegar a su conclusión. La finalidad de este proceso ha sido la de permitir que las aplicaciones sean flexibles y puedan resolver las necesidades heterogéneas de la exploración de datos.

El resultado final del trabajo de modularización ha sido la creación de dos sistemas, **jviz-grammar** y **MGvizComponents**, cuya combinación cubre las necesidades expuestas anteriormente. Por un lado, **jviz-grammar** consiste en la formalización de un nuevo protolenguaje de visualización, centrado en la reactividad de los sistemas visuales para la exploración de datos. Por otro lado, **MGvizComponents** es una colección de componentes reusables que siguen la filosofía en la que se fundamenta todo este trabajo: la búsqueda de conocimiento a partir de una visualización versátil que permita la exploración de los datos desde diferentes perspectivas para ayudar a la generación de nuevas hipótesis.

Inicialmente, **jviz** se planteó como una librería de componentes completos y autónomos que también incluía sistemas de visualización. Esta primera versión se basaba en la tecnología de *Web Components* (utilizando la librería *Polymer*) para el desarrollo modular de las aplicaciones web. Este concepto de modularidad encajaba perfectamente con las necesidades de las aplicaciones bioinformáticas. La librería incluía tanto los componentes de UI (tablas, botones, formularios, etc..) como los componentes de visualización (ideograma, gráficos estadísticos, etc.) (**Figura 4.14**).

Aunque para análisis concretos y programados los resultados eran satisfactorios, este sistema carecía de la versatilidad y adaptación necesarias para el día a día de los laboratorios con continuos cambios en los experimentos o nuevas aproximaciones en los análisis. Para conseguir una interactividad y reactividad apropiadas en un rango mayor de escenarios (incluso los no previstos), fue necesario separar

los componentes UI (desarrollo en *React*) por un lado y la visualización (**jviz-view**) con su representación gráfica (**jviz-render**) por otro.

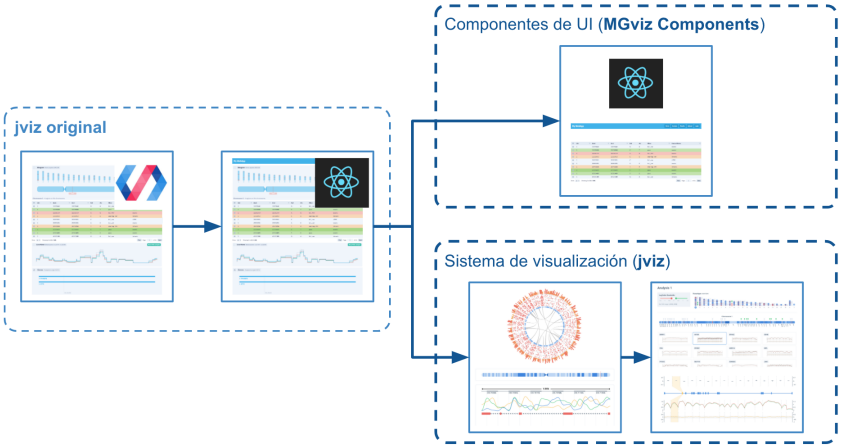


Figura 4.14. *Evolución de* ***jviz*** *desde librería de «Web Components» basada en «Polymer» hasta las dos librerías actuales: componentes de UI (MGvizComponents) y visualización (**jviz**).*

La investigación para llegar a un sistema de visualización basado en una formalización de una gramática pasó por diferentes etapas. La primera fue el uso de un componente siguiendo un modelo similar al Modelo-Vista-Controlador (MVC) (**Figura 4.15**).

Este modelo tenía la limitación de que el controlador era también un gestor de datos, y debía contener el cliente REST de los servicios consumidos por el componente. Esto producía dos inconvenientes: por un lado, creaba un «*fat controller*» acumulando la lógica de negocio en el controlador en vez de en el modelo; y, por otro, al vincular el componente con la adquisición activa de los datos, se

reducía la versatilidad del componente frente a nuevas fuentes de datos y complicaba el desarrollo.

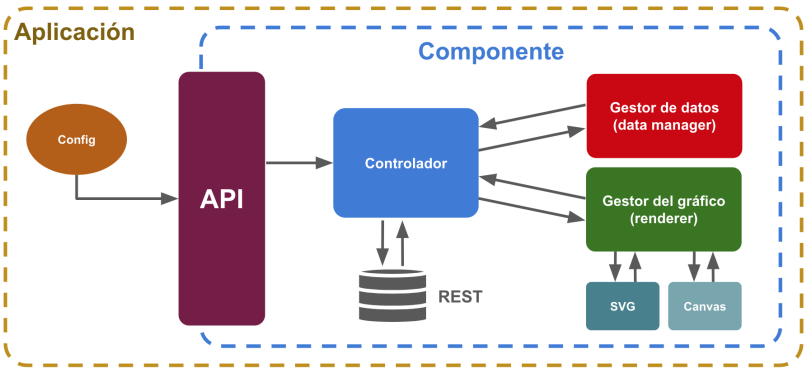


Figura 4.15. Esquema de la primera versión de los componentes, siguiendo una filosofía inspirada en el MVC. La aplicación se comunicaba con el componente a través de la API.

Para solventar estos problemas, se decidió extraer la parte de obtención de los datos fuera del componente (**Figura 4.16**). De esta forma, es la propia aplicación la encargada de la adquisición de datos, y es el componente el que recibe estos datos a través de su API. Este modelo permitía liberar al componente de la parte de la lógica de obtención y manipulación de los datos, haciendo que este fuera más agnóstico y que el control de su comportamiento ante diferentes escenarios se delegara a la aplicación.

Este modelo seguía basado en componentes con doble función (gestor de datos y visualización), lo que lo hacía eficaz para aplicaciones especializadas. Sin embargo, este modelo no escalaba en aplicaciones donde debe primar la versatilidad para adaptarse a

nuevas formas de análisis durante la exploración de los datos para la generación de hipótesis.

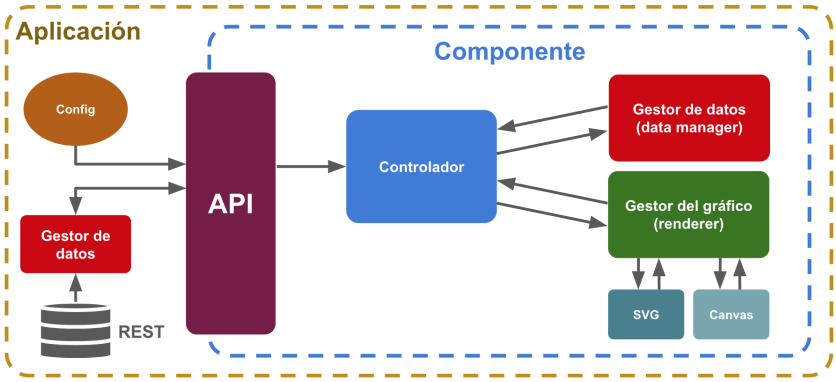


Figura 4.16. *Esquema de la segunda versión de los componentes, donde a diferencia de la primera versión las peticiones a los servicios REST son gestionados por la propia aplicación en lugar del componente.*

A partir del modelo de componente de la Figura 4.16, se generaron dos nuevos modelos complementarios. **(1)** El modelo de componente para *tracks* (donde **jviz** todavía sigue siendo la librería de componentes) (**Figura 4.17**), y **(2)** el modelo donde **jviz** se encarga solamente de la visualización y pasa a ser una parte interna de los componentes (llamados ahora **MGvizComponents**) (**Figura 4.18**).

En el modelo de *tracks*, cada una de ellas permitía realizar un tipo de representación gráfica (por ejemplo, una *track* permitía situar regiones en un cromosoma). Además, se modeló el sistema para poder tener *tracks* tanto lineales como circulares (**Figura 4.17**).

Sin embargo, este sistema tenía ciertas desventajas. En primer lugar, la parte de representación gráfica, pese a utilizar la parte de gestión

del SVG con la que contaba **jviz**, tenía un desarrollo muy específico para cada una de las *tracks*. Una vez más, esto hacía a los componentes muy eficientes para sus tareas predefinidas, pero impedía la alta versatilidad que requieren los desarrollos de aplicaciones para nuevas visualizaciones o preguntas biológicas. Por otro lado, cada nueva aplicación requería la creación o adaptación de nuevas *tracks,* que implicaba realizar muchos cambios en la parte de gestión del gráfico. Esto último reducía la usabilidad, y aumentaba tanto el tiempo de desarrollo como la complejidad de mantenimiento.

La versión actual de los componentes, **MGvizComponents** (**Figura 4.18**), está orientada a la versatilidad, y es un modelo agnóstico de la fuente de datos y su adquisición. Estos componentes solo se encargan del flujo de la información (eventos y llamadas de API) entre la aplicación y el sistema interno de representación gráfica **jviz**. En este nuevo modelo, el componente carece de lógica de negocio o de funciones extra que ahora se delegan en la aplicación, que es la que contiene las relaciones de interoperabilidad entre los componentes y el comportamiento de estos (**Figura 4.24**). Con ello, se favorece la rápida creación de aplicaciones, como puede verse en el capítulo 3.5 **MGvizApps.**

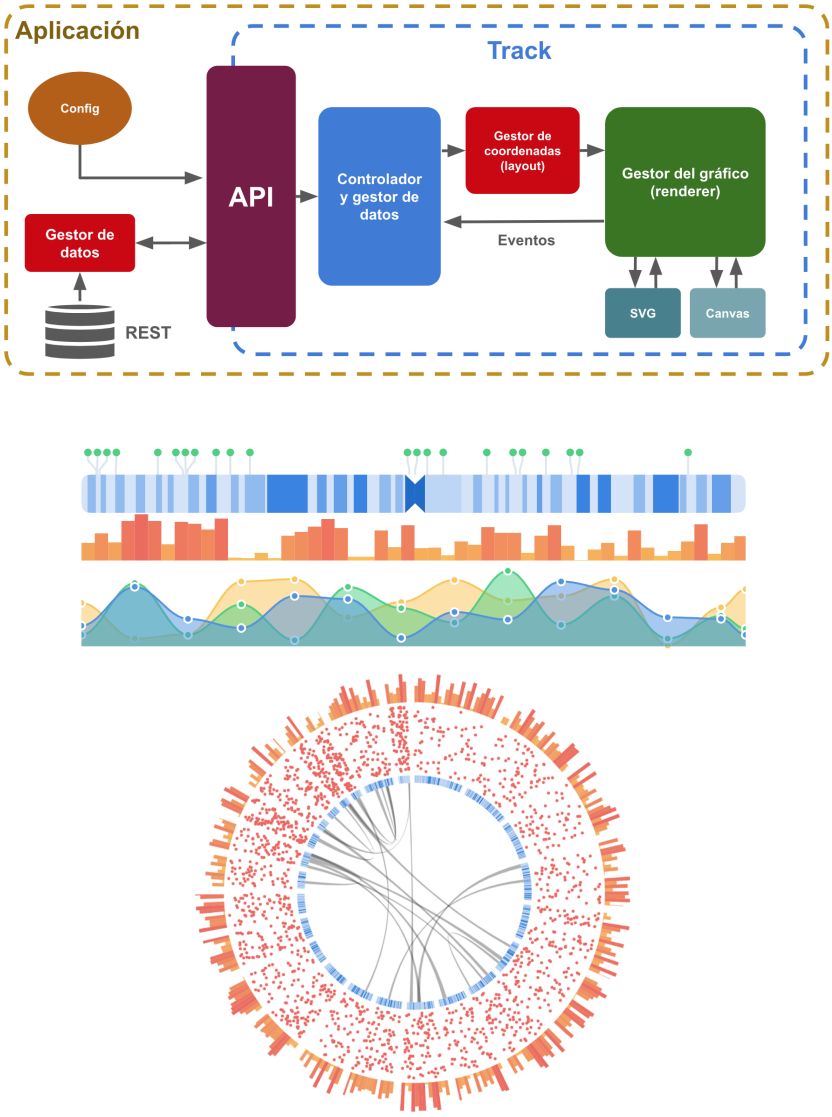


*Figura 4.17. Esquema del modelo de componentes basados en **tracks**, similar a la segunda versión de los componentes con la diferencia de que la track no lleva ninguna gestión de los datos, sino que dicha gestión es realizada por la aplicación. Este modelo permitía **tracks** lineales y circulares.*

En la versión actual del sistema, **jviz** se responsabiliza únicamente de la visualización. En lugar de proporcionar un conjunto de métodos y utilidades para crear gráficos (como se realizaba en el modelo previo) o depender de librerías de terceros (*D3.js*), se optó por definir cómo debe ser el gráfico, expresado mediante una gramática en un archivo de configuración. Para este fin, se creó un protolenguaje de visualización y un intérprete en JavaScript para su conversión a gráfico. De este modo, se liberó a los componentes de las tareas de gestión y manipulación del gráfico, las cuales quedan totalmente delegadas en **jviz** (**Figura 4.18**).

Como se ha visto en el capítulo 3.4, la gramática utilizada en **jviz** está basada en la formalización de los elementos que conforman un gráfico (que fueron descritos en el libro *The grammar of graphics* de Wilkinson, 2005). Lo importante sobre esta formalización es la adición de las capas de **fuerzas** y **expresiones**, necesarias para que se permita la descripción de gráficos no solo estáticos, sino también interactivos y reactivos (**Figura 4.19**). Además, la formalización de la gramática ha sido planteada de tal forma que pueda ser convertida en un futuro en un lenguaje específico de dominio (DSL), orientado exclusivamente a la descripción y generación de gráficos reactivos.

Con el sistema de **jviz,** se obtiene un control de los gráficos que no era posible con otros métodos hasta hace muy poco tiempo. Es cierto que *Vega* [56]tiene una interactividad parecida al sistema de **jviz,** e incluso cuenta con un mayor número de elementos implementados;

[56] Documentación de ***Vega***: https://vega.github.io/vega/

sin embargo, delega la representación y gestión de esa visualización a bibliotecas de terceros como *D3.js*[57], en lugar de tener su propio sistema de representación gráfica como hace **jviz**.

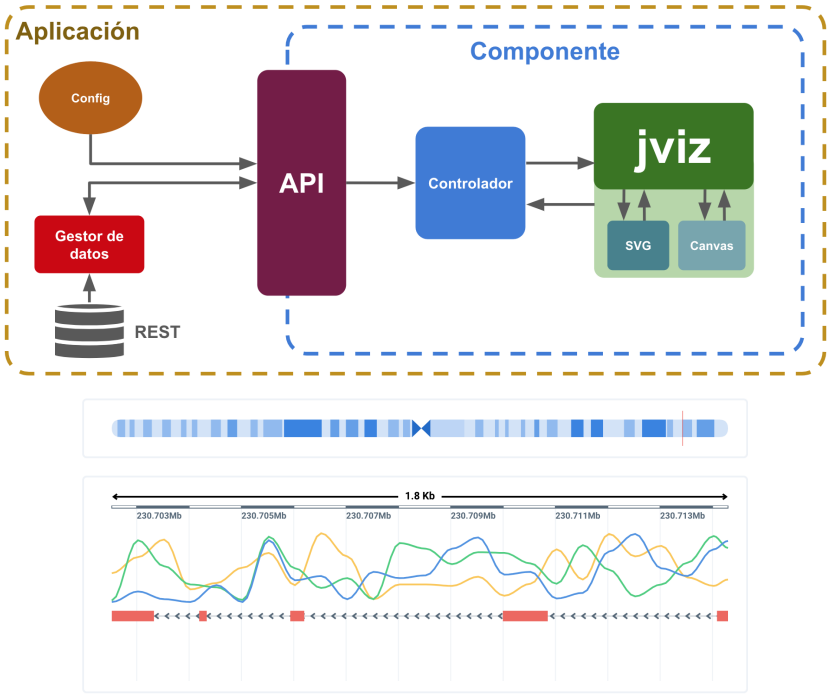


Figura 4.18. *Esquema del modelo de componentes actual (**MGvizComponents**), en el que la gestión de la representación gráfica se delega en **jviz**.*

[57] Documentación de **D3.js** (*Data Driven Documents*): https://d3js.org

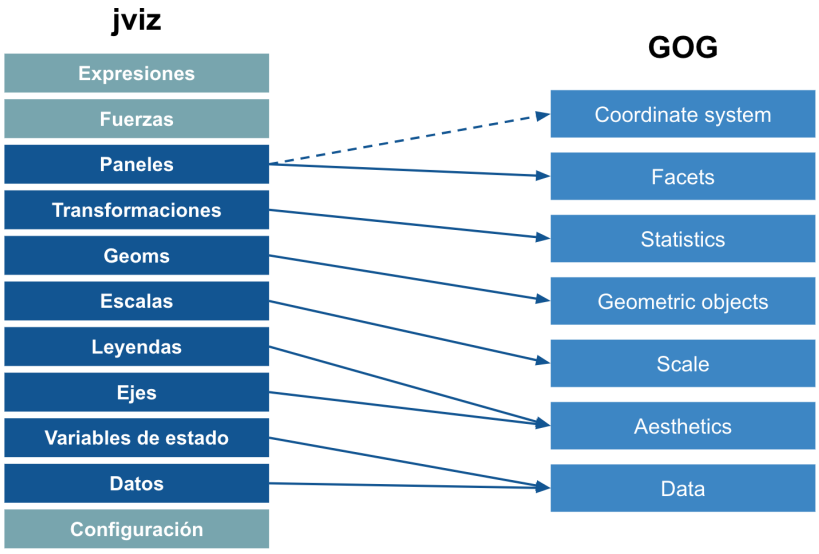


Figura 4.19. *Comparación entre los elementos de la gramática del jviz y la gramática de gráficos originalmente propuesta por Wilkinson en 2005.*

Con este sistema de componentes visuales, creados a partir de un esquema formal, no se ha pretendido hacer una biblioteca de creación de gráficas de alto nivel que sean fáciles para el usuario final, como por ejemplo *amCharts*[58] o *Google Charts*[59]. Por el contrario, se ha buscado la creación de una librería que dé el control completo a los desarrolladores para que puedan crear cualquier representación gráfica con total libertad y expresividad. Con la idea de facilitar esta

[58] Documentación de **amCharts**: https://www.amcharts.com

[59] Documentación de **Google Charts**: https://developers.google.com/chart

tarea, se ha creado **jvizlab**[60], un entorno de desarrollo integrado (IDE) que ayuda en el proceso de escritura de los esquemas para la creación de los gráficos interactivos, provee de un módulo de previsualización donde envuelve el elemento gráfico en un componente y expone la API de este componente a la aplicación **jvizlab** para poder obtener una interactividad en tiempo real (capítulo 3.4) (**Figura 4.20**).

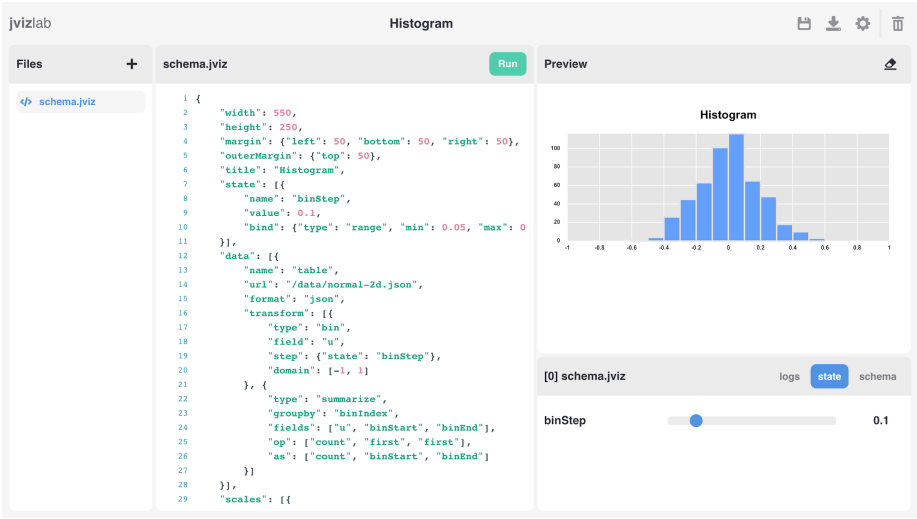


***Figura 4.20**. Editor online **jvizlab**, en el que se muestra un histograma generado a partir del esquema de la derecha. En la parte inferior se muestra un manejador para poder interactuar con el histograma.*

Llegados a este punto es importante resaltar otro de los hitos de este trabajo, que es la investigación de los modelos de reactividad de gráficos, que hace hincapié en dos aspectos importantes. Por un lado,

[60] Editor online **jvizlab**: https://viz.mgviz.org

en la arquitectura de su ciclo de vida, permitiendo la interactividad mediante la actualización de algunos de sus elementos (**Figura 4.21**). Por otro lado, en la generación de un grafo como estructura de datos para solucionar la detección de los elementos que cambian con la interacción del usuario y actualizar el resto de elementos dependientes del valor modificado, obteniendo un sistema eficiente basado en una interactividad escalable (**Figura 4.22**).

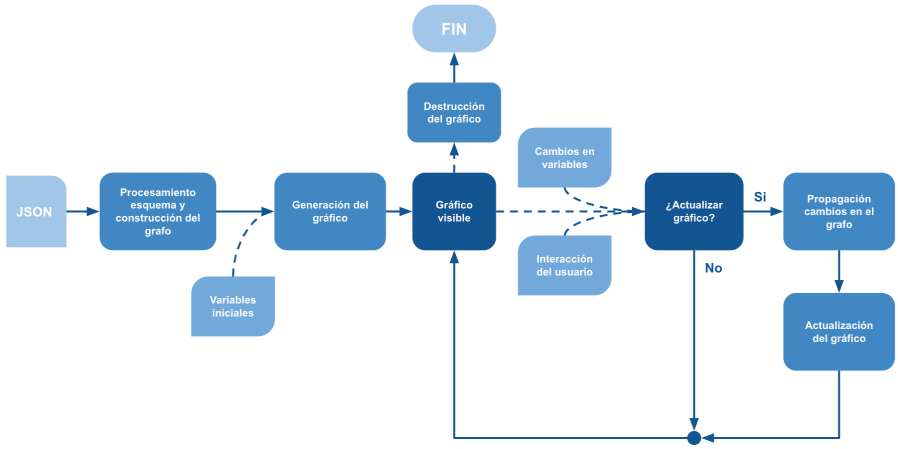


*Figura 4.21. Ciclo de vida de un gráfico de **jviz**.*

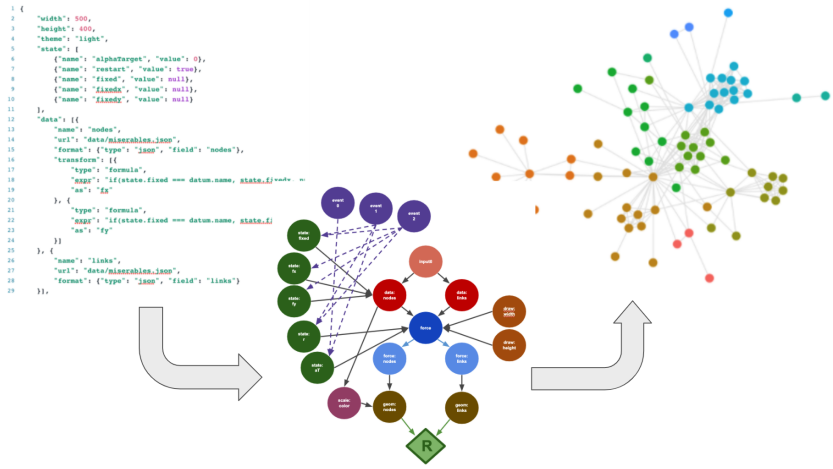


*Figura 4.22. Diagrama de construcción del gráfico de **jviz**, en el que se muestra cómo a partir de un esquema en el que se especifican todos los elementos del gráfico, este es traducido en un grafo que conecta todos estos elementos entre sí. Esto permite que, ante cualquier interacción con el gráfico generado que conlleve una modificación de algún elemento del grafo, el sistema se actualiza reactivamente.*

MGvizApps y sus componentes

Todos los resultados obtenidos a partir de la investigación realizada han permitido cumplir con los objetivos iniciales. Para demostrarlo, se ha creado una plataforma (**MGvizApps**) que integra a modo de prueba de concepto muchos de los componentes creados, sintetizados en las tres aplicaciones bioinformáticas descritas en el capítulo 3.5. (**Figura 4.23**).

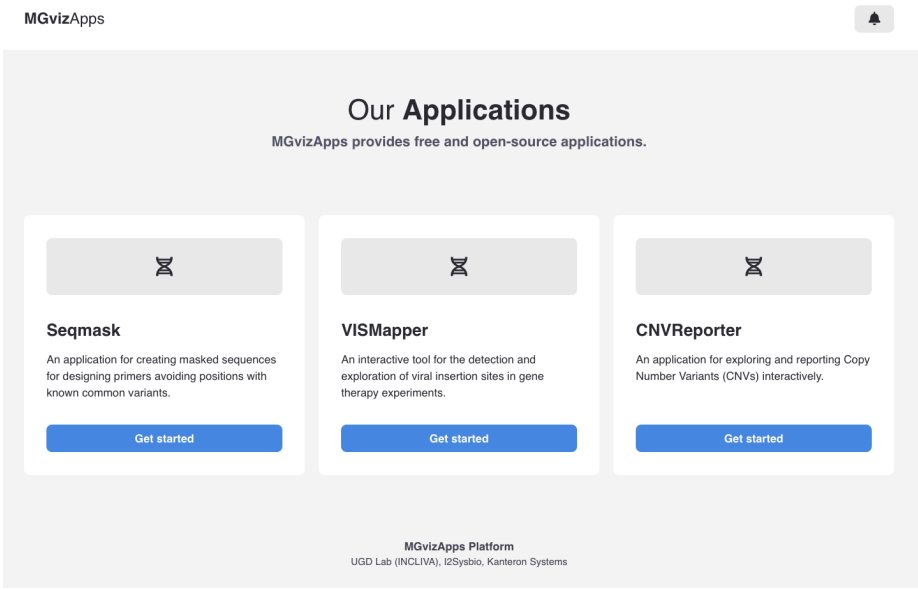


*Figura 4.23. Pantalla inicial de la plataforma **MGvizApps** con tres de las aplicaciones creadas.*

En estas aplicaciones se combinan componentes según las necesidades de análisis y de exploración de los datos, con tal de resolver la pregunta biológica planteada. La lógica de negocio, específica para las necesidades concretas a resolver en cada caso, reside en la propia aplicación que define, de esta forma, las reglas de interoperabilidad entre los componentes agrupados en el sistema. Esta delegación de la lógica en la aplicación facilita, en gran medida, la creación de componentes de visualización (sin lógica específica de los datos) que puedan ser utilizados con mayor facilidad en múltiples aplicaciones.

La comunicación de los elementos este sistema se realiza de manera siguiente. Por un lado, la aplicación se comunica con los componentes

a través de la API de cada uno, mientras que cada componente se comunica con la aplicación por medio de eventos (**Figura 4.24**). Esto confiere máxima versatilidad a las aplicaciones de exploración interactiva de datos, donde la cooperación transparente entre los componentes es fundamental.

La combinación de estos dos enfoques (la delegación de la lógica en la aplicación y la comunicación con sus componentes) hace que se reduzca notablemente el tiempo de desarrollo de este tipo de aplicaciones tan heterogéneas. Así, mientras las primeras aplicaciones mostradas en los capítulos 3.1, 3.2, .3.3, requerían meses de trabajo, las últimas que han sido desarrolladas bajo este modelo, tan solo necesitaron escasos días.

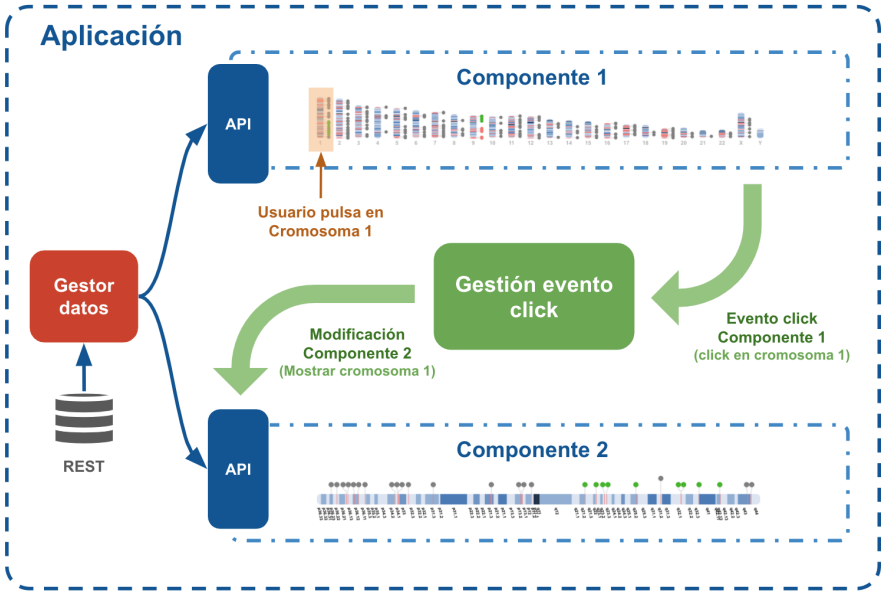


***Figura 4.24**. Interacción de la aplicación con los componentes. En el esquema, la aplicación obtiene datos a partir de un REST, que transfiere a*

los dos componentes por medio del API. Si el usuario pulsa sobre uno de los cromosomas mostrados en el componente 1, este emite un evento que es escuchado por la aplicación, la cual lo gestiona y comunica al componente 2 que muestre dicho cromosoma.

A continuación, se presentan algunos ejemplos de componentes utilizando esta arquitectura:

- **Cariotipo** (**Figura 4.25**), que está orientado a la representación y categorización de elementos genómicos (como genes, variantes, etc..) en los cromosomas de cualquier especie.
- **Estadísticas** (**Figura 4.26**), Valores resumen de propiedades genéticas agrupados por genes o regiones. Esta vista contiene propiedades interactivas para extraer más información de esos datos.
- **Ideograma** (**Figura 4.27**), al igual que el cariotipo permite representar y categorizar elementos genómicos, pero en un contexto de un único cromosoma.
- **Explorador de regiones** (**Figura 4.28**), que facilita la exploración de métricas de valores genómicos proyectado en una región (como por ejemplo niveles de expresión o perfiles de coberturas).
- **Galería de genes** (**Figura 4.29**) En este caso genes o transcritos, con sus valores de cobertura o expresión.
- **Generador de informes en WORD** (**Figura 4.30**), módulo con un área de texto para rellenar la información relevante para la interpretación de este gen o elemento genómico. A este

módulo se le pueden asociar funciones de procesado de lenguaje natural (NLP).

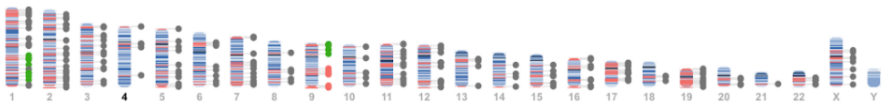


Figura 4.25. *Vista global en el cariotipo de los elementos genómicos de estudio.*

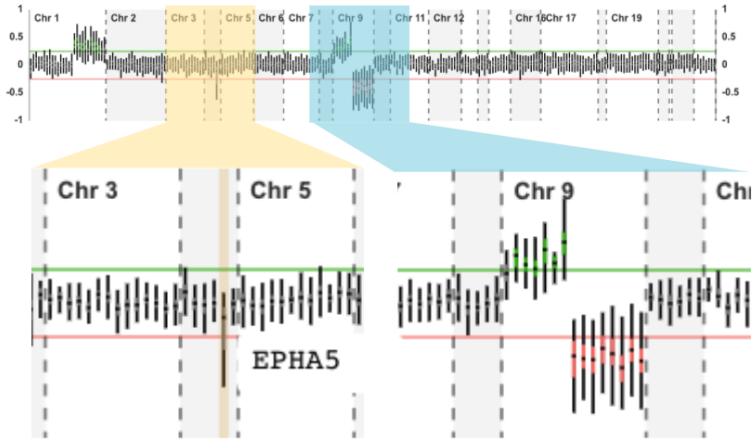


Figura 4.26. *Vista global de todas las sondas ordenadas por cromosoma mostrando un boxplot de los valores de log2ratio de cobertura tumor vs germinal de todas las bases agrupadas por gen.*

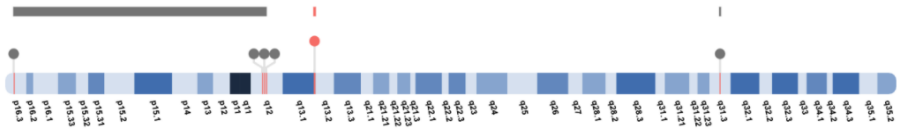


Figura 4.27. *Vista del ideograma en el que se sitúan elementos genómicos en el contexto del cromosoma.*

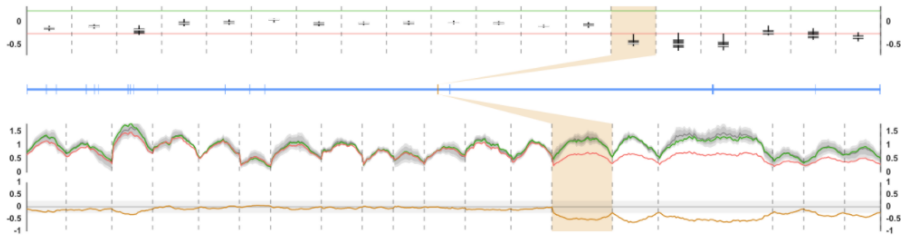


Figura 4.28. Componente de exploración de regiones.

Figura 4.29. Componente de galería de genes.

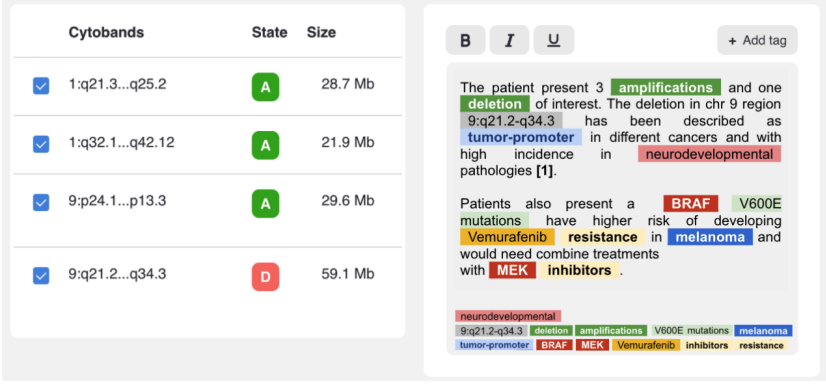


Figura 4.30. Tabla resumen de los CNVs validados y área de texto para el informe. Este área tiene etiquetado automático de términos clínicos.

Llegados a este punto es conveniente mencionar otro de los problemas que aquejan a este tipo de aplicaciones: cómo proporcionar los datos al sistema de análisis, usualmente en un sistema remoto. Un cuello de botella en los diseños de las aplicaciones genómicas es el cómo alimentar al sistema con los datos que necesita. La solución más sencilla sería habilitar un sistema de subida de ficheros al servidor desde el cliente, lo cual sólo es viable cuando los ficheros son pequeños y no muy numerosos. Cuando el número de ficheros es elevado, o su estructura de directorios es compleja, se opta por su archivado en un fichero ZIP o TAR. De esta manera, solo se tendría que manejar un fichero, pero a la larga esto se convierte en un factor limitante, sobre todo cuando el tamaño del fichero supera los 200 Mb.

Ante este problema, una solución posible es realizar un procesamiento previo de los datos más voluminosos en el ordenador local fuera de la aplicación web, para así mandar únicamente el resultado de este proceso por medio del cliente web. En versiones iniciales de las aplicaciones desarrolladas en este trabajo se exploró esta posibilidad, creando procesadores de ficheros SAM y BAM en C++ y también utilizando *Electron*[61] para darle un uso similar a una aplicación de escritorio. Sin embargo, este método tenía dos factores limitantes. En primer lugar, la necesidad de instalar el *software* que realice el procesado previo en el ordenador del usuario, lo cual no siempre podía ser factible por restricciones de permisos. Y, en segundo lugar, el hecho de que el *software* que permite realizar el procesado debe estar

[61] Documentación de **Electron**: https://www.electronjs.org

compilado y probado su funcionamiento explícitamente para cualquier sistema operativo.

Ante este problema, se buscó el dotar a la aplicación de un sistema de gestión de subida de archivos robusta. Esto se consiguió mediante el troceado, la subida parcial y la comprobación de integridad tras la reconstrucción del archivo en el servidor. Una vez puesta en marcha esta implementación, se observó que esta misma aproximación técnica se podía utilizar para acceder y procesar ficheros locales, sin necesidad de tener que subirlos a un servidor. Este modelo también podía extenderse para permitir el manejo de ficheros TAR desde el lado cliente. Esto permite el desarrollo de nuevas soluciones de análisis con solo el lado cliente, sin necesidad de implementar un REST en el servidor para el procesado y persistencia de los análisis.

Con todo lo expuesto anteriormente, se han replicado todas las funcionalidades que tenían las aplicaciones web clásicas y, además, se ha mejorado su UI, UX y privacidad.

Para terminar esta discusión se analizarán los hitos conseguidos con la aplicación de **MGvizApps**: **CNVReporter** (**Figura 4.31**).

Esta aplicación resume perfectamente toda la investigación e innovación desarrollada en este trabajo y es la que se procederá a evaluar a continuación para mostrar de forma resumida la importancia de todo el trabajo realizado.

CNVReporter es una herramienta dinámica, que permite explorar los CNVs de forma visual y refinar manualmente los resultados. Representa la parte del *front end* de un sistema de análisis

denominado **OncoMGviz**, que contiene un módulo de anotación automática de CNVs llamado **MGvizCNV** (**Figura 4.32** y **Figura 4.33**).

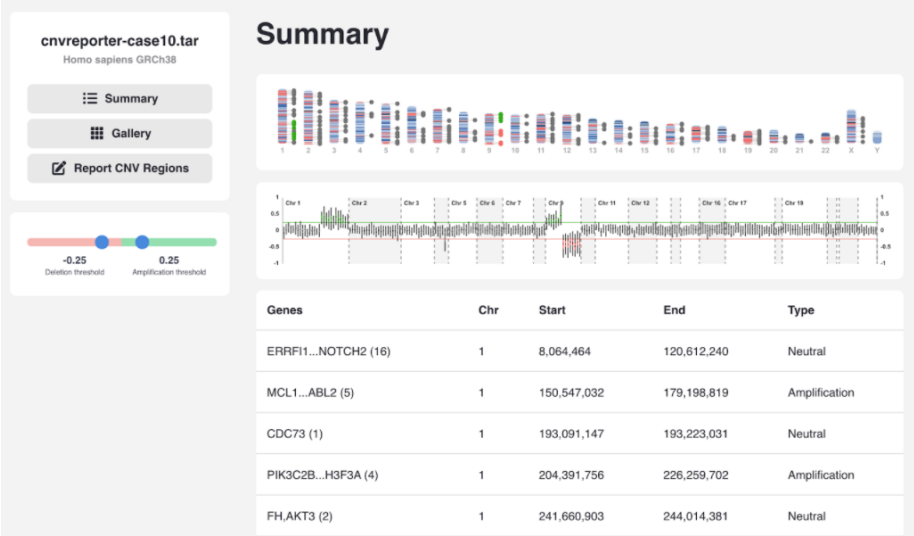


Figura 4.31. *Vista inicial de* ***CNVReporter****, donde se muestra un resumen general de los CNVs encontrados en el análisis.*

La aplicación CNVReporter está diseñada para ayudar al genetista clínico a evaluar y analizar experimentos de CNVs. La exploración visual es importante dado que la clasificación de los datos de CNV está lastrada por la alta tasa de falsos negativos y positivos, dada la naturaleza «ruidosa» de los datos en bruto.

Esta aplicación dispone de varias vistas que siguen un flujo de exploración de datos, permitiendo determinar interactiva y gráficamente el estado de las regiones cromosómicas estudiadas y crear informes a los genetistas clínicos. Para ello la aplicación trabaja mostrando coberturas normalizadas de las muestras, valores estadísticos de probabilidad de CNVs y anotación de CNVs

poblacionales, para ayudar a dilucidar el verdadero estado de la región a estudiar y determinar si está delecionado, amplificado o en estado neutro.

La detección de los CNVs en datos de NGS se basa en medir pérdidas o ganancias relativas de la profundidad de lectura en una región dada. El problema es que la mayoría de los algoritmos producen gran cantidad de CNVs, imposibles de validar en el laboratorio a gran escala. Durante esta tesis se ha colaborado en la creación de una plataforma de detección de CNVs basada en técnicas de *machine learning* cuyos resultados se pueden, posteriormente, visualizar y refinar en CNVReporter (**Figura 4.32** y **Figura 4.33**).

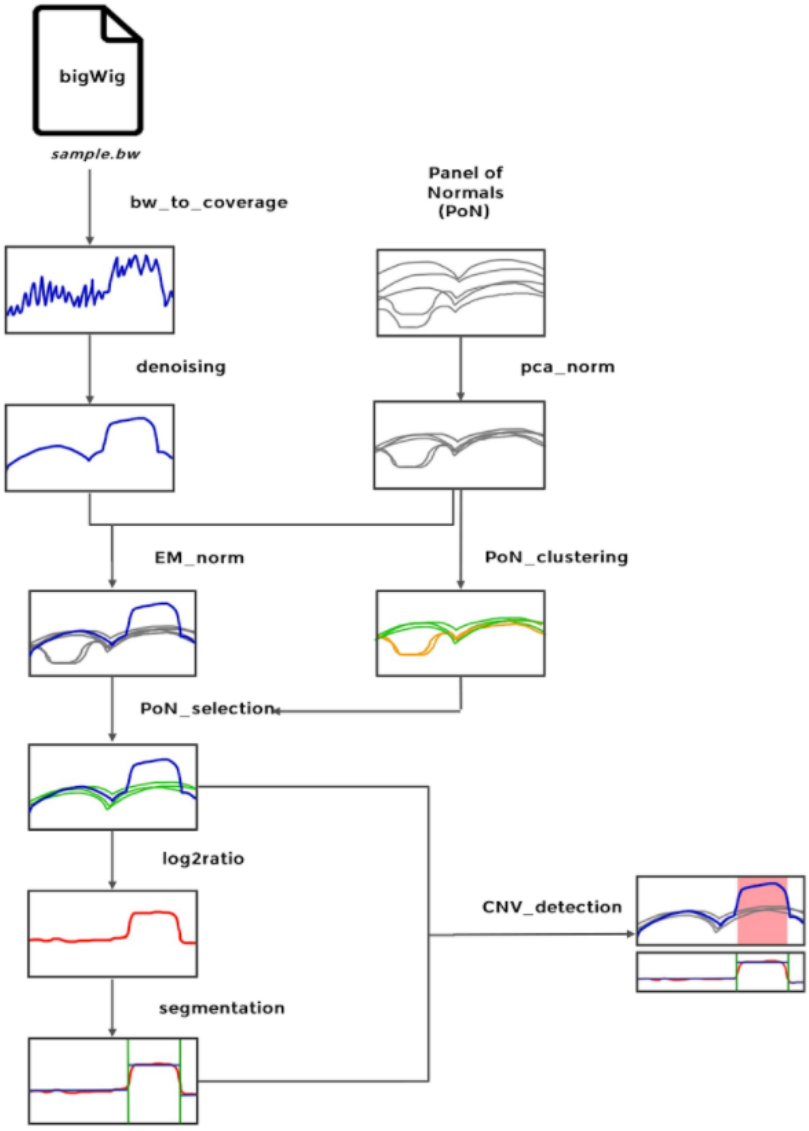


Figura 4.32. *Flujo de normalización y detección de variantes de número de copias (CNV) usando* ***CNVReporter****.*

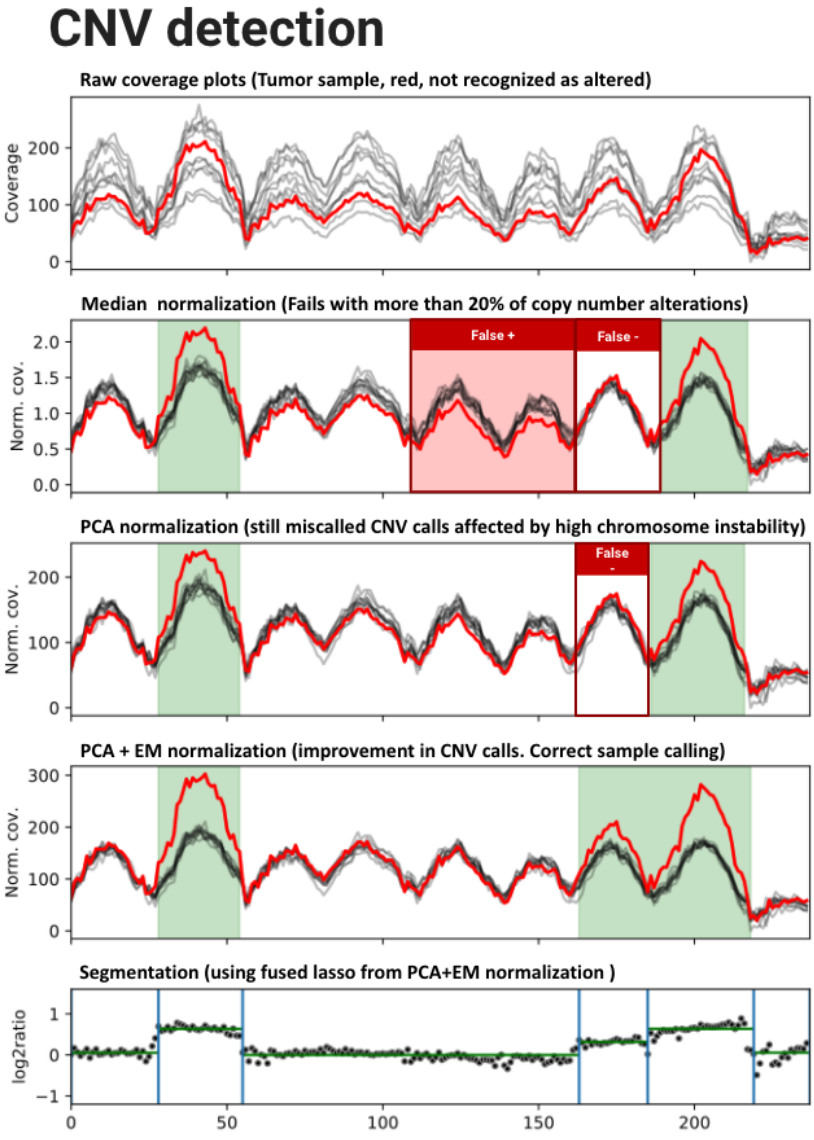


Figura 4.33. *Resultado del llamado de CNVs según diferentes métodos de normalización y detección de CNVs. Los mejores resultados se consiguieron con PCA + EM y son los que se usaron para la segmentación.*

La novedad y utilidad de CNVReporter es que se ha centrado en la exploración interactiva y estudiado la interacción del usuario con la interfaz, creando para ello unas vistas intuitivas donde se pueden evaluar los principales parámetros que ayudarán a tomar una decisión a la hora de evaluar el estado de una variante de CNV. Para ello, la aplicación provee cuatro niveles de detalle:

- **Vista global de todos los datos**, con el cariotipo (**Figura 4.25**), los *boxplots* de los *log2ratios* (**Figura 4.26**) y una tabla con listado de CNVs.
- **Vista intermedia a nivel de cromosoma** para ver el patrón y tamaño de los eventos de CNV con una vista de todos los genes implicados (**Figura 4.27** y **Figura 4.29**).
- **Vista más detallada** que muestra el nivel de evidencias (sondas-exones). Esta vista contiene también el *boxplot* resumen, con los valores log2ratio de las bases del gen (**Figura 4.28**). Y en la vista de cobertura se muestra el área de incertidumbre para una muestra normal y permite discriminar entre una muestra tumoral con CNV y una muestra ligeramente desviada de la media, pero dentro de la dispersión esperada (**Figura 4.34**).
- **Vista resumen y de generación del informe** de las regiones de CNVs anotadas (**Figura 4.35**). Esta vista contiene tanto las anotaciones de las interpretaciones añadidas en la web del informe, como las tablas de las evidencias y los gráficos de coberturas e ideogramas (**Figura 4.36** y **Figura 4.37**).

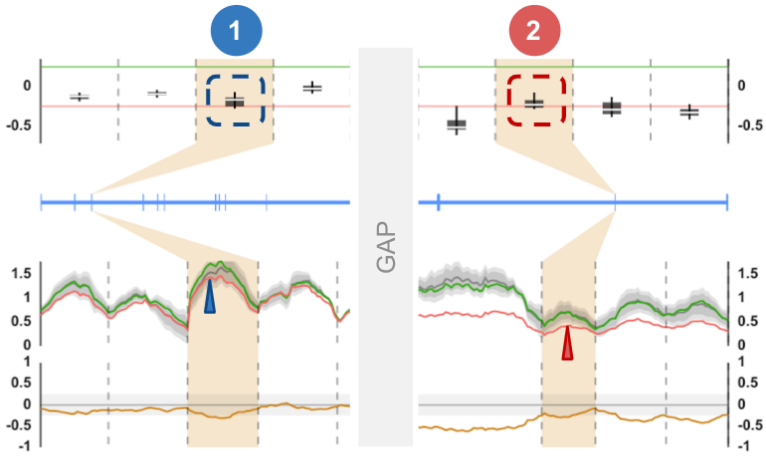


***Figura 4.34**. La vista detalle contiene también el boxplot resumen, con los valores log2ratio de las bases del gen. Y en la vista de cobertura se muestra el área de incertidumbre para una muestra normal y permite discriminar entre una muestra tumoral con CNV y una muestra ligeramente desviada de la media, pero dentro de la dispersión esperada.*

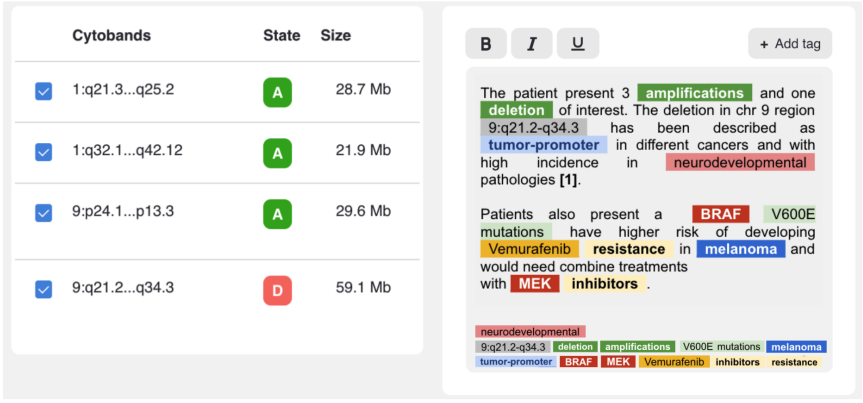


***Figura 4.35.** Tabla resumen de los CNV validadas y área de texto para el informe. Este área tiene etiquetado automático de términos clínicos.*

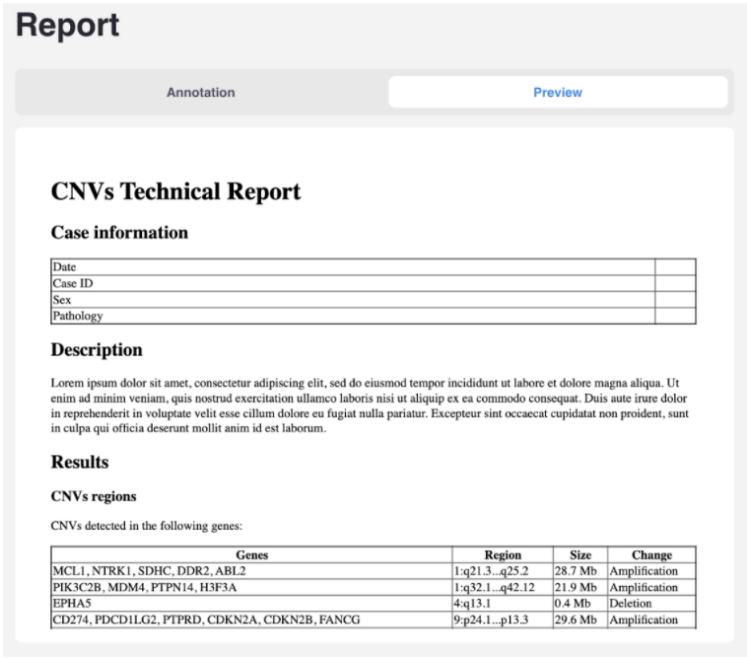


***Figura 4.36**. Exportación automática del informe a MS Word a partir de los datos seleccionados en la pantalla de resumen*

Figura 4.37. *El informe también contiene todas las evidencias gráficas que apoyan los CNV informados.*

Resumen del estudio

La hipótesis de esta tesis se centraba en el desarrollo de un sistema modular de componentes de visualización bioinformática especializado en NGS y la importancia de un sistema de gráficos agnóstico de dominio de conocimiento, que aporten una capa de reactividad a las plataformas que los incluyan y permiten la realización de análisis de datos exploratorios.

Para demostrar el éxito de esta propuesta se han construido varias plataformas con temáticas y objetivos biológicos diferentes pero que en el fondo usan muchos de los componentes modulares iguales y una misma estructura. El coste de creación de cada uno de estos

componentes se ha reducido del orden de semanas (en las primeras aplicaciones) a días (en la plataforma MGvizApps).

5

CONCLUSIONES

El trabajo presentado en esta tesis valida empíricamente la hipótesis de la importancia de la modularización y visualización reactiva en los análisis exploratorios de datos genómicos. Esto se consigue mediante el establecimiento de unos casos de éxito en la formalización e implementación de una gramática de visualización reactiva y unos modelos de arquitectura de aplicaciones biomédicas basados en componentes modulares, que han permitido la rápida creación de aplicaciones multifuncionales sin grandes costos de integración.

Se han creado varias pruebas de concepto que validan los resultados de esta investigación y que muestran que poseen una calidad y escalabilidad adecuada para ser usados como prototipos en entornos profesionales de genómica en los sistemas de salud.

1. Se han desarrollado nuevos métodos y aplicaciones bioinformáticas para la detección de CNVs, tanto en muestras germinales como en células tumorales.
2. Se ha creado un sistema de visualización para dotar a las aplicaciones de análisis genómicas de una adecuada capa de reactividad e interactividad (análisis a tiempo real).
3. Se ha creado un DSL denominado **Jviz** y su implementación mediante una biblioteca de JavaScript para su uso en aplicaciones web biomédicas.
4. Se ha creado un sistema de componentes que permita usabilidad e interoperabilidad para la creación de aplicaciones web modulares.
5. Se han creado métodos que facilitan la creación de aplicaciones bioinformáticas para la generación de informes biomédicos basados en análisis de NGS que se ejecuten en el lado cliente, pero siendo fácilmente extensibles a sistemas cliente-servidor.

Perspectivas futuras

Una vez consolidado todo este trabajo y visto la utilidad de la metodología y herramientas desarrolladas como resultado de esta investigación, los siguientes pasos se podrían resumir en:

1. Extender **jviz** a un lenguaje DSL (añadiendo elementos básicos como por ejemplo bucles o funciones), que permita sacar el máximo provecho de su flexibilidad y ayude a la creación de una versión de alto nivel de **jviz**, orientada a la creación de gráficos modelo a partir de configuraciones preestablecidas.
2. Continuar con el desarrollo de la aplicación **SeqMask** hasta su conversión en un gestor de experimentos de PCR para los laboratorios.
3. Mejorar la aplicación **CNVReporter** y añadirle la capa de predicción de CNVs, presente en el trabajo de **MGvizCNV**[62].
4. Escuchar las necesidades de los investigadores de nuestra red de colaboradores y ver las posibles herramientas que podrían desarrollarse según sus necesidades. Algunas de estas aplicaciones podrían ser:
    - Control de calidad de experimentos de PCR.
    - Priorizador de VCFs.
    - Visor y explorador de redes y rutas metabólicas.

[62] Poster del trabajo en **MGvizCNV**: https://figshare.com/articles/poster/MGvizCNV_a_QC_Machine_Learning_approach_for_CNV_evidence_scoring/12895778

- LOH (estudios de pérdida de heterozigosidad).
- Anotador y revisor de CNVs.
- Explorador de firmas mutacionales.
- Módulos de control de calidad de secuenciación de exomas.
- Cartilla de compatibilidades farmacogenómicas.

6

REFERENCIAS

Ahlberg, J. H. & Nilson, Edwin N. & Walsh, J. L. (1967). The theory of splines and their applications. New York: Academic Press.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. Journal of molecular biology, 215(3), 403-410.

Appelt, J. U., Giordano, F. A., Ecker, M., Roeder, I., Grund, N., Hotz-Wagenblatt, A., ... & Laufs, S. (2009). QuickMap: a public tool for large-scale gene therapy vector insertion site mapping and analysis. Gene therapy, 16(7), 885-893.

Arens, A., Appelt, J. U., Bartholomae, C. C., Gabriel, R., Paruzynski, A., Gustafson, D., ... & von Kalle, C. (2012). Bioinformatic clonality analysis of next-generation sequencing-derived viral vector integration sites. Human Gene Therapy, Part B: Methods, 23(2), 111-118.

Bleda, M., Tarraga, J., De María, A., Salavert, F., Garcia-Alonso, L., Celma, M., ... & Medina, I. (2012). CellBase, a comprehensive collection of RESTful web services for retrieving relevant biological information from heterogeneous sources. Nucleic acids research, 40(W1), W609-W614.

Brent, M. R. (2007). How does eukaryotic gene prediction work?. Nature biotechnology, 25(8), 883-885.

Bahrami-Samani, E., Vo, D. T., de Araujo, P. R., Vogel, C., Smith, A. D., Penalva, L. O., & Uren, P. J. (2015). Computational challenges, tools, and resources for analyzing co-and post-transcriptional events in high throughput. Wiley Interdisciplinary Reviews: RNA, 6(3), 291-310.

Basseville, M., & Nikiforov, I. V. (1993). Detection of abrupt changes: theory and application (Vol. 104). Englewood Cliffs: prentice Hall.

Cartier, N., Hacein-Bey-Abina, S., Bartholomae, C. C., Veres, G., Schmidt, M., Kutschera, I., ... & Mahlaoui, N. (2009). Hematopoietic stem cell gene therapy with a lentiviral vector in X-linked adrenoleukodystrophy. science, 326(5954), 818-823.

Cartwright, R. A., & Graur, D. (2011). The multiple personalities of Watson and Crick strands. Biology direct, 6(1), 7.

Catmull, E., and Rom, R. (1974). A class of local interpolating splines. Computer Aided Geometric Design, R. E. Barnhill and R. F. Reisenfeld, Eds. Academic Press, New York, pp. 317–326.

Cavazzana-Calvo, M., Payen, E., Negre, O., Wang, G., Hehir, K., Fusil, F., ... & Cavallesco, R. (2010). Transfusion independence and HMGA2 activation after gene therapy of human β-thalassaemia. Nature, 467(7313), 318-322.

Cherry, J. M., Adler, C., Ball, C., Chervitz, S. A., Dwight, S. S., Hester, E. T., Jia, Y., Juvik, G., Roe, T., Schroeder, M., Weng, S., & Botstein, D. (1998). SGD: Saccharomyces genome database. Nucleic acids research, 26(1), 73-79.

Depledge, D. P., Mohr, I., & Wilson, A. C. (2018). Going the Distance: Optimizing RNA-Seq Strategies for Transcriptomic Analysis of Complex Viral Genomes. Journal of virology, 93(1), e01342-18.

Finkel, R.A., Bentley, J.L. Quad trees a data structure for retrieval on composite keys. Acta Informatica 4, 1–9 (1974).

Forbes, S. A., Bindal, N., Bamford, S., Cole, C., Kok, C. Y., Beare, D., ... & Teague, J. W. (2010). COSMIC: mining complete cancer genomes in the Catalogue of Somatic Mutations in Cancer. Nucleic acids research, 39(suppl_1), D945-D950.

Gaspar, H. B., Parsley, K. L., Howe, S., King, D., Gilmour, K. C., Sinclair, J., ... & Jakobsen, M. A. (2004). Gene therapy of X-linked severe combined immunodeficiency by use of a pseudotyped gammaretroviral vector. The Lancet, 364(9452), 2181-2187.

González, R.C. and Woods, R.E. (2008). Digital Image Processing, 3rd Ed., Pearson Prentice Hall, Pearson Education, Inc., New Jersey, USA.

Haddad, R.A. & Akansu, A.N. (1991), A Class of Fast Gaussian Binomial Filters for Speech and Image Processing, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 39, 723-727.

Hocum, J. D., Battrell, L. R., Maynard, R., Adair, J. E., Beard, B. C., Rawlings, D. J., ... & Trobridge, G. D. (2015). VISA-Vector Integration Site Analysis server: a web-based server to rapidly identify retroviral integration sites from next-generation sequencing. BMC bioinformatics, 16(1), 1-5.

Ishak, C. A., & De Carvalho, D. D. (2020). Reactivation of Endogenous Retroelements in Cancer Development and Therapy. Annual Review of Cancer Biology, 4, 159-176.

Jacoby, M. A., Duncavage, E. J., & Walter, M. J. (2015). Implications of Tumor Clonal Heterogeneity in the Era of Next-Generation Sequencing. Trends in cancer, 1(4), 231–241.

Jiang, H., & Wong, W. H. (2008). SeqMap: mapping massive amount of oligonucleotides to the genome. Bioinformatics, 24(20), 2395-2396.

Juanes, J. M., Miguel, A., Morales, L. J., Pérez-Ortín, J. E., & Arnau, V. (2015). A web application for the unspecific detection of differentially expressed DNA regions in strand-specific expression data. Bioinformatics (Oxford, England), 31(19), 3228–3230.

Juanes, J. M., Gallego, A., Tárraga, J., Chaves, F. J., Marín-Garcia, P., Medina, I., ... & Dopazo, J. (2017). VISMapper: ultra-fast exhaustive cartography of viral insertion sites for gene therapy. BMC bioinformatics, 18(1), 1-5.

Kent, W. J. (2002). BLAT—the BLAST-like alignment tool. Genome research, 12(4), 656-664.

Klevebring, D., Bjursell, M., Emanuelsson, O., & Lundeberg, J. (2010). In-depth transcriptome analysis reveals novel TARs and prevalent antisense transcription in human cell lines. PloS one, 5(3), e9762.

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. bioinformatics, 25(14), 1754-1760.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... & Durbin, R. (2009). The sequence alignment/map format and SAMtools. Bioinformatics, 25(16), 2078-2079.

Machado-Lima, A., Del Portillo, H. A., & Durham, A. M. (2008). Computational methods in noncoding RNA research. Journal of mathematical biology, 56(1-2), 15-49.

Mattick, J.S. and Makunin, I.V. (2006) Non-coding RNA, Hum. Mol. Genet. 15, R17-29.

Medina I, Salavert F, Sanchez R, de Maria A, Alonso R, Escobar P, Bleda M, Dopazo J. Genome Maps, a new generation genome browser.

Nucleic Acids Res. 2013 Jul;41(Web Server issue):W41-6. doi: 10.1093/nar/gkt530. Epub 2013 Jun 8. PMID: 23748955; PMCID: PMC3692043.

Mitchell, R. S., Beitzel, B. F., Schroder, A. R., Shinn, P., Chen, H., Berry, C. C., ... & Bushman, F. D. (2004). Retroviral DNA integration: ASLV, HIV, and MLV show distinct target site preferences. PLoS Biol, 2(8), e234.

Molina-Navarro, M. M., Castells-Roca, L., Bellí, G., García-Martínez, J., Marín-Navarro, J., Moreno, J., Pérez-Ortín, J. E., & Herrero, E. (2008). Comprehensive transcriptional analysis of the oxidative response in yeast. Journal of Biological Chemistry, 283(26), 17908-17918.

Nusrat, S., Harbig, T., & Gehlenborg, N. (2019, June). Tasks, techniques, and tools for genomic data visualization. In Computer Graphics Forum (Vol. 38, No. 3, pp. 781-805).

Pérez-Ortín, J. E., de Miguel-Jiménez, L., & Chávez, S. (2012). Genome-wide studies of mRNA synthesis and degradation in eukaryotes. Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms, 1819(6), 604-615.

Poppendieck, M., T. Poppendieck (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley Professional. ISBN 978-0-321-15078-3.

Paruzynski, A., Arens, A., Gabriel, R., Bartholomae, C. C., Scholz, S., Wang, W., ... & Von Kalle, C. (2010). Genome-wide high-throughput integrome analyses by nrLAM-PCR and next-generation sequencing. Nature protocols, 5(8), 1379.

Slater GS, Birney E. Automated generation of heuristics for biological sequence comparison. BMC Bioinformatics. 2005 Feb 15;6:31. doi: 10.1186/1471-2105-6-31. PMID: 15713233; PMCID: PMC553969.

Suárez, E., Burguete, A., & McLachlan, G. J. (2009). Microarray data analysis for differential expression: a tutorial. Puerto Rico health sciences journal, 28(2).

Tárraga, J., Arnau, V., Martínez, H., Moreno, R., Cazorla, D., Salavert-Torres, J., ... & Medina, I. (2014). Acceleration of short and long DNA read mapping without loss of accuracy using suffix array. Bioinformatics, 30(23), 3396-3398.

Wickham H (2010). A layered grammar of graphics. Journal of Computational and Graphical Statistics (Vol. 19, No. 1, pp 3-28).

Wickham H (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York. ISBN 978-3-319-24277-4.

Wilkinson, L. (2005), The Grammar of Graphics (2nd ed.). Statistics and Computing, New York: Springer.

Wu, X., Li, Y., Crise, B., & Burgess, S. M. (2003). Transcription start regions in the human genome are favored targets for MLV integration. Science, 300(5626), 1749-1751.

Yates, A. D., Achuthan, P., Akanni, W., Allen, J., Allen, J., Alvarez-Jarreta, J., Amode, M. R., Armean, I. M., Azov, A. G., Bennett, R., Bhai, J., Billis, K., Boddu, S., Marugán, J. C., Cummins, C., Davidson, C., Dodiya, K., Fatima, R., Gall, A., Giron, C. G., … Flicek, P. (2020). Ensembl 2020. Nucleic acids research, 48(D1), D682–D688.

Yazaki, J., Gregory, B. D., & Ecker, J. R. (2007). Mapping the genome landscape using tiling array technology. Current opinion in plant biology, 10(5), 534–542.

7

APÉNDICE

A web application for the unspecific detection ofdifferentially expressed DNA regions in strand-specific expression data

Bioinformatics, 31(19), 2015, 3228–3230
doi: 10.1093/bioinformatics/btv343
Advance Access Publication Date: 2 June 2015
Applications Note

Gene expression

A web application for the unspecific detection of differentially expressed DNA regions in strand-specific expression data

José M. Juanes[1,†], Ana Miguel[2,3,†], Lucas J. Morales[2], José E. Pérez-Ortín[2,3] and Vicente Arnau[1,*]

[1]Departamento de Informática, Escola Tècnica Superior d'Enginyeria, [2]Departamento de Bioquímica y Biología Molecular, Facultad de Biología and [3]E.R.I. Biotecmed, Universitat de València, Burjassot, Spain

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors
Associate Editor: Ziv Bar-Joseph

Received on January 30, 2015; revised on May 20, 2015; accepted on May 29, 2015

Abstract

Genomic technologies allow laboratories to produce large-scale data sets, either through the use of next-generation sequencing or microarray platforms. To explore these data sets and obtain maximum value from the data, researchers view their results alongside all the known features of a given reference genome. To study transcriptional changes that occur under a given condition, researchers search for regions of the genome that are differentially expressed between different experimental conditions. In order to identify these regions several algorithms have been developed over the years, along with some bioinformatic platforms that enable their use. However, currently available applications for comparative microarray analysis exclusively focus on changes in gene expression within known transcribed regions of predicted protein-coding genes, the changes that occur in non-predictable genetic elements, such as non-coding RNAs. Here, we present a web application for the visualization of strand-specific tiling microarray or next-generation sequencing data that allows customized detection of differentially expressed regions all along the genome in an unspecific manner, that allows identification of all RNA sequences, predictable or not.

Availability and implementation: The web application is freely accessible at http://tilingscan.uv.es/. TilingScan is implemented in PHP and JavaScript.

Contact: vicente.arnau@uv.es

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Eukaryotic genomes are composed by coding and non-coding DNA sequences. Until recently it was thought that most coding regions had been identified, as they encode either protein-coding genes, highly conserved ribosomal RNA, transference RNA or other minor non-coding RNA (ncRNA) species which open reading frames (ORF) are relatively easy to find (Brent, 2007). Only 5′ and 3′ extended transcribed regions outside the ORF and some introns were not easily predictable (Bahrami-Samani *et al.*, 2014; Machado-Lima *et al.*, 2008). During the last years, however, evidence from different sources has made clear that most of the assumed 'non-coding' DNA was, in fact, encoding non-predictable RNA molecules. These non-coding sequences have proven not only to be expressed, but also to play many different regulatory roles within the cell (Mattick and Makunin, 2006).

Current genome-wide methods, such as tiling microarrays and next generation sequencing (NGS) provide platforms for the study of transcriptomes. In most cases, transcriptomic analyses are performed to compare relative RNA abundance in different samples, such as wild type versus mutant or before versus after stress

induction (Molina-Navarro *et al.*, 2008; see Pérez-Ortín *et al.*, 2012 for a review). Most current algorithms and bioinformatic applications for comparative analysis in tiling arrays or NGS exclusively search for expression changes within known transcribed regions of predicted genetic elements (see Suarez *et al.*, 2009 and Bahrami-Samani *et al.*, 2014, for reviews). Changes in transcript length, existence of unpredicted introns, antisense transcripts, or other unannotated ncRNAs are not detectable using these algorithms. To meet this need, we have developed TilingScan, a new web application for the visualization of microarray data that has been implemented with a new search algorithm for the detection of differentially expressed regions in an unsupervised manner. For this detection, we have developed a version of the geometric moving average algorithm, which corresponds to the class of methods of stationary random processes and constitutes one of the several methods developed for the detection of abrupt changes in a signal (Basseville and Nikiforof, 1993). This algorithm scans the signal based in scalable and sliding windows and searches for over- or under-expressed regions of a minimum size of nucleotides flanked by neutral regions (see supplementary materia for detailed description). Rather than focusing on changes in protein-coding genes exclusively, the application detects changes that occur all along the genome, including unannotated ncRNA, and then assigns them to the specific genetic elements annotated, if any. In addition, provided that microarray data signals often display noisy profiles, TilingScan enables the application of a smoothing algorithm that is based on a Gaussian filter, allowing signal noise removal prior to the search and clearer visualization of the data. The application can be used to analyze NGS data provided that BAM files are converted into compatible files using freely available software as described in the tutorial.

2 Implementation of the website

The website is implemented in PHP and JavaScript. PHP was used for the detection of significant differentially expressed regions and JavaScript was used for interactive visualization of the regions. Depending on the nature of the data set, strand specific or non-strand specific, visualization can be selected. By uploading data sets to the server each user can create a new project that will be saved under a unique identifier provided by the website (Project ID) as well as under a personal identifier selected by the user. This ID allows easy access to every created project, allowing different research groups to collaborate by sharing the specific project ID. The website contains a tutorial and a set of demonstration data for the trial of the application prior to any use. Along with the visualization and the detection of regions, the website has been implemented with a smoothing algorithm for signal noise removal (*Gauss filter*), as well as with a tool for manual annotation of regions of interest (*Annotate!*).

3 Structure and usage of the website

3.1 Visualization tool

The application enables the upload of data from tiling array (or data from NGS converted to BAM files) results as genome graphs, which display probe intensity alongside the genomic sequence, aligning it to a reference genome provided by the user. This allows customized visualization of either specific chromosomes or specific genes of interest using the *Visualize chromosome* and *Visualize gene* tools (Fig. 1A). If signal smoothing is required, the application of the Gaussian filter can be manually selected prior to visualization, and

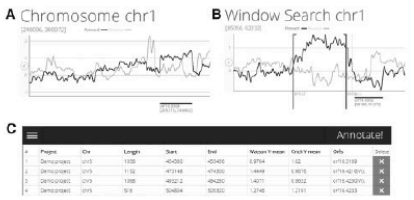


Fig. 1. TilingScan graphical outputs. (**A**) By selecting the chromosome or the gene of interest, signal intensity profiles will be shown, displaying the intensity profile on forward (Watson) strand in black and the reverse (Crick) strand in grey. Start and end chromosomal coordinates of the visualized region are shown in brackets (top, left). (**B**) Graphical output of a region detected using the demo data. Start and end chromosomal coordinates of the detected region are shown on the edges of the dark grey vertical lines respectively. (**C**) Image section of the Annotate! tool displaying recorded features for some of the regions detected in a transcriptomic study performed in yeast under hypoxia

permuted the desired number of times. If proper visualization of a specific region is required, a region of choice can be delimited and zoomed for X and Y scale adjustment. By delimiting a region manually and selecting the *Annotate* option, specific features of the selected region will be registered (such as chromosome, chromosomal coordinates, length and mean intensity value on each strand) and listed in a table, which can be edited and downloaded for further use (Fig. 1C).

3.2 Analysis tool for the detection of differentially expressed regions

For the detection of differential expression, the application has been implemented with a search algorithm that is based on a sliding window model (See Supplementary Material for description). Using the search tool (*Window search*) the application will accurately locate and identify differentially expressed regions, displaying both graphical outputs (Fig. 1B) and the register of the above-mentioned region features (chromosomal coordinates, length, etc.). If a given region is detected at specific genomic coordinates that correspond to previously annotated genetic elements, the ORF names found within the region will also be recorded. These regions will not only be annotated automatically after the user runs the search analysis, but can also be selected and annotated manually if other regions of interest are found via the *Annotate!* tool.

Funding

J.E.P-O.is supported by grants from the Spanish MCINN (BFU2013-48643-C3-3-P), and from the Regional Valencian Government (PROMETEO 2011/088 and PROMETEOII 2015/006). The authors are thankful to Tianlu Li for critical revision of the manuscript.

Conflict of Interest: none declared.

References

Brent,M.R. (2007) How does eukaryotic gene prediction work? *Nat. Biotechnol.*, 25, 883–885.

Bahrami-Samani,E. *et al.* (2014). Computational challenges, tools, and resources for analyzing co- and post-transcriptional events in high throughput. *Wiley Interdiscip. Rev. RNA*, **6**, 291–310.

Basseville,M. and Nikiforof,I.V. (1993) *Detection of abrupt changes: theory and application*. Prentice Hall, Englewood Cliffs, NJ.

Machado-Lima,A. *et al*. (2008) Computational methods in noncoding RNA research. *J. Math. Biol*., **56**, 15–49.

Mattick, J.S. and Makunin, I.V. (2006) Non-coding RNA. *Hum. Mol. Genet.*, **15**, R17–R29.

Molina-Navarro,M.M. *et al.* (2008) Comprehensive transcriptional analysis of the oxidative response in yeast. *J. Biol.Chem.*, **283**, 17908–17918.

Pérez-Ortín,J.E. *et al*. (2012) Genome-wide studies of mRNA synthesis and degradation in eukaryotes. *Biochimi. Biophys. Acta*, **1819**, 604–615.

Suárez,E. *et al.* (2009) Microarray data analysis for differential expression: a tutorial. *PR Health Sci. J*., **28**, 89–104.

VISMapper: ultra-fast exhaustive cartography of viral insertion sites for gene therapy

Juanes *et al. BMC Bioinformatics* (2017) 18:421
DOI 10.1186/s12859-017-1837-z

BMC Bioinformatics

SOFTWARE Open Access

VISMapper: ultra-fast exhaustive cartography of viral insertion sites for gene therapy

José M. Juanes[1,2†], Asunción Gallego[3,4†], Joaquín Tárraga[2,5], Felipe J. Chaves[6,7], Pablo Marín-García[6,8], Ignacio Medina[5], Vicente Arnau[1,2,8] and Joaquín Dopazo[3,9,10*]

Abstract

Background: The possibility of integrating viral vectors to become a persistent part of the host genome makes them a crucial element of clinical gene therapy. However, viral integration has associated risks, such as the unintentional activation of oncogenes that can result in cancer. Therefore, the analysis of integration sites of retroviral vectors is a crucial step in developing safer vectors for therapeutic use.

Results: Here we present VISMapper, a vector integration site analysis web server, to analyze next-generation sequencing data for retroviral vector integration sites. VISMapper can be found at: http://vismapper.babelomics.org.

Conclusions: Because it uses novel mapping algorithms VISMapper is remarkably faster than previous available programs. It also provides a useful graphical interface to analyze the integration sites found in the genomic context.

Keywords: Gene therapy, Viral insertion, Viral integration, Sequence mapping, Genome viewer

Background

The stable, long-term correction of diseases by integrating viral vectors carrying healthy copies defective genes in the patient's genome has become mainstream procedure in clinical gene therapy [1, 2]. However, despite its successful application, viral integration based therapies are not exempt of risks, such as the accidental activation of oncogenes that can cause malignant transformation of the cells [3, 4]. Vector locations in the host genome constitute molecular markers that help monitoring the fate of affected cells. Analysis of vector insertion sites (ISs) is carried out by the amplification (currently using Next Generation Sequencing –NGS- technologies) of sequences from retroviral vectors with a long terminal repeat (LTR). Primers mapping LTRs produce sequence reads with LTR-chromosome junctions, which can be used to accurately determine the chromosomal region of insertion of the viral vector [4]. Such monitoring is required because it is known that distinct gene transfer vectors can have preferences to target gene coding regions, CpG islands, or transcriptional start sites [5–7].

Here we present a new web server, VISMapper, a web tool to manage sequencing data for the detection of viral vector insertion sites in gene therapy experiments. VISMapper is much faster than other alternative software available and provides a comprehensive graphic interface that allows interactive visualization of the viral ISs in the genomic context.

Implementation

VISMapper is written in Node.js (a JavaScript runtime) and uses GenomeMaps [8] for the visual representation of the results in the context of the genome. Thus the resulting viral insertion sites of an experiment can be visualized along with the genomic features they have around, including reads mapped, genes and other type of genomic elements. Supported assemblies for the human genome are GRCh37 and GRCh38.

Cancer genes were taken from the COSMIC [9] database through the CellBase [10] webservices.

* Correspondence: joaquin.dopazo@juntadeandalucia.es; joaquindopazo@gmail.com
†Equal contributors
[3]Clinical Bioinformatics Research Area, Fundación Progreso y Salud, Hospital Virgen del Rocío, 41013 Sevilla, Spain
[9]Bioinformatics and Data Analysis Unit, Genomic Medicine Institute Imegen, Valencia, Spain
Full list of author information is available at the end of the article

Fig. 1 Screenshot showing the different graphical representations in the dashboard: the karyotype viewer and the genome viewer. Also, a table with the list of IS found is displayed

Results

Data upload and workspace

VISMapper reads standard FASTQ or FASTA files containing reads corresponding to the insertion sites of the virus. If FASTA files are provided, they are converted to FASTQ format. Since FASTA files lack the quality parameter, this is set to 20 by default for the FASTQ file generated. A value of 20 minimizes the false positive rate when the original sequences are standard quality. In any case, the use of FASTQ containing quality values is obviously preferable. Files can be ZIP compressed. During the upload, user can optionally provide an email to be notified of the end of the data processing (given the speed of data processing it is usually unnecessary).

Read mapping

Reads in the FASTQ file are mapped onto the reference human genome using BWA [11] or HPG-Align [12]. Typically mapping runtimes are in the range of seconds, which makes of VISMapper a truly interactive and accurate tool for exploring the result of retroviral insertion experiments. IS locations are detected by identified reads partially mapped. We use the CIGAR information for this. When the CIGAR of a mapping contains soft or hard clippings it indicates that the corresponding read

have part of the genome sequence as well as part of the viral sequence. The reads are arranged by chromosome using SAMTools [13] and are inserted in a MySQL database for facilitating a faster access to them.

Dashboard

The Dashboard is a graphical working environment composed by three panels: the karyotype viewer, the genome viewer and the control panel (See Fig. 1). The karyotype viewer provides a general perspective of all the ISs along the chromosomes. Clicking with the left mouse button magnifies the chromosome, with ISs marked as red lines. Exact details on the IS location are provided by setting the cursor over them. A vertical panel on its left (See Fig. 1) allows filtering IS by the number of reads supporting them. It also allows searching those reads which are closer to oncogenes of genes related to specific tumor types. When the mouse hovers the chromosome in the karyotype a detailed view of the selected chromosome with the IS is displayed. Setting the mouse over the ISs pops up information on its exact location and the number of reads supporting it.

A more detailed view of the region in which the ISs occur (that can be selected by clicking in the karyotype viewer) can be obtained with the genome viewer, which implements GenomeMaps [8]. Several tracks are available at different detail level depending on the zoom level in the genome viewer: a) the surrounding genomic region, b) oncogenes located in the neighborhood (the cursor over them displays information on the genes) and c) reads mapped around the IS (again, information on the read, such as strand, mapping quality, etc. is provided by hovering the mouse on them).

Finally, the control panel allows setting a threshold based on the number of reads that support ISs and allows finding specific cancer genes or genes of specific cancer types (see Fig. 1, left part). Specifically, a box allows setting a threshold with the minimum number of reads to consider a IS (5 by default). The second box allows selecting a specific oncogene (can be searched by name or selected from a list). The list of oncogenes has been extracted from COSMIC. Another box allows displaying only the genes known to be associated with a given tumor.

Report

The control panel allows generating a comprehensive tabular report of the results found. The button report directs to another page with a table containing all the ISs found that can be arranged by all the criteria shown in the header of the columns (chromosome, position, quality, etc.) Different filters (number of reads that support the IS and distance to a cancer gene) can be applied to expand or reduce the number of ISs to consider. This list can be downloaded in tab delimited format and a BAM file with the alignments found by the mapper can also be downloaded.

For any IS considered with the filtering schema used, the report contains the following items:

- Chromosome
- Position

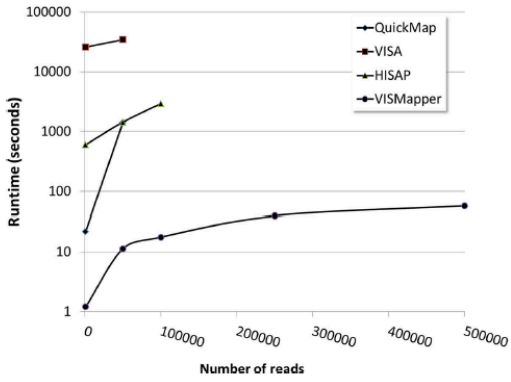


Fig. 2 Runtimes observed for different programs QuickMap (line with diamonds), VISA (line with squares) HISAP (line with triangles) and VISMapper (line with circles) with datasets of increasing sizes. In the case of QuickMap, VISA and HISAP, the lines are interrupted according to internal hard limits for the number to sequences that the programs can process

Table 1 Comparison of VISA and VISMapper using four datasets generated with the IS generator program from the the VISMapper website (https://visa.pharmacy.wsu.edu/bioinformatics/random_site_generator.html)

Dataset	Input size (reads)	Insertion sites	Performance	VISA	VISMapper
Input 1			Runtime	~72 h	~60 s
	100,000	100,000	IS detected	99,694	99,793
			Total sequences mapped	99,694	99,881
Input 2			Runtime	~72 h	~60 s
	50,000	50,000	IS detected	49,854	49,897
			Total sequences mapped	49,855	49,936
Input 3			Runtime	~72 h	~30 s
	10,000	10,000	IS detected	9992	9969
			Total sequences mapped	9995	9981
Input 4			Runtime	~5 h	~30 s
	1000	1000	IS detected	906	929
			Total sequences mapped	906	930

Runtimes of both programs are shown for the four datasets, along with the number of sequences correctly mapped, that correspond to the IS detected, and the total number of sequences mapped, which in both cases is slightly superior, demonstrating a low rate of false positives in both cases

- Number of reads mapped in this position
- Average quality of all the reads mapped in the position
- Closest oncogene
- Distance to the oncogene (0 means that the IS maps within the oncogene)
- Position of the oncogene with respect to the IS
- Entrez entry of the oncogene
- URL to the Entrez entry of the oncogene

Comparison to other web servers for viral is mapping

There are a few web servers for viral vector insertion site analysis, such as, HISAP [14], SeqMap (requires user registration) or QuickMap [15], or the recently published VISA [16]. However, all of them use BLAST [17] or BLAT [18] for read mapping that involve comparatively much longer runtimes. Figure 2 shows a comparative of runtimes where the increase in speed gained by the use of more sophisticated mapping algorithms in VISMapper is obvious. The data used in the comparison were taken from the VISA website and can also be downloaded at the VISMapper documentation site (https://github.com/jmjuanes/vismapper/tree/master/ismapper-test).

In addition, a more detailed comparison was made with the VISA program by generating 4 datasets with known number of IS using the IS generator program from the VISA website (https://visa.pharmacy.wsu.edu/bioinformatics/random_site_generator.html). Table 1 shows the results of the comparison. Relative runtimes are similar to the ones shown in Fig. 2. While both methods give a very small number of false positives, in general VISMapper is able to map a higher percentage of sequences and found more IS sites than VISA.

In addition, QuickMap does not process more than 50,000 sequences and VISA limits are between 50,000 and 100,000. HISAP could manage up to 100,000 in about 50 min, but cannot arrive to 250,000 sequences. Moreover, none of the other programs provide a graphic interface to analyze the results. Furthermore, QuickMap and HISAP do not support GRCh38.

Conclusions

Because of its speed and sensitivity, VISMapper constitutes an attractive alternative to the options available for viral insertion site analysis. VISMapper offers a unique, interactive graphical working environment that allows a detailed and exhaustive exploration of the consequences and potential risks of the viral vectors inserted in the analyzed genome.

Abbreviations

BAM: Binary alignment map; BWA: Burrows–wheeler algorithm; IS: Insertion Site; LTR: Long terminal repeat; NGS: Next generation sequencing

Acknowledgements

Not applicable

Funding

This work is supported by grants BIO2014–57291-R from the Spanish Ministry of Economy and Competitiveness (MINECO), and Plataforma de Recursos Biomoleculares y Bioinformáticos PT13/0001/0007 from the ISCIII, both co-funded with European Regional Development Funds (ERDF); H2020-INFRADEV-1-2015-1 ELIXIR-EXCELERATE (ref. 676,559). None of the funding bodies played any role in the design or conclusions of the study.

Availability of data and materials

VISMapper can be found at: http://vismapper.babelomics.org. VISMapper code can be found in the GitHub repository https://github.com/jmjuanes/vismapper. Associated documentation can be found at: https://github.com/jmjuanes/vismapper/wiki. The data used in the general comparison can be found at: https://github.com/jmjuanes/vismapper/tree/master/ismapper-test

Authors' contributions
JMJ, and AG programmed the code, JT and IM programmed and optimized the mapping of sequences, FJC and PMG helped with the programming, VA coordinated the programming work and JD conceived the work and wrote the paper. All the authors read and approved the final manuscript.

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]Departamento de Informática, Escuela Técnica Superior de Ingeniería (ETSE), Universidad de Valencia, 46100 Valencia, Burjassot, Spain. [2]Computational Genomics Department, Prince Felipe Research Center, 46012 Valencia, Spain. [3]Clinical Bioinformatics Research Area, Fundación Progreso y Salud, Hospital Virgen del Rocío, 41013 Sevilla, Spain. [4]Bioinformatics in Rare Diseases (BiER), Centro de Investigación Biomédica en Red de Enfermedades Raras (CIBERER), Hospital Virgen del Rocío, 41013 Sevilla, Spain. [5]HPC Service, University Information Services, University of Cambridge, Cambridge, UK. [6]Genotyping and Genetic Diagnosis Unit, Health Research Institute, INCLIVA, Valencia, Spain. [7]CIBERDem, Health Institute Carlos III, Madrid, Spain. [8]Institute for Integrative Systems Biology (I2SysBio), Universidad de Valencia-CSIC, 46980 Valencia, Paterna, Spain. [9]Bioinformatics and Data Analysis Unit, Genomic Medicine Institute Imegen, Valencia, Spain. [10]Functional Genomics Node, INB-ELIXIR-es, Hospital Virgen del Rocío, 42013 Sevilla, Spain.

Received: 13 February 2017 Accepted: 12 September 2017
Published online: 20 September 2017

References

1. Gaspar HB, Parsley KL, Howe S, King D, Gilmour KC, Sinclair J, Brouns G, Schmidt M, Von Kalle C, Barington T, et al. Gene therapy of X-linked severe combined immunodeficiency by use of a pseudotyped gammaretroviral vector. Lancet. 2004;364(9452):2181–7.
2. Cartier N, Hacein-Bey-Abina S, Bartholomae CC, Veres G, Schmidt M, Kutschera I, Vidaud M, Abel U, Dal-Cortivo L, Caccavelli L, et al. Hematopoietic stem cell gene therapy with a lentiviral vector in X-linked adrenoleukodystrophy. Science. 2009;326(5954):818–23.
3. Cavazzana-Calvo M, Payen E, Negre O, Wang G, Hehir K, Fusil F, Down J, Denaro M, Brady T, Westerman K, et al. Transfusion independence and HMGA2 activation after gene therapy of human beta-thalassaemia. Nature. 2010;467(7313):318–22.
4. Paruzynski A, Arens A, Gabriel R, Bartholomae CC, Scholz S, Wang W, Wolf S, Glimm H, Schmidt M, von Kalle C. Genome-wide high-throughput integrome analyses by nrLAM-PCR and next-generation sequencing. Nat Protoc. 2010;5(8):1379–95.
5. Schroder AR, Shinn P, Chen H, Berry C, Ecker JR, Bushman F. HIV-1 integration in the human genome favors active genes and local hotspots. Cell. 2002;110(4):521–9.
6. Mitchell RS, Beitzel BF, Schroder AR, Shinn P, Chen H, Berry CC, Ecker JR, Bushman FD. Retroviral DNA integration: ASLV, HIV, and MLV show distinct target site preferences. PLoS Biol. 2004;2(8):E234.
7. Wu X, Li Y, Crise B, Burgess SM. Transcription start regions in the human genome are favored targets for MLV integration. Science. 2003;300(5626):1749–51.
8. Medina I, Salavert F, Sanchez R, de Maria A, Alonso R, Escobar P, Bleda M, Dopazo J. Genome maps, a new generation genome browser. Nucleic Acids Res. 2013;41(Web Server issue):W41–6.
9. Forbes SA, Bindal N, Bamford S, Cole C, Kok CY, Beare D, Jia M, Shepherd R, Leung K, Menzies A, et al. COSMIC: mining complete cancer genomes in the catalogue of somatic mutations in cancer. Nucleic Acids Res. 2011;39(Database issue):D945–50.
10. Bleda M, Tarraga J, de Maria A, Salavert F, Garcia-Alonso L, Celma M, Martin A, Dopazo J, Medina I. CellBase, a comprehensive collection of RESTful web services for retrieving relevant biological information from heterogeneous sources. Nucleic Acids Res. 2012;40(Web Server issue):W609–14.
11. Li H, Durbin R. Fast and accurate short read alignment with burrows-wheeler transform. Bioinformatics. 2009;25(14):1754–60.
12. Tarraga J, Arnau V, Martinez H, Moreno R, Cazorla D, Salavert-Torres J, Blanquer-Espert I, Dopazo J, Medina I. Acceleration of short and long DNA read mapping without loss of accuracy using suffix array. Bioinformatics. 2014;30(23):3396–8.
13. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. The sequence alignment/map format and SAMtools. Bioinformatics. 2009;25(16):2078–9.
14. Arens A, Appelt JU, Bartholomae CC, Gabriel R, Paruzynski A, Gustafson D, Cartier N, Aubourg P, Deichmann A, Glimm H, et al. Bioinformatic clonality analysis of next-generation sequencing-derived viral vector integration sites. Human gene therapy methods. 2012;23(2):111–8.
15. Appelt JU, Giordano FA, Ecker M, Roeder I, Grund N, Hotz-Wagenblatt A, Opelz G, Zeller WJ, Allgayer H, Fruehauf S, et al. QuickMap: a public tool for large-scale gene therapy vector insertion site mapping and analysis. Gene Ther. 2009;16(7):885–93.
16. Hocum JD, Battrell LR, Maynard R, Adair JE, Beard BC, Rawlings DJ, Kiem HP, Miller DG, Trobridge GD. VISA--vector integration site analysis server: a web-based server to rapidly identify retroviral integration sites from next-generation sequencing. BMC Bioinformatics. 2015;16:212.
17. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10.
18. Kent WJ. BLAT--the BLAST-like alignment tool. Genome Res. 2002;12(4):656–64.