Escuela Técnica Superior de
Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de ArrayExpress Processor for TriGen: una herramienta bioinformática para la explotación automatizada de datos genómicos en modelos de aprendizaje automático

Realizado por
Alejandro Márquez González

Para la obtención del título de
Grado en Ingeniería de la Salud - Informática Clínica

Dirigido por
David Gutierrez Áviles

En el departamento de
Lenguajes y Sistemas Informáticos

Convocatoria de Julio, curso 2024/25

Agradecimientos

Todo mi esfuerzo y agradecimiento es por y para mis hermanos, Javier y Pablo. Al mayor, por ser mi guía constante; al pequeño, por motivarme a ser mejor persona cada día, para poder ser un buen ejemplo para él.
A mis padres, por todo el amor incondicional que me han dado y por cada sacrificio que han hecho por mí. Toda mi ambición nace del deseo de poder devolvérselo.
A mis abuelos, primos y tíos, por acompañarme y sostenerme con su cariño durante estos cuatro años.
A mis amigos de Córdoba, porque aunque la distancia se note, el tiempo jamás ha pasado entre nosotros.
A mis amigos de Sevilla, por convertirse en mi familia lejos de casa y hacer que este camino haya sido posible.
A todos mis profesores, desde el colegio hasta la universidad, por su dedicación y por vuestra labor, tan valiosa para la sociedad.

Resumen

Se ha desarrollado *ArrayExpress Processor for TriGen*, una herramienta bioinformática completa e interactiva que permite automatizar el flujo de análisis de expresión génica a partir de experimentos públicos disponibles en el repositorio ArrayExpress. Esta herramienta, accesible tanto mediante scripts en R como a través de una interfaz web desarrollada con Shiny, integra funcionalidades para la descarga, preprocesamiento y estructuración de datos transcriptómicos, la ejecución del algoritmo de triclustering TriGen, el análisis funcional mediante la API de PantherDB y la visualización interactiva de los resultados. Todo el sistema ha sido diseñado para ser reproducible, modular y fácilmente extensible.

Para ello, se ha implementado un conjunto de funciones en R que permiten desde la obtención de datos crudos y metadatos de cada experimento, hasta la generación automática de los archivos de entrada que requiere el algoritmo TriGen. Además, se ha diseñado un proceso de enriquecimiento funcional basado en llamadas directas a la API de PantherDB, integrando visualizaciones y filtrado de resultados. También se han incorporado funcionalidades para explorar los triclústeres generados, visualizar la variación de los niveles de expresión por condición y tiempo, y consultar la información biológica de los genes implicados. Todo ello ha sido encapsulado en una aplicación Shiny que centraliza la ejecución de los análisis, permitiendo que el usuario interactúe con la herramienta introduciendo únicamente el identificador del experimento.

Este trabajo se enmarca en el objetivo propuesto por el TFG titulado ”Explotación de repositorios de datos genómicos para la aplicación en modelos Machine Learning”, ya que permite transformar datos genómicos crudos procedentes de repositorios públicos en conjuntos de datos estructurados y enriquecidos funcionalmente, listos para su análisis mediante técnicas de aprendizaje automático. La herramienta desarrollada actúa como puente entre el acceso a datos reales y su aplicación en modelos computacionales, resolviendo las principales barreras de interoperabilidad, curación y preparación de datos en el ámbito de la genómica funcional. Este trabajo sienta así las bases para futuros desarrollos que integren directamente modelos de machine learning aplicados al descubrimiento de patrones biológicos.

Palabras clave: Genómica, ArrayExpress, TriGen, PantherDB, Bioinormática.

Abstract

We have developed ArrayExpress Processor for TriGen, a comprehensive and interactive bioinformatics tool designed to automate the workflow for gene expression analysis from public experiments available in the ArrayExpress repository. This tool, accessible both through R scripts and a Shiny-based web interface, integrates functionalities for downloading, preprocessing, and structuring transcriptomic data, executing the TriGen triclustering algorithm, performing functional enrichment analysis via the PantherDB API, and interactively visualizing the results. The entire system has been designed to be reproducible, modular, and easily extensible.

To achieve this, a set of R functions has been implemented to handle everything from retrieving raw data and experiment metadata to automatically generating the input files required by the TriGen algorithm. Additionally, a functional enrichment process has been designed based on direct calls to the PantherDB API, including integrated visualization and result filtering. Features have also been added to explore the generated triclusters, visualize expression level variation across conditions and time points, and query the biological information of the involved genes. All these components have been encapsulated in a Shiny application that centralizes the analysis execution, allowing the user to interact with the tool by simply entering the experiment identifier.

This project aligns with the goal proposed in the Bachelor's Thesis titled "Exploitation of Genomic Data Repositories for Application in Machine Learning Models", as it enables the transformation of raw genomic data from public repositories into structured and functionally enriched datasets, ready for analysis using machine learning techniques. The developed tool serves as a bridge between access to real-world data and its application in computational models, addressing key challenges in interoperability, data curation, and preparation within the field of functional genomics. This work thus lays the groundwork for future developments integrating machine learning models directly into biological pattern discovery.

Keywords: Genomics, ArrayExpress, TriGen, PantherDB, Bioinformatics.

Índice general

Índice de figuras

Índice de extractos de código

1. Introducción

La titulación de Ingeniería de la Salud (especialmente la mención en Informática Clínica) se centra en formar ingenieros capaces de combinar fundamentos de ingeniería, informática y ciencias de la salud gracias a la sólida base de conocimiento biológico en distintas ramas(Anatomía, Fisiología, Bioquímica, Genética…) y el aprendizaje de una amplia gama de lenguajes de programación(R, Python, Java, MySql…). Estos rasgos, potencian y describen a la perfección las cualidades de un bioinformático. Es por ello, que un trabajo como el realizado es perfecto para finalizar y demostrar todo lo aprendido.

El papel del bioinformático ha ido emergiendo y adaptándose durante las últimas décadas, podríamos decir, que es un científico interdisciplinar capaz de gestionar, integrar y analizar grandes volúmenes de datos biológicos mediante la combinación de conocimientos relacionados con la Biología, Informática y la Matemática. Podríamos describirlo como un analista de datos derivados de las tecnologías de secuenciación masiva como la genómica, proteómica y la transcriptómica.

De esta forma, se introduce el concepto de ciencias ómicas, en plural, ya que albergan un conjunto de disciplinas dedicadas a caracterizar y cuantificar las moléculas de un mismo tipo presentes en un organismo, tejido, célula o microbioma mediante el empleo de plataformas analíticas de alto rendimiento.

Entre los retos que debe afrontar un científico de este perfil, se encuentra: la complejidad y gran magnitud de datos generados en investigaciones biomédicas, la transformación de datos en bruto para la ayuda en la toma de decisiones clínicas, farmacológicas y biotecnológicas y la necesidad de hacer las herramientas y datos reutilizables en el tiempo y fáciles de divulgar entre la comunidad científica.

Además, no solo asume un papel esencial en el ámbito científico o de la investigación sino que también actúa como puente entre el personal sanitario de todo ámbito(médicos, farmacólogos, biotecnólogos…), gracias a las competencias en el campo de la Biología, y las nuevas tecnologías, el Big Data y la Inteligencia Artificial, extraídas del campo de la Ingeniería Informática.

2. Motivación personal, retos y objetivos

Personalmente, la elección de este Trabajo Fin de Grado nace de mi interés por adentrarme en el mundo de la investigación científica, especialmente en el ámbito de la bioinformática y el análisis de datos así como desarrollar una herramienta propia funcional, automatizada y útil desde cero, enfrentándome a los retos reales que implica. Esta experiencia me ha permitido no solo profundizar en conceptos técnicos y científicos, sino también consolidar mi interés por seguir creciendo en el campo de la investigación computacional ligada al campo de la salud.

Tras la primera reunión con el profesor Don David Gutiérrez Avilés, se acordó a grandes rasgos cuál sería el despliegue final del Trabajo Fin de Grado. El desarrollo de una herramienta, ya fuera una web o aplicación de escritorio que:

- Sea capaz de acceder a la base de datos ArrayExpress para descargar sus experimentos.
- Sea capaz de procesar los datos, ya fuera en bruto o ya procesados, mediante un pipeline hasta tener el formato válido que el algoritmo TriGen requiere.
- Represente los resultados del algoritmo ya fuera mediante la exportación de informes o la representación de gráficas.
- Tuviera interfaz gráfica para facilitar la explotación de la base de datos para la aplicación en modelos Machine Learning como el utilizado en el algoritmo TriGen.

Estos objetivos no sólo se han cumplido con éxito, sino que se han superado mediante el despliegue del algoritmo desde la propia herramienta y utilizando e integrando nuevos recursos bioinformáticos para el análisis de los resultados del algoritmo que no se habían utilizado previamente, entre otras funcionalidades implementadas.

Para conseguir el resultado obtenido, he debido afrontar ciertos retos tanto de estudio como programáticos como:

- La comprensión de la estructura y proceso de uso de plataformas de secuenciación masiva, así como la comprensión de la representación de sus resultados.

- Estudio de la organización de los experimentos en la base de datos ArrayExpress.

- Estudio de la estructura de los experimentos, así como sus metadatos y datos en bruto.

- Estudio del algoritmo Trigen en sí mismo, los datos de entrada que requiere y los datos de salida que devuelve.

- Búsqueda de librerías actualizadas capaces de procesar y manejar los datos en bruto de los experimentos(la mayoría se encuentran obsoletas).

Todos estos retos han sido afrontados pasando muchas horas frente al ordenador, buscando mucha información en artículos, foros y otros medios de divulgación, con un proceso de prueba y error y, por supuesto, gracias a la ayuda de mi tutor.

3. Planificación temporal del proyecto.

La planificación temporal se ha elaborado para asegurar una distribución equilibrada del esfuerzo a lo largo del desarrollo del proyecto. Se han cubierto los tiempos requeridos por los créditos asignados al TFG, equivalentes a un total de 150 horas de trabajo por cada 6 créditos ECTS.

Para estructurar esta planificación, se ha tomado como referencia la técnica del desglose del trabajo (WBS, Work Breakdown Structure), dividiendo el proyecto en fases y actividades específicas que permiten un control más preciso del avance. Esta estrategia no solo mejora la organización del tiempo, sino que facilita la identificación temprana de desviaciones y cuellos de botella. Este enfoque asegura una gestión eficaz del tiempo y una ejecución coherente con los objetivos del proyecto.

Fase	Actividad	% de fase	Horas
Reuniones iniciales y definición del TFG	Primera reunión con el tutor para definir el enfoque y objetivos	15	5
	Identificación del algoritmo TriGen como núcleo del análisis		10
	Decisión de trabajar con la base de datos ArrayExpress		10
	Planificación inicial de tareas y cronograma aproximado		20
Investigación preliminar	Estudio del triclustering y del algoritmo TriGen	30	25
	Revisión de conceptos de transcriptómica, microarrays y RNA-Seq		20
	Estudio del uso de BioConductor, Affy, ArrayExpress, PantherDB		25
	Exploración de Gene Ontology, enriquecimiento funcional y herramientas similares		20
Implementación y desarrollo de la herramienta	Desarrollo de funciones para descarga de datos desde ArrayExpress y BioStudies	45	20
	Automatización del preprocesamiento y generación de archivos de entrada para TriGen		35
	Ejecución y control del algoritmo TriGen desde R		20
	Análisis funcional con la API de PantherDB		15
	Desarrollo de visualizaciones interactivas con Shiny		25
	Despliegue local de la app Shiny y manejo de dependencias		20
Redacción de la memoria	Redacción del resumen, introducción y motivación	10	5
	Documentación técnica del desarrollo y funcionalidades de la herramienta		15
	Conclusiones, líneas futuras y edición final en LaTeX		10

Tabla 3.1: Planificación temporal detallada con distribución de actividades, porcentaje asignado y horas estimadas.

Con la finalidad de representar visualmente la planificación temporal de tareas a lo largo del tiempo, utilizaremos el diagrama de Gantt. Esta representación permite identificar el orden de ejecución de las tareas y el reparto del esfuerzo a lo largo del proyecto. Se ha construido a partir de las actividades detalladas previamente, considerando una equivalencia aproximada de 1 día por cada 5 horas de dedicación, esta escala permite visualizar de manera proporcional el esfuerzo estimado en cada etapa del desarrollo.

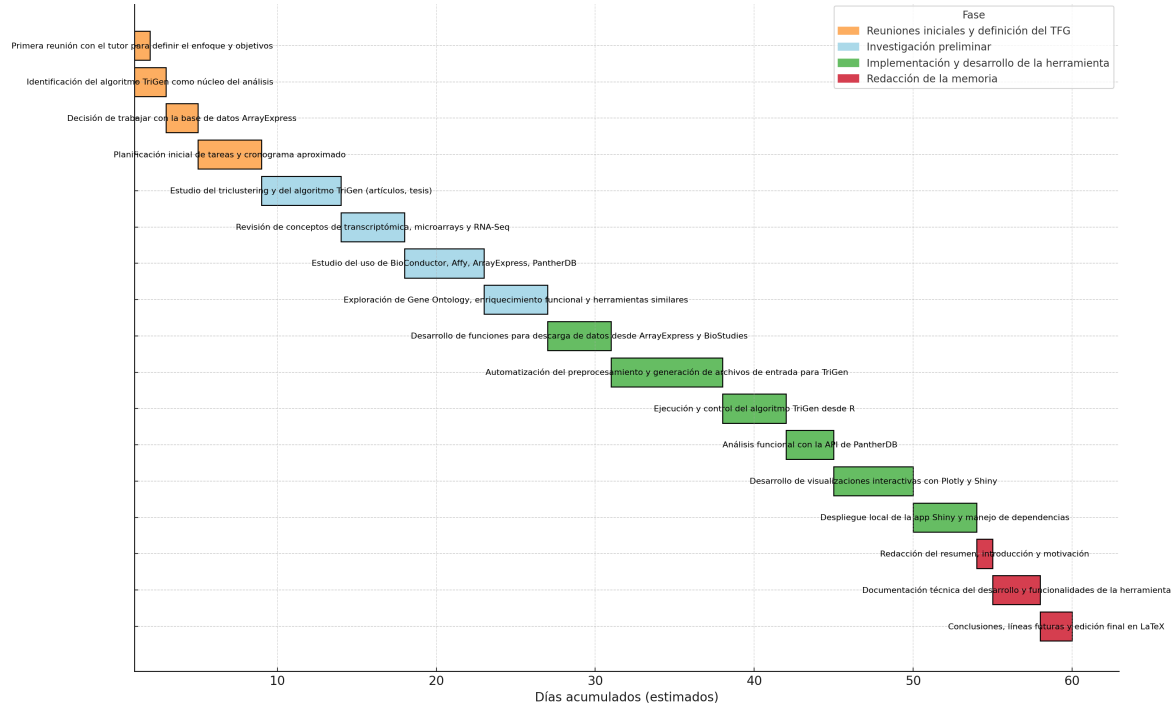


Figura 3.1: Diagrama de Gantt.

4. Ciencias Ómicas

Las ciencias ómicas se refieren a los enfoques novedosos que se centran en el análisis del genoma del ser humano y otros seres vivos, así como sus interrelaciones. Su objetivo es estudiar las influencias en el fenotipo de las interacciones entre genes y moléculas.

Este enfoque [1] puede ser clasificado bajo distintos criterios. El más común diferencia según la naturaleza de la biomolécula estudiada:

- **Genómica:** Estudio del genoma de un organismo, su estructura, función, variaciones, la interacción con el bioma y su impacto en la salud.
- **Transcriptómica:** Estudio del conjunto de transcritos de ARN en una célula, tejido u organismo en un momento concreto.
- **Proteómica:** Estudio del conjunto completo de proteínas expresadas en una célula, tejido u organismo en un momento dado; analiza la estructura, función, modificaciones e interacciones post-traduccionales.
- **Metabolómica:** Estudio de las pequeñas moléculas producidas en el metabolismo (metabolitos) presentes en una célula, tejido u organismo en un momento dado.

Existen otros métodos para clasificar estas ciencias como: El nivel de regulación y función biológica(Epigenómica, Interactómica y Microbioma) o la técnica de análisis utilizada.

En este trabajo nos centraremos en la genómica y la transcriptómica al ser, en gran proporción, la disciplina de la que tratan los experimentos que se encuentran en ArrayExpress.

4.1. Genómica

Para ponernos un poco en contexto y a modo de introducción, debemos ir un poco atrás en el tiempo, concretamente los 80, cuando Frederick Sanger conocido como “el padre de la genómica” y ganador de dos premios Nobel, desarrolló un método para la lectura de bases A(Adenina), C(Citosina, G(Guanina) y (Tinina) en hebras de ADN[2]. Este método fue clave en lo que el comienzo del análisis en

laboratorios del genoma humano se refiere y estaba basado en el uso de enzimas para la copia de fragmentos de ADN y "terminadores de cadena", la separación de las cadenas resultantes mediante electroforesis en gel y la posterior visión, permitiendo leer la secuencia y la reconstrucción de la secuencia completa del ADN original.

Como prueba de su trabajo y la eficacia de su método, Sanger consiguió secuenciar genomas de distintos tamaños: desde el de un simple virus bacterial con más de 5000 nucleótidos hasta virus bacterianos más complejos con más de 48000 nucleótidos.

Su contribución fue clave para la consecución del Proyecto Genoma Humano cuyo objetivo consistía en la secuenciación y mapeo completo del ADN humano. El proyecto se llevó a cabo entre 1990 hasta 2003.

La genómica consiste en el estudio del genoma de un organismo en un momento dado. Sus estudios se caracterizan por el análisis al mismo tiempo de gran cantidad de genes[3]. A partir del 2003 surgieron dos grandes enfoques en los que se centran la mayoría de los estudios relacionados con la secuencia de ADN que conforma un organismo, estos son:

- **Genómica estructural:** Este enfoque se basa en la construcción de mapas genéticos, la identificación de genes y sus características y la comparación entre estructuras genómicas.

- **Genómica funcional:** Se centra en el estudio de la función de aquellas secuencias de ADN con la capacidad de generar moléculas funcionales (ARN, genes y proteínas) en el genoma y en el rol que estos elementos toman en procesos celulares como la transcripción, traducción, regulación génica, metabolismo y ciclo celular[4].

Los grandes temas de estudio de estos enfoques son el mapeo y secuenciación del genoma además de la comparación con otras estructuras genómicas y el análisis de sus funcionalidades.

Los procesos de mapeo del genoma consisten en encontrar la ubicación relativa de los genes, mutaciones o rasgos de un cromosoma, mediante la creación de mapas de ligamiento genético(indican la posición relativa de marcadores genéticos según la frecuencia con la que se heredan juntos), mapas físicos(localizan marcas específicas en el ADN sin considerar patrones de herencia) y mapas citológicos(identifican patrones de bandas en cromosomas teñidos bajo microscopio).[5]

La descripción del genoma a nivel de pares de bases individuales se conoce como secuenciación del ADN, existen dos estrategias principales para realizar dicha descripción: el Enfoque Shotgun(secuenciar al azar fragmentos de ADN clonados

desde ambos extremos) y el enfoque jerárquico(mapeo del genoma en fragmentos largos clonados para su posterior secuenciación).

4.1.1. Tecnologías innovadoras

Next-generation Sequencing(NGS).

Como introdujimos anteriormente, Sanger fue el líder respecto a los métodos de secuenciación en lo que a primera generación se refiere. Por otro lado, NGS es considerada una tecnología de secuenciación masiva en paralelo de alto rendimiento de segunda generación[6]. Explicado de forma más sencilla, NGS es una tecnología para determinar las bases o nucleótidos de miles a millones de moléculas de ADN simultáneamente, se consideran de alto rendimiento por la gran reducción de costes que suponen. Por ejemplo, mientras que el presupuesto de la primera secuenciación del genoma humano era de 3000 millones de euros, en las últimas décadas se ha conseguido por menos de 1 millón de euros. Desde que esta tecnología salió al mercado, se desarrollaron distintas plataformas para su implementación. Veámos las 3 principales, ordenadas cronológicamente según su año de comercialización:

- **Roche 454 (2005):** Fue la primera plataforma NGS introducida al mercado. Se basaba en la técnica de pirosecuenciación, es decir, la detección de pirofosfato inorgánico (PPi) liberado durante la síntesis de ADN. Esto permitía reconocer la incorporación de nucleótidos por una señal luminosa generada mediante reacciones químicas. Era capaz de secuenciar 0.7 Gigabases en 23 horas. Actualmente está en desuso debido a la reducción de costes de otras tecnologías más eficientes.

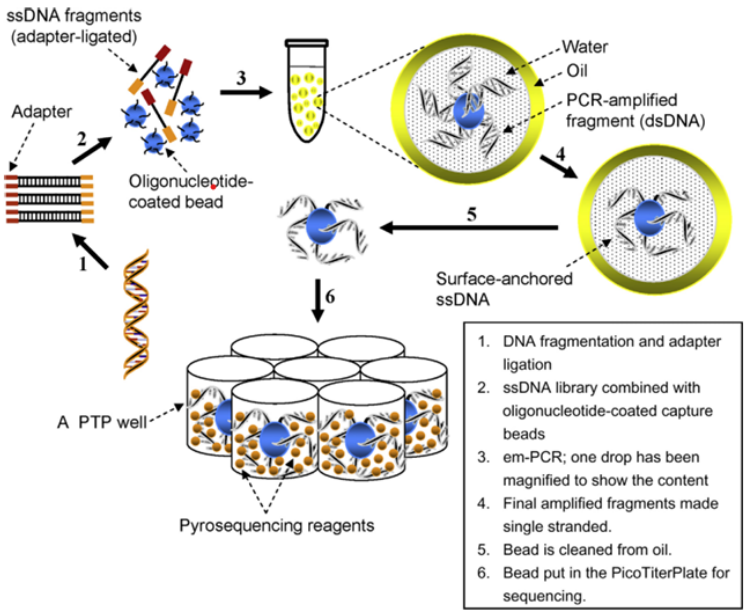


Figura 4.1: Principios del proceso de secuenciación Roche 454

- **Illumina Solexa (2008):** Esta plataforma revolucionó la secuenciación al emplear nucleótidos etiquetados y la secuenciación por síntesis. Permite secuenciar 600 Gigabases en una sola ejecución. La técnica se basa en la fijación de fragmentos de ADN que se amplifican y luego se secuencian añadiendo nucleótidos marcados con fluorescencia, detectando cada incorporación ciclo a ciclo. Es actualmente la técnica más utilizada tanto en genómica como en transcriptómica y diagnósticos genéticos, por su rapidez y bajo coste.

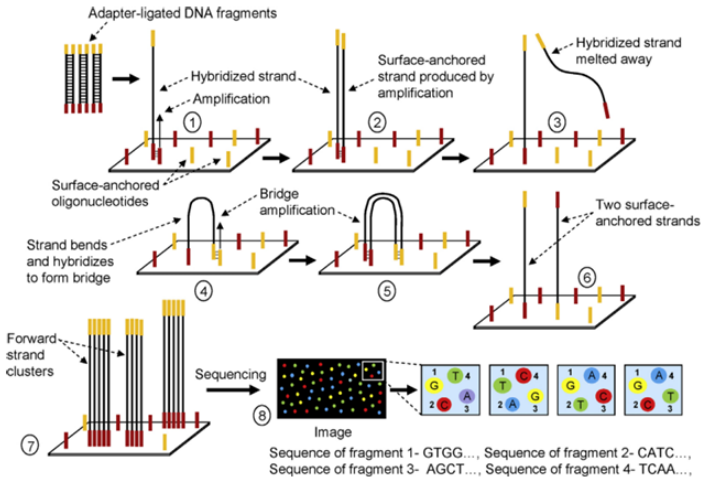


Figura 4.2: Principios del proceso de secuenciación Illumina Solexa

- **ABI SOLiD (2008):** Utiliza ligamiento y detección de oligonucleótidos marcados con fluorescencia para la secuenciación. Puede alcanzar entre 80–100 Gigabases por ejecución. El proceso implica la hibridación y ligamiento de sondas al ADN, y cada nucleótido se codifica dos veces para mayor precisión. Aunque ofrece alta exactitud, ha sido reemplazada por tecnologías más rápidas y económicas.

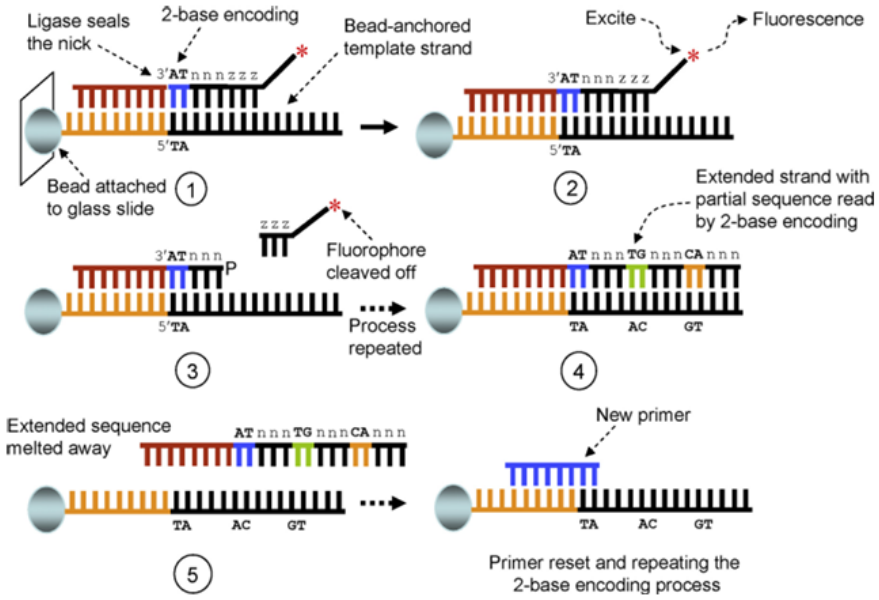


Figura 4.3: Principios del proceso de secuenciación ABI SOLiD

Aunque estas plataformas han conseguido resolver el problema de coste en plataformas de el alto rendimiento, continúa existiendo el reto en la mejora de la eficiencia en el campo de la secuenciación. Es por ello, que uno de los objetivos futuros es el desarrollo de Next-Next Generation Sequencing Technologies o lo que es lo mismo, de tercera generación[1]. Estas estarían caracterizadas por ser tecnologías de secuenciación de molécula única, sin amplificación PCR, preparación menos compleja de las muestras, ausencia de pausas en la secuenciación tras la incorporación de cada base (lo que aumenta la velocidad de secuenciación), mayor longitud de las lecturas, disminución de la velocidad de secuenciación...

Existen algunas tecnologías actuales, como *Ion Torrent* y *Helicos*, que se sitúan entre la segunda y tercera generación, utilizando secuenciación por síntesis con detección óptica o de pH. Sin embargo, la única tecnología de tercera generación consolidada es la secuenciación en tiempo real de molécula única ***SMRT*** de PacBio, que permite observar directamente la síntesis de ADN mediante nanoestructuras llamadas guías de onda de modo cero ZMW. Esto permite lecturas más largas y tiempos de secuenciación más rápidos. También se investigan métodos como microscopía electrónica y secuenciación por nanoporos.

4.2. Transcriptómica

Tal y como se postula en el dogma central de la Biología, el ADN se convierte en ARN por medio del proceso de transcripción, dando lugar a los transcritos de ARN tras un proceso coordinado de expresión génica. La suma de todos estos transcritos

da lugar al Transcriptoma o lo que es lo mismo, el conjunto completo de moléculas de ARN en una célula, población de células o un organismo[7].

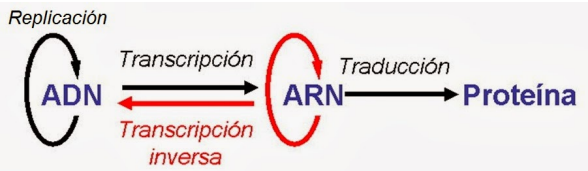


Figura 4.4: Dogma central de la biología

En consecuencia, podemos definir la Transcriptómica como el análisis de este conjunto completo bajo condiciones fisiológicas específicas con la finalidad de medir la expresión de los genes.

Esta disciplina es una herramienta fundamental dentro de la genómica funcional ya que permite identificar los genes que se encuentran activos y en qué medida, ayudando a entender procesos celulares y respuestas a distintos estímulos.

La primera publicación que trato de capturar el transcriptoma se promulgó en la década de los noventa[8], cuando se reportaron más de 500 transcritos del cerebro humano, desde ese momento los retos en este campo se han ido redefiniendo década tras década conforme iban apareciendo nuevas técnicas que mejoraban tanto en sensibilidad como costes.

4.2.1. Tecnologías Innovadoras

Microarray

Los avances más recientes en lo que Transcriptómica se refiere, se centran en el uso de microarrays, esta es una tecnología innovadora cuyo objetivo es el estudio de muchos genes al mismo tiempo, colocando gran cantidad de secuencias génicas en un portaobjetos[9]. El método se basa en la cantidad de luz medible que produce el apareamiento de bases complementarias muestra-secuencia, identificando los genes que se expresan en esa muestra. Nos centraremos más en esta tecnología ya que, en nuestro repositorio de datos ArrayExpress, la mayoría de experimentos útiles para nuestro estudio, la utilizan.

Cada uno de estos microarrays están compuestos de los siguientes elementos fundamentales:

- **Soporte físico:** Superficie donde se fijan las sondas de ADN, generalmente compuesta por una lámina de vidrio, silicio o plástico.

- **Sondas (probes):** Fragmentos cortos de ADN de secuencia conocida,

colocados sobre el soporte en posiciones específicas (locations o sitios). Dependiendo del modelo de la plataforma, puede haber desde cientos hasta millones de sondas.

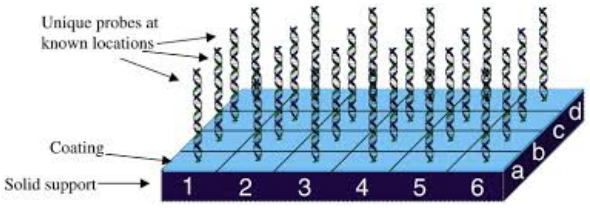


Figura 4.5: Estructura típica de una plataforma de secuenciación

Para clasificarlos, se utiliza el criterio de densidad de la plataforma, es decir, según la cantidad de sitios que tenga, se diferencian dos tipos:

- **Alta densidad:** Contienen desde miles hasta millones de spots en cada unidad de área de la plataforma, lo que permite analizar miles de genes simultáneamente. Son las más utilizadas en experimentos de expresión génica, como en nuestro caso. Un ejemplo representativo son los chips de Affymetrix.

- **Baja densidad:** Contienen entre 25 y 10,000 sitios. Son más específicos y están diseñados para objetivos concretos. Suelen incorporar dispositivos electrónicos que permiten el monitoreo directo. Un ejemplo de estas plataformas son los NanoChip de Nanogen.

El proceso de uso de esta tecnología se puede describir en seis etapas clave:

1. **Extracción de ARN:** Se extrae ARN de una muestra biológica, el cual se convierte en ADN complementario (ADNc) y se marca con un fluoróforo.

2. **Aplicación de la muestra:** El ADNc marcado se aplica sobre el portaobjetos (lámina de vidrio) que contiene las sondas.

3. **Hibridación:** El ADNc se hibrida con las sondas complementarias presentes en el chip, bajo condiciones específicas controladas.

4. **Escaneo del chip:** Un lector láser escanea la superficie del chip y detecta la fluorescencia emitida en cada spot, la cual es proporcional al nivel de hibridación.

5. **Procesamiento de datos:** La señal fluorescente se transforma en valores numéricos tras un proceso de eliminación de ruido, corrección de fondo y análisis estadístico. Se dedicará un apartado específico de esta memoria al método RMA.

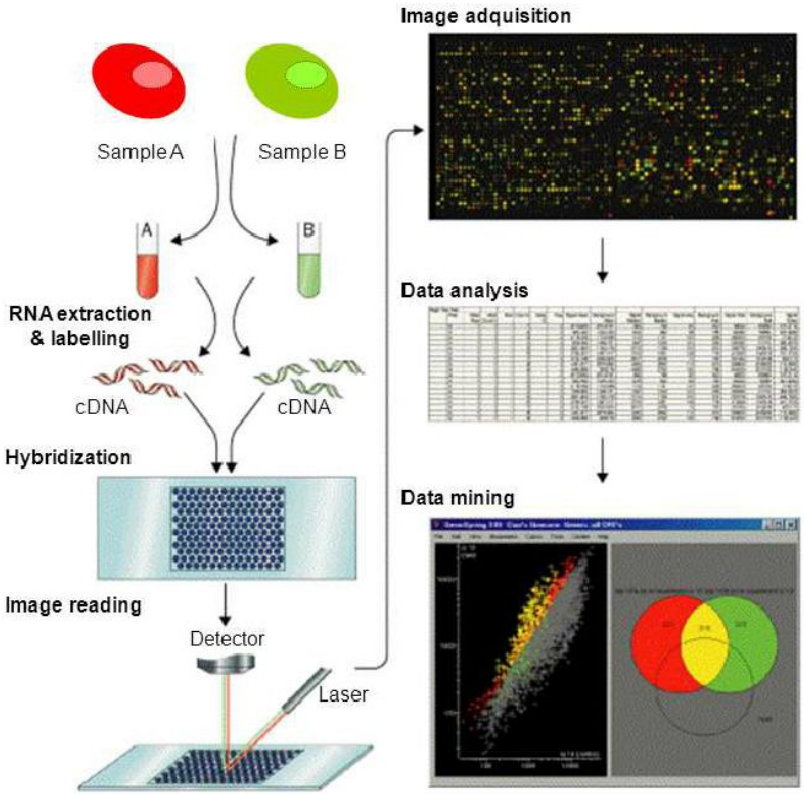


Figura 4.6: Proceso obtención intensidades con una plataforma de secuenciación

A niveles prácticos, cada archivo `.CEL` que encontramos en los experimentos de ArrayExpress representa una muestra.

El resultado para cada muestra analizada, serán parejas de valores de genes detectados y sus respectivos niveles de expresión génica [10]. Hay que mencionar, que el valor de los genes detectados no es en sí el símbolo o identificador de un gen, sino los nombres de las sondas, que representan una región del genoma destinada a identificar la expresión de un gen. Este hecho, presenta un nuevo reto programático debido a los datos de entrada que requiere el algoritmo TriGen, más adelante se explicará cómo ha sido abordado.

ARN-Seq

El otro gran avance a destacar es el uso de ARN-Seq, una técnica que utiliza tecnologías de secuenciación profunda para transformar una población de ARN (total o fraccionado) en una biblioteca de ADNc con adaptadores [11]. Posteriormente, se secuencia empleando técnicas de secuenciación de alto rendimiento. Los fragmentos obtenidos se alinean a un genoma o se ensamblan de nuevo para construir un mapa genómico, definiendo tanto la estructura como el nivel de expresión de cada gen.

Entre sus ventajas, ARN-Seq no se limita a secuencias conocidas ya que permite identificar con precisión transcripciones nuevas y los límites de la transcripción a nivel de una sola base. A diferencia de los microarrays, no son

sensibles a genes poco expresados ni dependen de anotaciones anteriores del genoma; su análisis bioinformático requiere una capacidad computacional mucho más grande y costosa, además de un flujo de trabajo más elaborado.

5. ArrayExpress

La colección de datos ArrayExpress fue desarrollada por el EML-EBI (European Bioinformatics Institute) en el año 2002, con el objetivo de cubrir una demanda: la necesidad de almacenar y compartir datos de microarrays de expresión génica. Para la estandarización de estos datos se utilizó, en primer lugar, el referente MIAME (Minimum Information About a Microarray Experiment), útil para datos de experimentos de microarrays. Más adelante, para incluir las nuevas tecnologías NGS se adaptó al estándar MINSEQE. MIAME y MINSEQE se diferencian, a grandes rasgos en el enfoque de experimento para el que fueron creados y el tipo archivos para la representación de estos(el primero utiliza matrices tabulares de expresión mientras que el segundo matrices de conteo).

En el año 2021, se realizó un importante cambio en el formato de representación de sus experimentos debido a su integración en BioStudies, una plataforma más completa y sostenible que alberga experimentos multimodales, es decir, este es un punto de acceso a los experimentos aunque estos se encuentren almacenados en otras bases más específicas como ArrayExpress u otras colecciones como BioImage Archive, Empiar, Europe Pubmed Central…

En concreto, ArrayExpress contiene datos de genómica funcional, ordenando esta información en experimentos. La mayoría de estos experimentos estudian la expresión génica basándose en el análisis de microarrays y secuencias de alto rendimiento(Next Generation Sequencing), facilitando la generación de gran cantidad de datos rápidamente.

Para entender la información que recoge, debemos aplicar los conocimientos introducidos en la sección anterior relacionados con el campo de las ciencias ómicas y todas las tecnologías e innovaciones que se utilizan en el ámbito Bioinformático actualmente.

Figura 5.1: Logo de ArrayExpress

5.1. Experimentos

En ArrayExpress, la unidad básica de organización de los datos son los experimentos. Según la Real Academia Española se define este concepto como la operación destinada a descubrir, comprobar o demostrar determinados fenómenos o principios científicos. Para identificar cada experimento o operación, se le asigna un numero de acceso tras ser importado. Debido a la integración en BioStudies, la nomenclatura ayuda tanto a rastreaer el origen de los datos como a facilitar la interoperabilidad entre plataformas. Es interesante detallar la estructura del esquema[12] que siguen estos códigos ya que va a ser la forma de realizar la consulta para acceder a cualquier experimento desde nuestra herramienta. El esquema, de forma general, sigue la estructura que mostramos a continuación:

X-YYYY-ZZZZ

Estos 3 campos corresponden:

- **X:** Letra que indica el tipo de recurso. Toma el valor `E` si se trata de un experimento, y `A` si es una plataforma de microarrays.
- **YYY:** Código que indica la procedencia o tipo de registro. Por ejemplo, toma el valor `GEOD` si se trata de un experimento importado desde la base de datos GEO (Gene Expression Omnibus), o `MTAB` si fue cargado directamente en ArrayExpress.
- **ZZZZ:** Número identificador único dentro de su origen.

Debido a la integración en BioStudies, la nomenclatura ayuda tanto a rastreaer el origen de los datos como a facilitar la interoperabilidad entre plataformas.

Una vez hemos visto la manera en la que se diferencian los experimentos entre sí, debemos estudiar que información es accesible para el procesamiento programático de cada uno de ellos.

5.1.1. Metadatos y datos

Cada experimento cuenta con dos hojas dónde se detallan los metadatos del mismo. Esto se conoce como documento MAGE-TAB y cuenta con un formato estandarizado, formado por distintas "hojas" que , a efectos prácticos, son archivos de texto separados por tabulaciones:

- En la primera se describe el experimento en sí y se llama **IDF(Investigation Description Format)**, se recoge información como el número de acceso y título, descripción, contacto del publicador, protocolos utilizados, información sobre la publicación y las variables experimentales.

- La segunda de ellas describe las características de la muestra y estímulos a la que se le ha sometido relacionándolo con cada archivo de los datos. A esta se llama **SDRF(Sample and Data Relationship Format).**

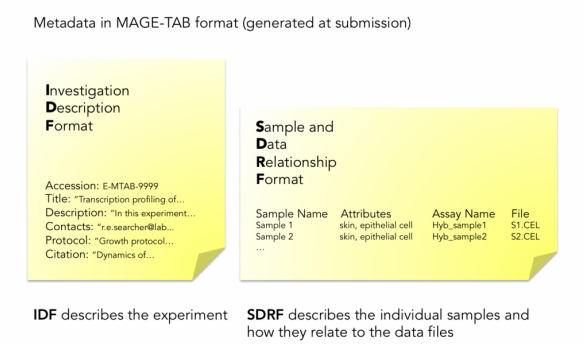


Figura 5.2: Hojas IDF y SDRF

- El documento **ADF(Array Design Format)**, describe el diseño de un microarray. Contiene dos secciones principales: metadatos con información sobre el microarray; datos de las sondas, lista de sondas del microarray con detalles como identificador, secuencias y posición de la matriz. Este documento nos ayudará a mapear(relacionar) las señales de expresión y los genes correspondientes.

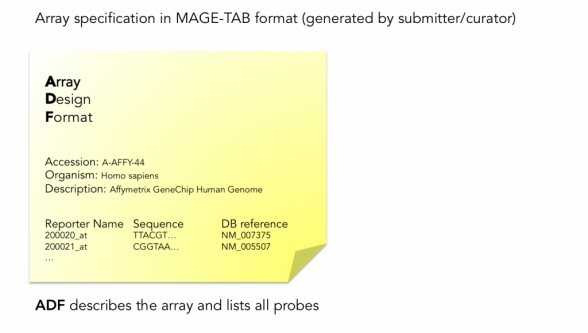


Figura 5.3: Hoja ADF

Respecto los datos en bruto de cada experimento, el documento MAGE-TAB contiene tres tipos más de archivos, distinguimos:

- Los archivos de datos sin procesar en distintos formatos según la tecnología utilizada(`.CEL` o `.GPR` para microarrays y `.FASTQ` o `.BAM` para RNA-Seq). Esta información muestra la intensidad de fluorescencia detectada por cada sonda del microarray o plataforma de secuenciación. Nos centraremos en la tecnología microarray. Cada archivo ".CEL" almacena: información de la imagen del microarray; intensidad de fluorescencia medidas en cada sonda; datos de calidad del escaneo. Cada uno de estos archivos representa los resultados de los experimentos realizados en una muestra específica y podemos descargarlos fácilmente en la sección derecha azul de la información específica de cada experimento

- Los archivos de datos procesados en formato `.txt` que contienen datos de expresión génica ya normalizados y listos para el análisis e incluyen: archivos `.txt` de tablas con niveles de expresión por gen/transcrito; archivos de datos normalizados según metodologías específicas; datos de regiones genómicas procesadas. No todos los experimentos contienen estos archivos, el buscador nos permite filtrar los que si los contienen.

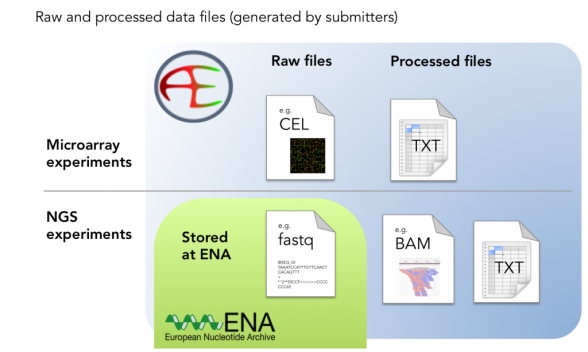


Figura 5.4: Datos procesados y sin procesar

6. El algoritmo TriGen

Para explotar los datos genómicos obtenidos, utilizaremos TriGen, un algoritmo capaz de detectar triclusteres en experimentos *"Time Series Design"*. Esta metodología innovadora fue presentada por David Gutiérrez Avilés, Cristina Rubio Escudero, Francisco Martínez Álvarez y José C. Riquelme y publicada en la revista científica Neurocomputing en el año 2014.

El objetivo principal de este algoritmo es identificar patrones de expresión génica que varían a lo largo del tiempo bajo distintas condiciones experimentales basandose en el uso de algoritmos genéticos e implementando el concepto de triclustering.

A continuación, introducimos los fundamentos en los que se basa para asimilar los resultados que obtendremos y poder aplicar el preprocesado correcto a nuestros datos de entrada.

6.1. Fundamentos

6.1.1. Experimentos Time Series Design

Este concepto se desarrolló originalmente en el contexto de las ciencias sociales [13], permite no solo registrar los cambios, sino también distinguir entre variaciones reales y fluctuaciones aleatorias (ruido) en los datos.

En bioinformática este tipo de experimentos son de gran utilidad, ya que los datos ómicos suelen ser ruidosos y de alta dimensionalidad. Su finalidad se basa en analizar la dinámica de los procesos biológicos mediante la medición repetida de la expresión génica a lo largo del tiempo. Este enfoque permite conocer la respuesta ante ciertos estímulos del transcriptoma, permitiendo identificar patrones de expresión en función del tiempo.

6.1.2. Triclustering

Para llegar a entender el triclustering, antes debemos definir en que consiste el clustering. Se define como:

"El proceso de agrupar objetos de datos en un conjunto de clases disjuntas, llamadas clústeres, de modo que los objetos dentro de una misma clase tengan alta similitud entre sí, mientras que los objetos en clases distintas sean altamente disímiles"

Actualmente, los enfoques basados en clustering aplicados a datos de expresión génica se realizaban de tres formas: mediante clustering basado en genes(genes considerados objetos y muestras caracteristicas), clustering basado en muestras(muestras son objetos y genes características) y el clustering basado en subespacios(genes y muestras se tratan de la misma forma, este último asume que solo un subconjunto de genes participa en un proceso celular determinado, y dicho proceso celular solo ocurre en un subconjunto de las muestras. Estas técnicas revelan estructuras naturales y ciertos patrones[14].

Aunque estas técnicas revelan tanto estructuras naturales como ciertos patrones y son de gran utilidad en este campo, debido a la gran dimensionalidad que generan los experimentos de microarrays es necesario utilizar otros algoritmos para extraer patrones coherentes y ocultos.

Es por ello, que el triclustering ofrece una mejora ya que estas técnicas son capaces de hayar las relaciones ocultas y coherentes características de ciertos procesos celulares. En este contexto, el comportamiento coherente de los genes se refiere a: La expresión similar entre sí cuando se observan bajo un subconjunto específico de condiciones experimentales y durante un segmento concreto del tiempo. Dicho de otro modo, aunque estos genes no necesariamente se comporten de forma similar en todo el experimento, existe un intervalo temporal y un conjunto de muestras en el que sus niveles de expresión varían de forma paralela o correlacionada, lo cual puede ser indicativo de una co-regulación biológica, participación conjunta en una misma vía funcional o respuesta común ante un estímulo determinado.

Una vez explicado el por qué de su uso y el avance que supone, diremos que el triclustering es un enfoque avanzado de mineria de datos aplicado al análisis de expresión génica que permite identificar patrones entre subconjuntos de genes, condiciones(muestras) y puntos temporales. Es decir, a partir de una matriz tridimensional generada por los experimentos de series temporales cuyas dimensiones serán:

Genes x Muestras x Puntos temporales,

encuentra submatrices tridimensionales(triclusters) que representan comportamientos coordinados de un conjunto de genes G_D durante segementos específicos en el tiempo T_D bajo ciertas condiciones C_D.

A este formato de conjunto de datos y en este área en específico, se conoce como *GST(Gene Sample Time) microarray data*[15].

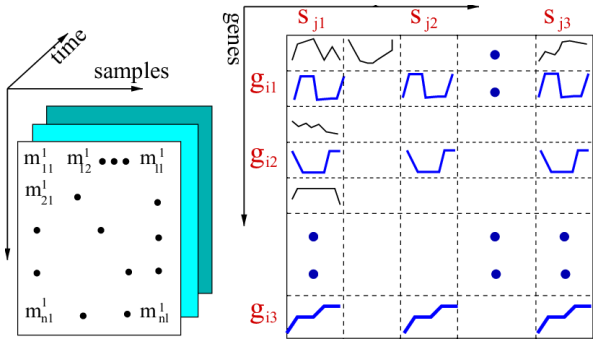


Figura 6.1: Ejemplo genérico GST microarray data

Como podemos observar, cada celda m_k del dataset representa el nivel de expresión para un gen g_i bajo cierta condición s_j en un tiempo t_k.

6.1.3. Algoritmos Genéticos

Un algoritmo genético no es mas que un proceso iterativo basado en los conceptos biológicos de selección natural y genética que busca la mejora progresiva de una población de soluciones[16]. Es por ello que este enfoque proporciona un mecanismo de búsqueda global que ofrece resoluciones potentes por su capacidad de encontrar soluciones óptimas en espacios de búsqueda grandes e irregulares(como los dataset GST) y fácil de integrar con el método de triclustering por dos motivos principalmente:

- Está basado en el uso de operadores fácilmente ajustables y sustituibles para su uso.

- No requieren información previa del dominio del problema al evaluar cada solución con una función de aptitud arbitraria.

La unidad básica de computación es el individuo o cromosoma, el cual representa una solución candidata a resolver el problema. En términos prácticos, cada individuo es una estructura codificada(cadena de bits, números, símbolos...) que contiene todos los parámetros necesarios para describir una solución posible. Por ejemplo, imaginemos un problema modelable con 3 parámetros y cada uno a su vez, con dos valores distintos. Entonces, podriamos representar cada parametro con 1 bit, donde el bit indica cada valor posible para ese parámetro.

Ejemplo de cromosoma: 1 0 1

Estas unidades se modifican durante el proceso de búsqueda de soluciones hasta encontrar la óptima mediante las siguientes 3 operaciones u operadores:

- **Operador de selección:** Consiste en elegir, utilizando una función de *fitness*, de entre todos los individuos aquellos que serán ”padres” en la próxima generación.

- **Operador de cruce:** Consiste en combinar la información genética de dos ”padres”para generar uno o mas hijos, que serán nuevos cromosomas. La combinación se produce mediante la elección aleatoria de uno o varios puntos de cruce y seguidamente el intercambio de segmentos entre los padres. Por ejemplo, si suponemos el punto de corte entre el primer y segundo bit, el resultado sería el siguiente:

  Ejemplo de padre I: 1 | 1 1
  Ejemplo de padre II: 0 | 0 0
  Hijo I: 1 | 0 0
  Hijo II: 0 | 1 1

- **Operador de mutación:** Consiste en la alteración aleatoria de parte de un individuo con el fin de mantener la diversidad(se introducen nuevas características no presentes en los padres) y así evitar el estancamiento.Por ejemplo, si aplicamos este operador al *Hijo I* del ejemplo anterior en su último bit obtenemos:

  Hijo I: 1 0 ***1***

Estas tres operaciones simulan el proceso de evolución natural y permiten explorar y explotar todo el espacio de búsqueda.

Por último, nos falta mencionar el uso de la función *fitness* para evaluar como de buena es cada solución dentro de la población de cromosomas y así poder dar saltos generacionales. Está función no es mas que una fórmula matemática que da valor único a cada cromosoma y cuanto mejor sea este valor para la función objetivo a optimizar, mayor posibilidad de reproducirse”(ser escogido por el operador de selección) tendrá.

6.2. Metodología del algoritmo

En la siguiente sección, veremos como aplica estos conceptos el algoritmo *TriGen* para encontrar triclústeres significativos. Para ello, vamos a basarnos en el siguiente semicódigo que representa el funcionamiento:

```
Input:  Dataset D

Output:  Set of tricluster solutions

Begin TriGen algorithm
   Repeat for each tricluster solution
        Generate initial population
        Evaluate population
        Repeat for number of generations
            Select population
            Cross population
            Mutate population
            Evaluate population
        End Repeat
        Select best individual
        Add to the solution set
    End Repeat
 End Trigen algorithm
```

Figura 6.2: Pseudo-código genérico del algoritmo

El primer paso consisten en la **codificación de los individuos**. Cada individuo, representa un Tricluster TC con la estructura explicada en la sección anterior. Por ejemplo, un posible tricluster candidato sería formalmente:

$$\textbf{Posible tricluster } TC\text{:} \quad G_D = \{g_1, g_2\}, \quad C_D = \{c_3, c_4, c_5\}, \quad T_D = \{t_{10}\}$$

$$\text{donde } g_1, g_2 \in G, \quad c_3, c_4, c_5 \in C, \quad t_{10} \in T \quad \text{y} \quad T \times C \times G = D$$

Tras iniciar una población aleatoria de individuos, a continuación, se evalua la población. Para ello se utilizan, las siguientes dos funciones *fitness* que introducimos de la siguiente forma:

$$f_{MSR}(TC) = MSR - PESOS - DISTINCION$$

Esta función consiste en la adaptación en tres dimensiones de la métrica *Mean Squared Residue(Residuo Cuadrático Medio*, tipicamente utilizada en análisis bidimensionales de expresión genética; un conjunto de *PESOS* para el número de genes, condiciones y tiempos a añadir en el tricluster *TC*; métrica *DISTINCIÓN* del inividuo que se está evaluando respecto a los triclusteres previamente encontrados. Su objetivo será medir la coherencia del comportamiento de los genes entre triclusteres.

$$f_{LSL}(TC) = LSL - PESOS - DISTINCION$$

La segunda de ella comparte los términos *PESOS* y *DISTINCIÓN* y se diferencia en el uso de la métrica *Least Squares Line(Línea de Mínimos Cuadrados)*. Esta función se encargará de evalúar si los genes de un tricluster siguen una misma tendencia a lo largo del tiempo.

Para una descripción formal de cada fórmula y término, se puede revisar el articulo [17].

Una vez evaluados los individuos, se aplican los operadores adaptados a nuestro análisis:

- **Operador de mutación:** Cada individuo puede ser mutado con una probabilidad p_m(se otorga como parámetro), se le aplica una acción de entre las seis siguientes: añadir nuevo gen a G_D, añadir nueva condición a C_D , añadir nuevo punto temporal a T_D, eliminar gen a G_D, eliminar condición a C_D, eliminar punto temporal a T_D.

- **Operador de selección:** Cada individuo puede ser seleccionado para reproducirse en función del valor *fitness* mediante el método de selección por ruleta, es decir, cuanto mayor es el *fitness*, mayor es su porción en la ruleta. Este método favorece la diversidad entre las soluciones(todas tienen posibilidad de reproducción) y a su vez, el avance hacia las mejores.

- **Operador de Cruce:** Cada individuo sufre un intercambio entre parte de las listas de genes, condiciones y tiempos entre los dos padres usando un punto de corte aleatorio.

Estos pasos de generación, evaluación y modificación de individuos se repiten iterativamente hasta encontrar un subconjunto, de tamaño escogido como parámetro, de triclusteres óptimos.

6.3. Datos de Entrada. Recursos consumidos.

Una vez comprendida la metodología del algoritmo y consultado su manual, podemos establecer con claridad el formato y la estructura de los archivos de entrada que requiere TriGen. Estos archivos deben representar explícitamente las tres dimensiones simétricas sobre las que funciona el algoritmo: genes, condiciones experimentales y puntos temporales. Cada una de estas dimensiones debe estar correctamente definida y estructurada para que TriGen pueda construir y analizar la matriz tridimensional de expresión génica.

A continuación, se describe en detalle cómo debe estructurarse cada una de estas dimensiones dentro de los archivos de entrada.

Genes

La primera dimensión está definida por los genes. Se representa mediante un archivo en formato `.txt` que contiene una lista simple de nombres o símbolos de genes, con una entrada por línea. Cada línea indica un gen cuyos valores de expresión serán evaluados en distintas muestras. Este archivo determina el número de filas de la matriz tridimensional. Aunque algunos genes puedan repetirse en el dataset original, esta lista no debe contener duplicados.

Puntos temporales

La segunda dimensión corresponde a los puntos temporales en los que se ha tomado cada medida de expresión. También se representa mediante un archivo `.txt` en forma de lista simple, donde cada línea contiene un punto de tiempo con su respectiva unidad (por ejemplo: 3 day, 6 hour, etc.). Este archivo define la profundidad del conjunto de datos tridimensional.

Condiciones experimentales

La tercera dimensión está compuesta por los condiciones experimentales asociadas a cada muestra del experimento. Esta dimensión se representa mediante dos tipos de archivos:

- Un archivo `.txt` con una lista simple que contiene los nombres de las condiciones experimentales aplicadas a las muestras.

- Un conjunto de archivos `.csv`, uno por cada punto temporal, que contienen las matrices de expresión génica. En cada matriz, las filas representan los genes (en el mismo orden que en el vector de Genes) y las columnas representan las condiciones experimentales.

Es fundamental que los tres conjuntos de archivos sean simétricos y estén perfectamente alineados.A nivel programático, esto significa que:

1. El orden de los genes en Genes.txt debe coincidir exactamente con las filas de cada archivo `.csv`. Así como la longitud del vector de genes debe ser igual al numero de filas de cada archivo `.csv`.

2. El orden de las condiciones en `Samples.txt` debe coincidir con las columnas de cada archivo `.csv`.

3. El orden de los archivos `.csv` debe corresponder al orden de los tiempos en `Times.txt`.

4. Los archivos `.csv` deben tener el mismo número de genes(filas) y condiciones(columnas) entre sí y no pueden deben contener celdas vacías.

Además de los archivos de entradas mencionados, el algoritmo TriGen requiere dos archivos adicionales :

- El primero de ellos es llamado ***Resources.xml***, actúa como un índice estructurado de los datasets disponibles. Este archivo en formato XML contiene entradas `<dataset>`, cada una de las cuales describe un conjunto de datos tridimensional que se desea analizar. Cada entrada contiene los metadatos, así como referencias a los archivos necesarios para la ejecución del algoritmo. A continuación, se muestra un ejemplo de entrada:

```
<dataset description="Cultured epidermal keratinocytes treated with OsM
    1, 4, 24 &amp;amp; 48hrs, and Skinethic epidermal substitutes treated 1,
    4, 24, 48h &amp;amp; 7days, each with untreated control Experiment Overall
     Design: Each timepoint has treated/untreated matched control.
    Experiment Overall Design: Skinethic 4h treated was replicated.
    Experiment Overall Design: Note: OncostatinM treatment, GSM61488, is
    the 1h control!!" geneSize="12625" id="E-GEOD-2822" maxC="4" maxG="150
    " maxT="3" minC="2" minG="2" minT="2" name="E-GEOD-2822" organism="
    Human" sampleSize="4" timeSize="3" type="b">
     <resources separator=";">
       <resource>exprs_time_1.csv</resource>
       <resource>exprs_time_24.csv</resource>
       <resource>exprs_time_48.csv</resource>
     </resources>
     <genes>genes.txt</genes>
     <samples>samples.txt</samples>
     <times>times.txt</times>
   </dataset>
```

Extracto de código 6.1: Entrada para eperimento E-GEOD-2822.

- El segundo de ellos es un archivo de configuración llamado **Config_file.tricfg** que contiene los parámetros de ejecución para un experimento determinado. Entre estos parametros se encuentran: el nombre de la entrada `<dataset>` del archivo *Resources.xml* sobre la que se lanzara el algoritmo, la ruta donde se almacenarán los resultados generados, etc. Este archivo permite reproducir TriGen, definiendo todos los parámetros necesarios sin modificar el código. Esto facilita su uso tanto desde la consola como desde scripts en R.

```
dataset = E-GEOD-2822
out = C:\Users\alexl\Desktop\TFG/TrLab5/OutputTriGen/E-GEOD-2822/E-GEOD-2
    822
fitness = msl
N = 3
G = 500
I = 200
Ale = 0.5
Sel = 0.5
Mut = 0.4
Wf = 0.8
Wg = 0.0
Wc = 0.05
Wt = 0.05
WOg = 0.05
WOc = 0.03
Wot = 0.02
minG = 2
maxG = 150
minC = 2
maxC = 4
minT = 2
maxT = 3
```

Extracto de código 6.2: Archivo de configuración experimento E-GEOD-2822.

7. Análisis ontológico. PantherDB.

Una vez obtenidos los resultados del análisis con Trigen(los triclústeres solución) generados por el algoritmo, es fundamental interpretar su significado biológico. Para ello, se utiliza el análisis ontológico de genes, una técnica que permite asociar los genes identificados con funciones biológicas, procesos celulares y rutas metabólicas reconocidas.

Para ello emplearemos la herramienta PANTHER (Protein ANalysis THrough Evolutionary Relationships), una plataforma ampliamente utilizada para realizar análisis de enriquecimiento funcional y clasificación ontológica de genes que permite identificar qué funciones están sobrerrepresentadas en un conjunto de genes, lo que proporciona una visión funcional y contextualizada de los resultados obtenidos.

Figura 7.1: Logo PantherDB

7.1. Análisis ontólogico

Fuera del contexto bioinformático, este término se refiere al proceso mediante el cual se exploran, describen y organizan conceptos y sus relaciones dentro de un dominio específico. Este tipo de análisis se utiliza para comprender mejor la estructura conceptual de un área del conocimiento, facilitando así la búsqueda y razonamiento sobre la información.

Por tanto, este término aplicado a nuestro ámbito y en concreto al análisis de datos moléculares es de gran utilidad, ya que da sentido a los resultados del análisis TriGen, que sin este estudio tan solo serían listas de genes, que por si solos, no ofrecen una interpretación clara.

Para darle sentido a estas listas, existe GO(Gene Ontology), una herramienta bioinformática estándarizada que permite la vinculación de grupos de genes a categorías funcionales. Estas categorías están organizadas en tres grandes grupos:

- **Función molecular:** Describe la actividad bioquímica principal que realiza un gen o su producto (generalmente una proteína) a nivel molecular. Se refiere a lo que hace una molécula dentro de la célula, no al proceso en el que participa ni a su localización. Por ejemplo, si su función es la unión celular, actividad enzimática, actividad transportadora...

- **Proceso biológico:** Representa una serie de eventos o funciones moleculares, que llevan a cabo un objetivo biológico específico, es decir, se refiere a los procesos celulares, fisiológicos o de desarrollo en los que está involucrado un gen o proteína. Por ejemplo, si se encuentra involucrado en procesos de división celular, apoptósis, división celular...

- **Componente celular:** Describe dónde se encuentra funcionalmente un producto génico dentro de la célula o su entorno, es decir, su localización subcelular. Por ejemplo, si se encuentra en el núcleo, ribosomas, mitocondrias...

Esta clasificación permite comprobar que funciones están sobrerrepresentadas en un conjunto de genes. Existen varias plataformas bioinformáticas que integran esta ontología en sus herramientas de interpretación, por ejemplo DAVID,Enrichr o PantherDB [18]. En la siguiente sección, describiremos esta última y argumentaremos por qué ha sido la escogida.

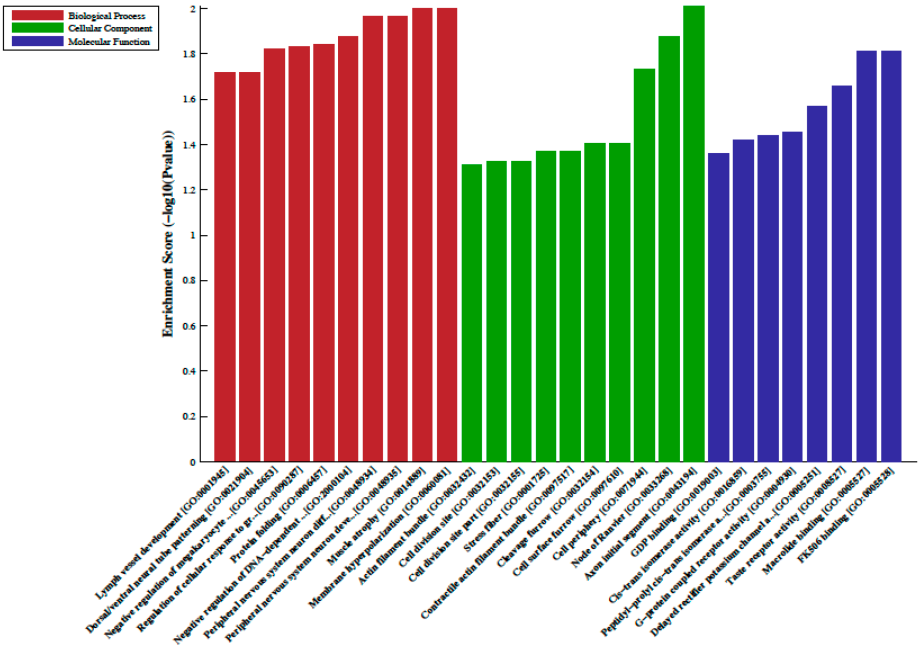


Figura 7.2: Clasificación GO y términos asociados.

7.2. PantherDB

Esta plataforma fue desarrollada por SRI International a principios de los 2000 con el objetivo de predecir funciones biológicas en genes poco caracterizados mediante el principio de homología(similitud de estructuras o características entre diferentes organismos). Actualmente, se encuentra completamente integrada a los estándares de GO y ha evolucionado hasta convertirse en una herramienta clave para el análisis funcional de genes y la interpretación de datos ómicos, esto es debido a:

- Alta actualización y respaldo científico.
- Fácil accesibilidad tanto vía web como API.
- Disponibilidad de pathways propios(interacciones moléculares que caracterizan funciones específicas) comprobados.
- Combinación de los estándares GO con estructuras evolutivas(agrupaciones según la historia evolutiva)
- La base de datos no sólo abarca el genoma humano o de ratón sino mas de 140 especies.

Como vemos, estas ventajas hacen que PantherDB sea una de las herramientas mas fiables y potentes tanto por su gran precisión y cobertura, el fácil acceso programático y la válidez científica que le respalda. A continuación, veremos cómo realiza el proceso de análisis funcional.

7.2.1. Análisis de enriquecimiento funcional

Está técnica estadística permite la interpretación de grandes grupos de genes. Su objetivo principal consiste en determinar si algunas funciones biológicas, procesos o rutas metabólicas se encuentran sobrerrepresentados en comparación con lo que se podría esperar al azar. Este enfoque se basa en que la interconectividad de los procesos biológicos, es decir, los genes no actúan de manera aislada.

Existen varios métodos para realizar este análisis, los mas relevantes son: el **análisis de sobrerrepresentación (ORA)** que evalúa si ciertos términos funcionales están presentes con mayor frecuencia en el conjunto de genes de interés que en un conjunto de referencia; el **análisis de enriquecimiento de conjuntos de genes (GSEA)** que considera todos los genes en un experimento, clasificándolos según su expresión y determinando si los miembros de un conjunto de genes específico se encuentran en la parte superior o inferior de esta clasificación y el **análisis basado**

en topología de rutas que tiene en cuenta la estructura de las rutas biológicas, considerando las interacciones y posiciones de los genes dentro de estas rutas para evaluar su enriquecimiento.

PantherDB emplea el método ORA para sacar conclusiones y lo hace siguiendo estos pasos:

1. Se proporciona una lista de genes (por ejemplo, de un triclúster) y se especifica el organismo.

2. Asocia esos genes con términos de Gene Ontology(GO) y pathways.

3. Compara la frecuencia de aparición de cada término en la lista de genes frente a su frecuencia en el fondo genómico, es decir, el genoma del organismo.

4. Cálcula la significancia estadística de cada término mediante la (método para comprobar la asociación entre dos variables cualitativas).

5. Cálcula la significancia estadística de cada término mediante la *prueba de Fisher*(método para comprobar la asociación entre dos variables cualitativas).

6. Ajuste mediante una tasa para evitar falsos positivos.

Véamos un ejemplo de uso para comprender que conclusiones podemos sacar:

"Se realiza un experimento en células de expresión génica en células humanas para el estudio de un tipo de cáncer y se obtiene una lista de 200 genes. Tras introducir esta lista y el organismo de estudio en PantherDB, este realiza un análisis de enriquecimiento funcional que nos muestra que los procesos biológicos de proliferación célular, apoptosis y reparación de ADN se encuentran sobrerrepresentados en la lista de genes, es decir, cada proceso o término esta vinculado a mas genes de nuestra lista de lo esperado en referencia al genoma humano."

De esta forma podremos comprobar la coherencia biológica de nuestro experimento, validar los resultados obtenidos y generar nuevas hipótesis basadas en los resultados. Estas conclusiones serán de gran ayuda en la búsqueda de dianas terapéuticas o biomarcadores del tipo de cáncer estudiado.

8. Implementación de la herramienta

Una vez abordados los fundamentos teóricos y técnicos en las secciones anteriores(contexto de las ciencias ómicas, descripción del repositorio ArrayExpress, el funcionamiento del algoritmo TriGen y la interpretación de los resultados con PantherDB), estamos en disposición de comenzar con el desarrollo de una herramienta que automatice el procesamiento de experimentos transcriptómicos para su posterior análisis con TriGen. El objetivo de esta herramienta es facilitar la extracción, transformación y preparación de los datos de expresión génica disponibles en ArrayExpress, generando de forma estructurada y alineada los archivos de entrada que el algoritmo necesita. Además se han integrado funcionalidades de interpretación de resultados con una herramienta, que no había sido utilizada previamente para el análisis de triclústeres, esta es *PantherDB*.

En esta sección se detalla el proceso de implementación: desde la obtención de los datos hasta la generación de los recursos formateados, describiendo tanto la lógica del código como las decisiones técnicas adoptadas. Así como la creación de una ShinyApp, haciendola aún mas completa con una interfaz gráfica que facilita el uso de la herramienta.

8.1. Retos y limitaciones adoptadas.

Durante el desarrollo han surgido una serie de retos programáticos asociados a la heterogeneidad de los datos almacenados en ArrayExpress, lo que ha requerido establecer reglas de filtrado y limitaciones para garantizar la automatización del proceso. Algunos de estos retos han sido:

- **Falta de estandarización en los metadatos:** Los metadatos de los experimentos (archvos `IDF` y `SDRF`) presentan estructuras muy variables entre estudios, lo que dificulta su procesamiento automático de forma generalizable.

- **Experimentos incompletos:** Muchos experimentos carecen de los archivos `.CEL` (datos crudos de microarrays), lo cual impide construir las matrices de expresión desde cero.

Para asegurar que los experimentos seleccionados sean compatibles con el algoritmo, el preprocesado se ha desarrollado para ciertos experimentos de ”prueba”que tienen en común ciertos rasgos:

- Son experimentos etiquetados como ”Time Series Design”
- Son experimentos que contienen Raw Data Available”.
- Los metadatos muestran específicamente: una columna para el organismo biológico de estudio y otra los puntos temporales en el archivo *.SDRF*; una columna que muestre los SCANs utilizados para diferenciar cada muestra.

Se ha validado el funcionamiento de la herramienta con cinco experimentos distintos cuyo identificador comienza por E-GEOD-XXXX, lo que indica que han sido importados desde el repositorio GEO y contienen los archivos `.CEL` necesarios. En todos estos casos, el proceso de preprocesamiento automático han funcionado correctamente.

Durante el desarrollo, se ha tratado de generalizar al máximo la lógica de automatización, de forma que el proceso sea escalable y adaptable a nuevos experimentos sin necesidad de ajustes manuales. Por tanto, se puede afirmar que si un experimento cumple con todas las reglas y condiciones previamente especificadas, es capaz de procesarlo correctamente.

8.2. Herramientas utilizadas

Para el procesamiento y análisis de los datos en R, se han utilizado distintas librerías y paquetes, principalmente integrados en Bioconductor. En esta sección se describirán para detallar su utilidad en el flujo.

8.2.1. BioConductor

Bioconductor es un proyecto ”open source”, lanzado en el año 2001 por Robert Gentleman y Jeffrey Leek, el cual proporciona un marco de trabajo con gran cantidad de herramientas y documentación para el análisis de datos de microarrays a través del lenguaje de programación R[19]. Desde entonces, la plataforma se ha ido adaptando a las tecnologías mas recientes haciendola la mas divulgada y utilizada entre bioinformáticos. Nos será de gran utilidad durante el desarrollo de código debido a: su amplia gama de paquetes especializados que permiten tanto el acceso a datos públicos(como los de ArrayExpress), el manejo de estructuras complejas(como *AffyBatch Objects*[20] o la anotación de datos transcriptómicos(como el paquete *AnnotationDBI*[21]). A continuación, veremos en profundidad los que mas se han utilizado durante la etapa de implementación.

Figura 8.1: Logo de BioConductor

ArrayExpress

Este paquete [22] es esencial ya que aporta distintas funciones para el acceso y descarga de los datos contenidos en los experimentos del repositorio:

- Mediante la función **`getAE()`** se permite la descarga automática de los metadatos(archivos `.IDF`, `.SDRF`y `.ADF`) y los datos asociados a un experimento(tanto los datos en crudo como los datos procesados, si se requiere), tomando como parámetro el número de acceso. Para ello, la función consulta la API de BioStudies.

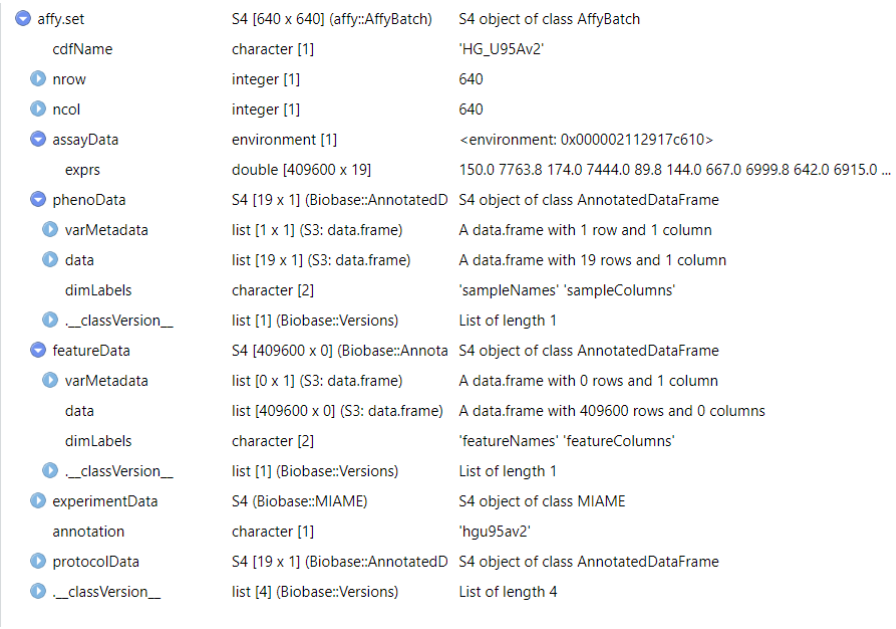
Affy

Este paquete fue disñado específicamente para el análisis de datos de microarrays Affymetrix en todos sus niveles(tanto probes como probesets). Su principal utilidad consiste en la transformación de los datos en crudo obtenidos en estructuras organizadas, como lo son los ***AffyBatch Objects***, para ello se emplean las funciones que se muestran a continuación:

- Mediante la función ***ReadAffy()***, se cargan los datos en crudo desde el directorio específicado como parámetro, construyendo una estructura que nos facilita trabajar con múltiples muestras al mismo tiempo, asociando la información del experimento coon los valores de intensidades contenidos en las distintas muestras.

AffyBatch Object

Este objeto organiza todos los elementos necesarios mediante componentes internos(o *slots*. Por ejemplo, se recogen datos como el nombre del chip utilizado, la matriz de intensidades crudas, metadatos fenotípicos(condición experimental, tiempo, tratamiento...) o información técnica sobre como se generaron los datos(fecha, scans...), etc.



affy.set	S4 [640 x 640] (affy::AffyBatch)	S4 object of class AffyBatch
cdfName	character [1]	'HG_U95Av2'
nrow	integer [1]	640
ncol	integer [1]	640
assayData	environment [1]	<environment: 0x000002112917c610>
exprs	double [409600 x 19]	150.0 7763.8 174.0 7444.0 89.8 144.0 667.0 6999.8 642.0 6915.0 ...
phenoData	S4 [19 x 1] (Biobase::AnnotatedD	S4 object of class AnnotatedDataFrame
varMetadata	list [1 x 1] (S3: data.frame)	A data.frame with 1 row and 1 column
data	list [19 x 1] (S3: data.frame)	A data.frame with 19 rows and 1 column
dimLabels	character [2]	'sampleNames' 'sampleColumns'
.__classVersion__	list [1] (Biobase::Versions)	List of length 1
featureData	S4 [409600 x 0] (Biobase::Annota	S4 object of class AnnotatedDataFrame
varMetadata	list [0 x 1] (S3: data.frame)	A data.frame with 0 rows and 1 column
data	list [409600 x 0] (S3: data.frame)	A data.frame with 409600 rows and 0 columns
dimLabels	character [2]	'featureNames' 'featureColumns'
.__classVersion__	list [1] (Biobase::Versions)	List of length 1
experimentData	S4 (Biobase::MIAME)	S4 object of class MIAME
annotation	character [1]	'hgu95av2'
protocolData	S4 [19 x 1] (Biobase::AnnotatedD	S4 object of class AnnotatedDataFrame
.__classVersion__	list [4] (Biobase::Versions)	List of length 4

Figura 8.2: Estructura AffyBatch Object del experimento E-GEOD-2822

Las matrices de intensidades contenidas en AffyBatch contienen los datos a nivel de sondas individuales(probes), es decir, cada fila de esta matriz representa una sonda específica del array, no un gen completo. Por lo tanto, debemos hacer una agrupación en conjuntos de sondas(probesets), estos si se podrán relacionar con genes anotados. Para ello, se aplicara el proceso de congregación y corrección que se explica a continuación.

RMA function

Esta técnica fue propuesta por Rafael Izarry en 2003 [23] como alternativa a otros métodos anteriores(*MAS 5.0* o *AvDiff*), mediante un proceso de trres etapas que se detalla a continuación:

1. **Corrección de fondo** donde se ajustan las intensidades de cada sonda para eliminar el ruido de fondo a partir de una distribución logarítmica de los valores de la sondas *Mismatch*(sondas de control con bases cambiadas respecto a la secuencia real del gen).

2. **Normalización entre arrays** mediante normalización por cuantiles asegurando distribuciones de intensidade sidénticas en todas las muestras, eliminando variabilidad por errores técnicos.

3. **Sumarización** donde se cálcula un valor único para cada gen representado por un conjunto de sondas mediante un modelo aditivo lineal ajustado para reducir el impacto de valores atípicos(*outliers*).

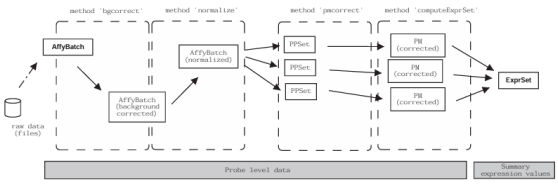


Figura 8.3: Esquema método RMA

AnnotationDBI(mapIds)

Este paquete permite el acceso a paquetes de anotación biológicas, estructuras organizadas para poder vincular identificadores de sondas a sus respectivos genes. Estos paquetes pueden estar relacionados específicamente a orgaismos, chips, ontologías o genomas. En nuestro caso, los metadatos de los experimentos(en concreto el archivo `.ADF`) nos da acceso a los paquetes de anotación asociados a los chips, por lo que son los que utilizaremos. Los reconoceremos por acabar en `.db`. Para el mapeo y extracción de genes anotados, se ofrecen distintos métodos útiles, nosotros utlizaremos el siguiente:

- El método **`mapIds()`** con el cual extraemos una sola columna de anotación por conjunto de identificadores. Para el uso de este método, se debe especificar el paquete de anotación objetivos, un vector de identificadores, la información que deseamos obtener y el tipo de claves que estamos usando(en nuestro caso será "PROBEID", es decir, identificadores de sondas).

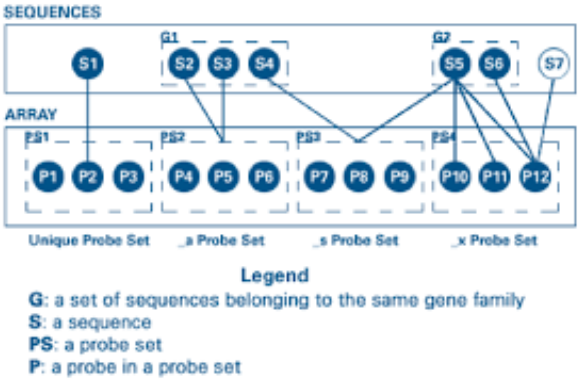


Figura 8.4: Proceso mapeo de genes desde nivel probes.

De este modo, conseguimos el vector de genes que el algoritmo TriGen utilizara como dato de entrada.

8.2.2. Otros paquetes

Además de los paquetes bioinformáticos específicos descritos previamente, el desarrollo de código ha requerido el uso de otros paquetes que complementan funcionalidades clave en distintas etapas del proceso. Por ejemplo, **`httr`** y **`jsonlite`** han sido utilizados para interactuar con APIs REST y procesar respuestas en formato JSON, mientras que **`xml2, XML y RCurl`** han facilitado la lectura, escritura y manipulación de archivos `XML`. Por otro lado, **`dbplyr, tidyr y biomaRt`** han sido fundamentales para la gestión, transformación y anotación de datos de forma eficiente. Por último, para la visualización y exploración de resultados, se han empleado paquetes como **`ggplot2 y plotly`**, que permiten generar gráficos,perimitiendo así la interpretación y validación de los datos procesados.

8.3. Preprocesamiento y ejecución del algoritmo

En esta sección se detalla el proceso de desarrollo de código llevado a cabo para implementar tres bloques principales: el acceso y descarga de los datos de experimentos desde el repositorio ArrayExpress, el preprocesamiento y transformación de los datos descargados con el fin de generar los archivos con la estructura que el algoritmo TriGen requiere como datos de entrada y por último, la ejecución automática del algoritmo TriGen. Durante todo el flujo, se ha trabajado tan sólo con el código de acceso del experimento como parámetro, con la finalidad de integrar estas funciones fácilmente en la ShinyApp, teniendo que introducir tan solo dicho identificador para el uso. Cada uno de estos bloques ha sido desarrollado en el lenguaje de programación R por la razones argumentadas en la sección anterior. A continuación, detallaremos en profundidad cada función de código desarrollado.

8.3.1. Descarga de datos

Como se ha introducido previamente, el primero de los bloques funcionales consiste: en el acceso y descarga de los datos dado un identificador de acceso y en el almacenamiento en una carpeta local ordenada y agrupada para futuros pasos. Para ello, se ha desarrollado la función `downloadData()`.

```
downloadData <- function(accesion_number, base_dir) {
  # Construccion directorio local para almacenamiento
  datos_dir <- file.path(base_dir, "DatosExp")
  if (!dir.exists(datos_dir)) {
    dir.create(datos_dir, recursive = TRUE)
  }
```

```r

  # Creacion del directorio en caso de no haberse creado previamente
  path_folder <- file.path(datos_dir, accesion_number)
  if (dir.exists(path_folder)) {
    archivos_existentes <- list.files(path_folder, recursive = TRUE)
    if (length(archivos_existentes) > 0) {
      message("El experimento ya ha sido descargado previamente.")
      return("already_downloaded")
    }
  } else {
    dir.create(path_folder, recursive = TRUE)
  }

  # Acceso al experimento y posterior descarga
  # Uso funcion auxiliar en caso de error.
  experiment <- tryCatch({
    getAE(accesion_number, type = "full", path = path_folder)
    path_folder
  }, error = function(e) {
    message("Error en la descarga: ", e$message)
    downloadFromFTP(accesion_number, dest_dir = datos_dir)
  })

  return(experiment)
}

downloadFromFTP <- function(accession, dest_dir, unzip_files = TRUE) {
  ftp_base  <- "ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/
   experiment"
  exp_type  <- toupper(sub("(^E-)([^-]+).*", "\\2", accession))
  ftp_url   <- file.path(ftp_base, exp_type, accession)
  local_dir <- file.path(dest_dir, accession)
  if (!dir.exists(local_dir)) dir.create(local_dir, recursive = TRUE)

  zip_filename   <- paste0(accession, ".raw.1.zip")
  zip_url        <- file.path(ftp_url, zip_filename)
  local_zip_path <- file.path(local_dir, zip_filename)

  tryCatch({
    download.file(zip_url, local_zip_path, mode = "wb")
    if (unzip_files) unzip(local_zip_path, exdir = local_dir)
  }, error = function(e) {
    message("No se pudo descargar ZIP: ", e$message)
  })

  for (ext in c(".sdrf.txt", ".idf.txt", ".adf.txt")) {
    fname <- paste0(accession, ext)
```

```
    url   <- file.path(ftp_url, fname)
    dest  <- file.path(local_dir, fname)
    tryCatch({
      download.file(url, dest, mode = "wb")
    }, error = function(e) {
      message("No encontrado ", fname)
    })
  }

  return(local_dir)
}
```

Extracto de código 8.1: Código para acceso y descarga de datos

Debido a la integración del repositorio en la plataforma BioStudies, el acceso y posterior descarga de algunos experimentos generaba errores por el cambio de formato(pasó de formato `XML` a `JSON`) que la respuesta de la API experimentó, es por ello que se ha desarrollado una función auxiliar `downloadFromFTP()` que accede al servidor FTP de ArrayExpress. Así, conseguimos mitigar los errores. Cabe explicar que la utilizamos como segunda opción ya que nuestro prioridad es explotar las herramientas del paquete BioConductor y por tanto, la función `getAE()`.

Esta función devuelve una carpeta de nombre igual al identificador del experimento, con todos los archivos relativos a el en su interior.

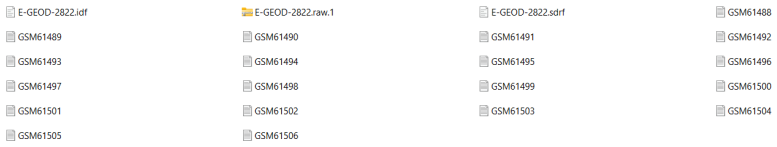


Figura 8.5: Datos descargados para experimento E-GEOD-2822

8.3.2. Generación y modificación de archivos

El siguiente bloque funcional del flujo corresponde a la generación y/o modificación de los archivos de entrada necesarios para la ejecución del algoritmo TriGen. Una vez descargados los datos de expresión génica, esta etapa se encarga de construir los archivos estructurados que definen las tres dimensiones sobre las que opera TriGen: genes, condiciones experimentales y puntos temporales. Además, se generan las matrices de expresión correspondientes a cada tiempo así como el archivo de configuración `config_file.tricfg` necesario y se modifica el archivo `Resources.xml` incluyendo la entrada del conjunto de datos generado a partir del nuevo experimento. Para ello, se ha desarrollado la función `fileCreationFinal()`, esta única función genera los recursos mediante la siguiente lógica:

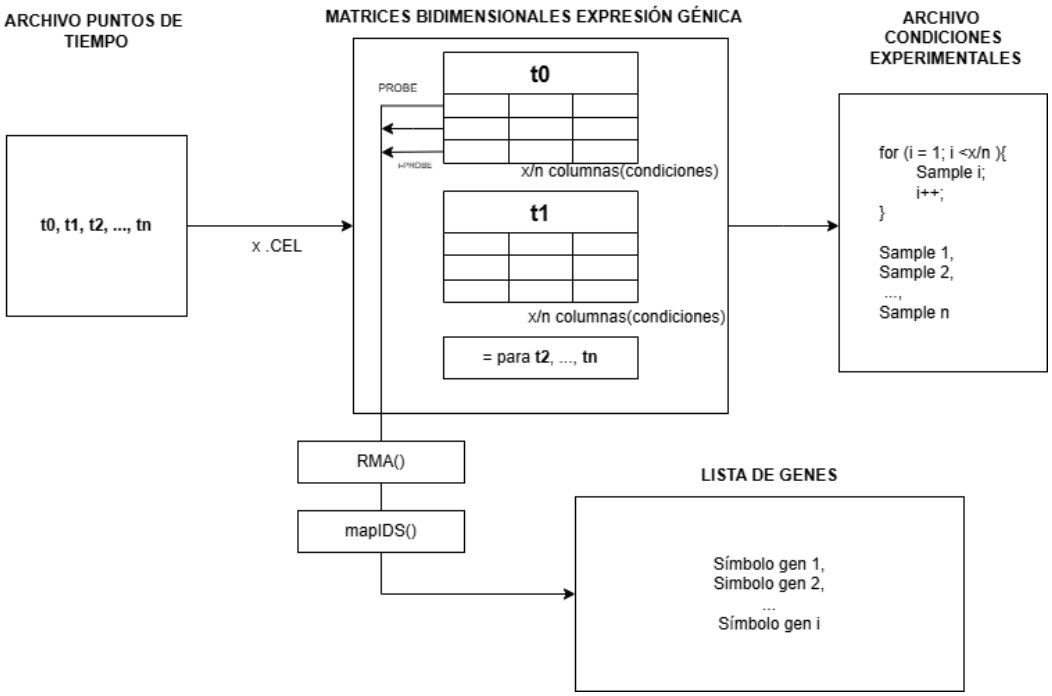


Figura 8.6: Esquema general preprocesamiento

La desglosaremos en 6 partes, para comprender su funcionamiento:

```
1
2  fileCreationFinal <- function(accesion_number, base_dir){
3
4    ### FICHERO ARCHIVO PUNTOS DE TIEMPO
5    sdrf_path <- file.path(base_dir,"DatosExp", accesion_number, paste0(
       accesion_number, ".sdrf.txt"))
6    sdrf_data <- read.delim(sdrf_path, header = TRUE, sep = "\t",
       stringsAsFactors = FALSE)
7
8    columnas_tiempo <- grep("time", colnames(sdrf_data), value = TRUE,
       ignore.case = TRUE)
9
10   output_dir <- file.path(base_dir, "TrLab5/resources", accesion_number)
11
12   if (!dir.exists(output_dir)) {
13     dir.create(output_dir, recursive = TRUE)
14   }
15
16   output_path <- file.path(output_dir, "time_points.txt")
17
18   time_columns_int <- columnas_tiempo[sapply(sdrf_data[columnas_tiempo],
       is.integer)]
19
20   sdrf_agrupado <- sdrf_data %>%
```

```
    dplyr::group_by(sdrf_data[[time_columns_int]]) %>%
    dplyr::summarise(Scans = paste(Array.Data.File, collapse = ", ")) %>%
    dplyr::arrange(time_columns_int)  # Ordenar por tiempo

chr_column <- sdrf_agrupado[[2]]
counts <- sapply(strsplit(chr_column, ","), length)
most_common_count <- as.numeric(names(sort(table(counts), decreasing = TRUE)[1]))
valid_rows <- counts == most_common_count
cleaned_sdrf_agrupado <- sdrf_agrupado[valid_rows, ]
valores_tiempo <- cleaned_sdrf_agrupado[[1]]
time_unit <- unique(sdrf_data[[columnas_tiempo[[2]]]])
cleaned_value_times <- paste(valores_tiempo, time_unit)

writeLines(cleaned_value_times, con = output_path)
```

Extracto de código 8.2: Generación de recursos I.

En este primer bloque 8.2 donde da comienzo la función se genera el archivo que contiene la lista de puntos temporales, mediante la lectura y procesamiento del archivo `.SDRF` donde se encuentran los metadatos detallados de cada muestra. Para ello, y debido a la diversidad en estos archivos ya mencionada, se filtran las columnas que contengan la columna `"time"`. A continuación, se agrupan las muestras por punto temporal y se resume la información para cada grupo, asociando cada tiempo con los archivos de escaneo correspondientes. Esta agrupación `cleaned_sdrf_agrupado` se ordena por puntos temporales y se filtran las filas que tienen un número de muestras por tiempo que permita mantener los puntos temporales con una estructura homogénea. Finalmente, se extraen los valores de tiempo y se combinan con su unidad (por ejemplo, "day", "hour"), y el resultado se escribe en el archivo **`time_points.txt`**.

```

### ARCHIVO CONDICIONES EXPERIMENTALES y MATRICES DE EXPRESION POR PUNTO TEMPORAL.
cleaned_sdrf_agrupado$scan_count <- sapply(strsplit(
 cleaned_sdrf_agrupado$Scans, ", "), length)

num_scans <- max(cleaned_sdrf_agrupado$scan_count)
samples <- sapply(1:num_scans, function(i) paste("Sample", i))

output_path <- file.path(output_dir, "samples.txt")

writeLines(samples, con = output_path)

affy.set <- affy::ReadAffy(celfile.path = file.path(base_dir, "DatosExp
 ", accesion_number))
```

```r

  eset <- affy::rma(affy.set)
  exprs_matrix <- as.data.frame(exprs(eset))

  exprs_matrix <- exprs_matrix[apply(exprs_matrix, 1, function(row) {
    all(!is.na(row) & trimws(row) != "")
  }), ]

  exprs_by_time <- list()
  cleaned_sdrf_agrupado <- as.data.frame(cleaned_sdrf_agrupado)
  colnames(cleaned_sdrf_agrupado)[1] <- "Tiempo"

  for (t in valores_tiempo) {
    scans_t <- cleaned_sdrf_agrupado %>%
      dplyr::filter(Tiempo == t) %>%
      dplyr::pull(Scans) %>%
      strsplit(", ") %>%
      unlist()

    exprs_by_time[[as.character(t)]] <- exprs_matrix %>%
      dplyr::select(all_of(scans_t))
  }

  resources <- list()
  for (t in names(exprs_by_time)) {
    write.table(exprs_by_time[[t]],
                file = paste0(output_dir, "/exprs_time_", t, ".csv"),
                sep = ";",
                col.names = FALSE,
                row.names = FALSE,
                quote = FALSE)

    resources <- append(resources, paste0("exprs_time_", t, ".csv"))
  }
```

Extracto de código 8.3: Código generación de recursos II.

En el segundo bloque 8.3 se generarán otros dos de los archivos necesarios: `samples.txt` y las matrices de expresión génica por punto temporal.

1. Utilizaremos la agrupación anterior `cleaned_sdrf_agrupado` para ver el número de muestras que hay por tiempo y así poder asignarle un nombre genérico(`Sample1, Sample 2, etc.`), los cuales los escribimos en **`samples.txt`**.

2. A continuación, se hace una lectura de los datos en crudo del experimento y se procesan empleando el método RMA(ya explicado en secciones

anteriores). Esta matriz se divide en subconjuntos según los puntos temporales válidos obtenidos en el bloque anterior, generando la matriz que necesitamos donde las columnas son las muestras asociadas a cada punto temporal. Cada una de estas matrices se almacena en su respectivo **exprs_time_XX** siendo XX el valor del punto de tiempo.

```
keys <- geneNames(affy.set)
  chip_name <- as.character(paste0(cleancdfname(affy.set@cdfName, addcdf
   = FALSE),".db"))

  if (!requireNamespace(chip_name, quietly = TRUE)) {
    BiocManager::install(chip_name, ask = FALSE)
  }

  library(chip_name, character.only = TRUE)
  db <- get(chip_name)
  gene_symbols_mapIDs <- AnnotationDbi::mapIds(db, keys, column = c("
   SYMBOL"),
                                               keytype = "PROBEID",
   multiVals = "first")

  output_path <- file.path(output_dir, "genes.txt")
  writeLines(gene_symbols_mapIDs, con = output_path)
```

El último bloque funcional de generación de archivos se encarga de general la lista de genes a partir de los identificadores de conjuntos de sondas obtenidos tras el procesamiento mediante RMA.

1. Utilizando el nombre de chip que se muestra en los metadatos(específicamente en el archivo `.IDF`), se construye el nombre de paquete de anotación correspondiente al chip, el cúal debemos instalar si no se ha hecho previamente.

2. A continuación, mediante la función `mapIds()`, se realiza el mapeo con los símbolos de genes.

3. Finalmente, se almacena la lista en el archivo **genes.txt**.

Todos estos archivos, se almacenan en la misma carpeta de tal forma que sean accesibles para TriGen y reconocibles en el archivo **resources.XML**, el cual se modifica de la siguiente forma.

```
   ##  MODIFICACION XML
  xml_path <- file.path(
    base_dir,
```

```
  "TrLab5", "resources", "resources.xml"
)
xml_file <- read_xml(xml_path)

idf_path <- file.path(base_dir,"DatosExp", accesion_number, paste0(
 accesion_number, ".idf.txt"))
idf_data <- read.delim(idf_path, header = FALSE, sep = "\t",
 stringsAsFactors = FALSE)
idf_data <- t(idf_data)
colnames(idf_data) <- as.character(idf_data[1, ])
idf_data <- as.data.frame(idf_data[-1, ])
idf_data[idf_data == ""] <- NA

datasets <- xml_find_all(xml_file, ".//dataset")
num_datasets <- length(datasets)
id <- num_datasets + 1
organism <- grep("organism",colnames(sdrf_data), ignore.case = TRUE,
 value = TRUE)[1]

exp_info <- c(idf_data$`Experiment Description`[1],
              length(keys),
              id,
              most_common_count,
              "150",
              length(cleaned_value_times),
              2,
              2,
              2,
              accesion_number,
              unique(sdrf_data[[organism]])[1],
              as.integer(num_scans),
              length(cleaned_value_times),
              "b"
)

if (tolower(exp_info[11]) == "mus musculus") {
  exp_info[11] <- "Mouse"
}
else if (tolower(exp_info[11]) == "homo sapiens"){
  exp_info[11] <- "Human"
}

dataset_name <- exp_info[10]

existing <- xml_find_all(
  xml_file,
  sprintf(".//dataset[@name='%s']", dataset_name)
```

```
48  )
49
50  if (length(existing) == 0) {
51    new_dataset <- xml_add_child(xml_file, "dataset",
52                                 description = exp_info[1],
53                                 geneSize    = exp_info[2],
54                                 id          = exp_info[10],
55                                 maxC        = exp_info[4],
56                                 maxG        = exp_info[5],
57                                 maxT        = exp_info[6],
58                                 minC        = exp_info[7],
59                                 minG        = exp_info[8],
60                                 minT        = exp_info[9],
61                                 name        = dataset_name,
62                                 organism    = exp_info[11],
63                                 sampleSize  = exp_info[12],
64                                 timeSize    = exp_info[13],
65                                 type        = "b"
66    )
67
68    resources_node <- xml_add_child(new_dataset, "resources", separator =
      ";")
69    for (resource in resources) {
70      xml_add_child(resources_node, "resource", resource)
71    }
72    xml_add_child(new_dataset, "genes",   "genes.txt")
73    xml_add_child(new_dataset, "samples", "samples.txt")
74    xml_add_child(new_dataset, "times",   "times.txt")
75
76    write_xml(xml_file, file.path(base_dir, "TrLab5/resources/resources.
     xml"))
77  } else {
78    message("El experimento '", dataset_name, "' ya esta registrado en
     resources.XML")
79  }
```

En este trozo de código registramos la nueva entrada <dataset> de tal forma que pueda ser procesado por TriGen durante su ejecución. Para ello:

1. Recuperamos información descriptiva del experimento de los archivos `.IDF` y `.SDRF` como la descripción del experimento, organismo biológico...

2. Actualizamos el nuevo identificador único que otorgaremos al experimento que estamos añadiendo comprobando el del anterior más uno.

3. Generamos el nuevo conjunto de datos con todos los atributos que requiere

el algoritmo. Algunos de ellos son: número de muestras, genes, numero de puntos de tiempo, etc. Cabe resaltar la importancia del parámetro `name` ya que será la referencia al archivo de configuración `config_file.tricfg` para obtener los parámetros que utilizara.

4. Se añade un subnodo `<resources>` que apunta a los archivos generados en los bloques previos.

5. Finalmente, se sobrescribe y guarda, haciendo que el experimento esté listo para ser analizado con TriGen, a falta de generar el archivo de congiguración correspondiente.

```
### GENERACION ARCHIVO CONFIGURACION .TRICFG
 out <- file.path(base_dir, "TrLab5/OutputTriGen", accesion_number)
 if (!dir.exists(out)) dir.create(out)

 output_trigen_dir <- paste0(out,"/",accesion_number)

 cfg_lines <- c(
   paste0("dataset = ", exp_info[10]),
   paste0("out = ", output_trigen_dir),
   paste0("fitness = msl"),
   paste0("N = 3"),
   paste0("G = 500"),
   paste0("I = 200"),


   paste0("Ale = 0.5"),
   paste0("Sel = 0.5"),
   paste0("Mut = 0.4"),
   paste0("Wf = 0.8"),
   paste0("Wg = 0.0"),
   paste0("Wc = 0.05"),
   paste0("Wt = 0.05"),
   paste0("WOg = 0.05"),
   paste0("WOc = 0.03"),
   paste0("Wot = 0.02"),

   paste0("minG = ", exp_info[8]),
   paste0("maxG = ", exp_info[5]),

   paste0("minC = ", exp_info[7]),
   paste0("maxC = ", exp_info[4]),

   paste0("minT = ", exp_info[9]),
   paste0("maxT = ", exp_info[6])
 )
```

```
36
37   tricfg_path <- file.path(output_dir, "config_file.tricfg")
38   writeLines(cfg_lines, tricfg_path)
39
40   return(list(
41     archivos = list.files(output_dir, full.names = TRUE, recursive = TRUE
      )
42   ))
43 }
```

En este bloque final de la función se genera el archivo `config_file.tricfg`. Este archivo contiene:

- El nombre del dataset en `resources.XML`.
- La ruta donde se almacenarán los resultados.
- Los parámetros necesarios para la inicialización del algoritmo genético(número de ejecuciones, tamaño de población, generaciones, probabilidades de cruce, selección y mutación).
- Los límites mínimo y máximo permitidos para el número de genes, condiciones y tiempos por triclúster.

8.3.3. Ejecución del algoritmo

El siguiente paso será el la ejecución de TriGen para obtener los triclústeres solución. La reproducción del algoritmo se realiza mediante la ejecución de comandos en la consola de R de la siguiente forma:

```
1  runTriGen <- function(accesion_number, base_dir) {
2    jar_path <- file.path(base_dir, "TrLab5/TriGenApp.jar")
3    config_dir <- file.path(base_dir, "TrLab5/resources", accesion_number)
4    config_path <- file.path(config_dir, "config_file.tricfg")
5
6    output_dir <- file.path(base_dir, "TrLab5/OutputTriGen",
      accesion_number)
7
8    cmd <- paste("java -jar", shQuote(jar_path), shQuote(config_path))
9    message("Ejecutando comando:\n", cmd)
10   system(cmd)
11
12   if (dir.exists(output_dir)) {
```

```
    archivos <- list.files(output_dir, full.names = TRUE, recursive =
    TRUE)
  } else {
    archivos <- character(0)
  }
  print(archivos)

  return(list(dir = output_dir, archivos = archivos))
}
```

Extracto de código 8.4: Ejecución mediante script del algoritmo Trigen

La función `runTriGen()` automatiza el lanzamiento mediante la ejecución del archivo `.jar` del algoritmo, un ejecutable que contiene todo el código compilado y los recursos necesarios para que funcione, en la consola de R. A este ejecutable se le debe indicar los parámetros de lanzamiento, que como explicamos en secciones anteriores, se encuentran en el archivo `config_file.tricfg`.

El ejecutable `.jar` fue proporcionado por mi tutor, es fácilmente accesible y descargable en su repositorio Github [24].

Tras esta ejecución, se genera una carpeta en la ubicación indicada que contiene los resultados obtenidos(triclústeres solución) representados con distintos archivos que nos serán de gran utilidad para la implementación de PantherDB, los más relevantes y utilizados e integrados en nuestra herramienta son:

- **Archivos subcarpeta `coordinates`:** Resume el tricluster en coordenadas: **tiempo(t), condición(s), símbolo del gen(g), el(nivél de expresión)**. Cada fila representa un punto del cubo tridimensional, lo que lo hace facilmente convertible a estrctura `data.frame` de R y, por tanto, manejable.

- **Archivos subcarpeta `genes`:** Define el conjunto de genes mediante sus simbolos que forman parte de un triclúster solución.

- Métricas archivo .triq: Muestra métricas asociadas a los triclústeres obtenidos que representan la calidad mediante el índice ponderado **`TRIQ`**:

  - La calidad biológica **`BIOQ`** mediante un análisis de enriquecimiento.

  - La coherencia(REFERENCIAR SECCION DONDE SE EXPLICA) gráfica de los patrones de expresión genética.

  - La correlación lineal entre las combinaciones de genes y condiciones a lo largo del tiempo mediante el *coeficiente de Pearson* **`PEQ`**.

  - La correlación(no necesariamente lineal)entre las combinaciones de genes

t	s	g	el
0	0	5021	41.063
0	0	8944	67.053
0	1	5021	4.131
0	1	8944	67.658
0	2	5021	40.735
0	2	8944	67.558
0	3	5021	41.845
0	3	8944	67.061
1	0	5021	40.374
1	0	8944	66.455
1	1	5021	40.299
1	1	8944	66.998
1	2	5021	39.766
1	2	8944	66.188
1	3	5021	41.099
1	3	8944	65.133
2	0	5021	41.034
2	0	8944	67.432
2	1	5021	42.972
2	1	8944	69.165
2	2	5021	40.101
2	2	8944	69.058
2	3	5021	41.131
2	3	8944	66.935

Tabla 8.1: Ejemplo coordenadas tricluster solucion experimento E-GEOD-2822

y condiciones a lo largo del tiempo mediante el *coeficiente de Spearman* **SPQ**.

Se representa formalmente mediante la siguiente fórmula:

$$TRIQ(TRI) = \frac{1}{W_{bio} + W_{gr} + W_{pe} + W_{sp}} \cdot \big[W_{bio} \cdot BIOQ(TRI) + W_{gr} \cdot GRQ(TRI) + W_{pe} \cdot PEQ(TRI) + W_{sp} \cdot SPQ(TRI)\big]$$

$$W_{bio} = 0{,}5$$
$$W_{gr} = 0{,}4$$
$$W_{pe} = 0{,}05$$
$$W_{sp} = 0{,}05$$

Para comprender el porque de estos pesos, se puede revisar el estudio y trabajo realizado en [25], concretamente en la subsección *3.6*.

Obtendremos tantos archivos en cada subcarpeta de coordenadas y genes cómo triclústeres óptimos a encontrar se hayan introducido cómo parámetro en el archivo de configuración de lanzamiento.

8.4. Funcionalidades

En esta subsección, describiremos el código y la utilidad de las funcionalidades integradas en la herramienta desarrollada, agrupada en dos grandes grupos. Por un lado, las que interactúan directamente con la API de PANTHER DB. Por otro lado, las que operan sobre los archivos de salida generados por TriGen, permitiendo la representación gráfica y muestra de estadísticas asociadas de los triclústeres obtenidos.

8.4.1. Funcionalidades basadas en la API de PantherDB

Aunque existen librerias en R [26] que implementan funciones predefinidas para interactuar con la API de PantherDB.

Se ha optado por consumir directamente los recursos ofrecidos [27] por la API REST de la plataforma por elección propia y así afrontar de forma directa los retos programáticos que un bioinformático debe abordar como el proceso de integración de APIs REST, la construcción de peticiones y/o respuestas personalizadas, con la posibilidad en un futuro, de continuar trabajando en esta herramienta para conseguir una integración completa de los servicios que ofrece PANTHERDB. Para ello, se ha desarrollado el código que se muestra a continuación.

```
1 ### FUNCION CONSUMO API PARA ANALISIS SOBREREPRESENTACION
2 runOverrepAnalysis <- function(genes, organism, annotDataSet,
    enrichmentTestType = "FISHER", correction = "FDR") {
3   body <- list(
4     geneInputList      = paste(genes, collapse = ","),
```

```
    organism           = organism,
    annotDataSet       = annotDataSet,
    enrichmentTestType = enrichmentTestType,
    correction         = correction
  )
  resp <- POST(
    url    = "https://pantherdb.org/services/oai/pantherdb/enrich/overrep
    ",
    body   = body,
    encode = "form"
  )
  json <- httr::content(resp, as = "text", encoding = "UTF-8")
  dat  <- fromJSON(json)

  as.data.frame(dat$results$result)
}
```

Extracto de código 8.5: Función para el análisis del sobrerreprentación mediante el consumo de API REST de PantherDB

En primer lugar, se ha desarrollado la función **runOverrepAnalysis()** (REFERNCIAR) para construir el formulario adecuado y realizar una consulta a la API, con el objetivo de realizar un análisis de sobre-representación, a partir de las listas de genes de cada tricluster solución. La respuesta obtenida en formato `JSON` se recibe y transforma en formato `data.frame`, para poder así manejarla.

```
### FUNCION AUXILIAR OBTENER N TERMINOS MAS SOBREREPRESENTADOS
getTopTerms <- function(df, n = 10) {
  top_terms <- df %>%
    dplyr::arrange(desc(number_in_list)) %>%
    dplyr::slice_head(n = 10) %>%
    unnest_wider(term, names_sep = "_") %>%
    dplyr::rename(term_label = term_label, term_id = term_id)
}
```

Extracto de código 8.6: Función auxiliar I para el análisis de sobrerreprentación

```
### FUNCION AUXILIAR PARA CREAR GRAFICO
plotEnrichment <- function(top_terms, title = NULL) {
  p <- ggplot(top_terms, aes(x = "", y = number_in_list, fill =
    term_label,
                             text = paste0("Termino: ", term_label, "<br>
    Genes: ", number_in_list))) +
    geom_bar(stat = "identity", width = 1) +
    labs(title = title, fill = "Termino") +
    theme_void() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

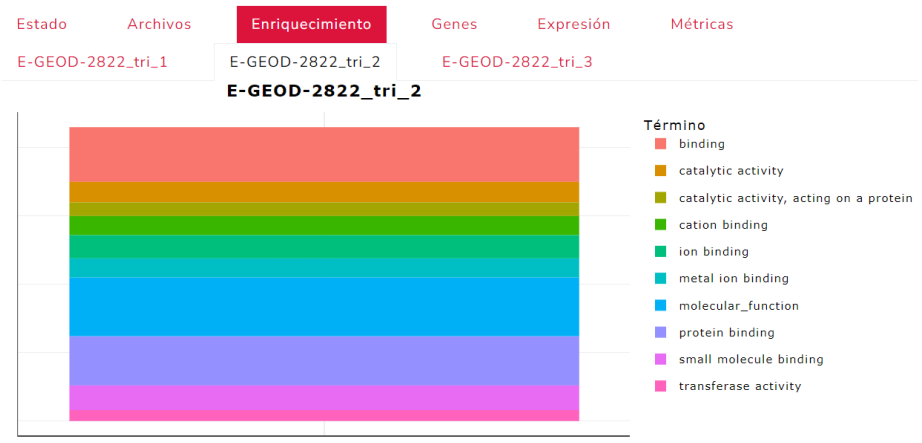


Figura 8.7: Análisis sobrerrepresentación para uno de los triclústeres solución del experimento E-GEOD-2822.

```
  ggplotly(p, tooltip = "text")
}
```

Extracto de código 8.7: Función auxiliar II para el análisis de sobrerreprentación

Además, se han desarrollado las funciones auxiliares **`getTopTerms()`** y **`plotEnrichment()`** para respectivamente, obtener los n terminos más reperesentados en los resultados de nuestro análisis y poder crear un gráfico interactivo para visualizar estos términos.

```
### FUNCION CONSULTA GENES DE CADA TRICLUSTER SOLUCION
queryGeneInfo <- function(accesion_number, organism, base_dir) {
  # Directorio con los .txt de genes
  geneLists_dir <- file.path(base_dir,
    "TrLab5/OutputTriGen",
    paste0(" ", accesion_number),
    accesion_number,
    "genes"
  )

  geneLists <- list.files(geneLists_dir, pattern = "\\.txt$", full.names
    = TRUE)

  results <- lapply(geneLists, function(fpath) {
    genes <- readLines(fpath, warn = FALSE)
    genes <- genes[genes != "" & !is.na(genes)]
    params <- list(
      organism      = organism,
      geneInputList = paste(genes, collapse = ",")
    )

    resp <- GET(
```

```
      url    = "https://pantherdb.org/services/oai/pantherdb/geneinfo",
      query = params,
      accept_json()
    )
    dat <- fromJSON(httr::content(resp, as = "text", encoding = "UTF-8"))

    if (!is.null(dat$search$mapped_genes$gene)) {
      df_genes <- as.data.frame(dat$search$mapped_genes$gene,
                                stringsAsFactors = FALSE)
    } else {
      df_genes <- data.frame()
    }
  })

  names(results) <- tools::file_path_sans_ext(basename(geneLists))
  return(results)
}
```

Extracto de código 8.8: Función para la consulta información de genes en la base de datos de PantherDB

En segundo lugar, se ha desarrollado la función **`queryGeneInfo()`** que permite consultar información detallada sobre los genes obtenidos en cada triclúster. Esta funcionalidad es de gran utilidad para poder facilitar el análisis y la interpretación de los resultados.

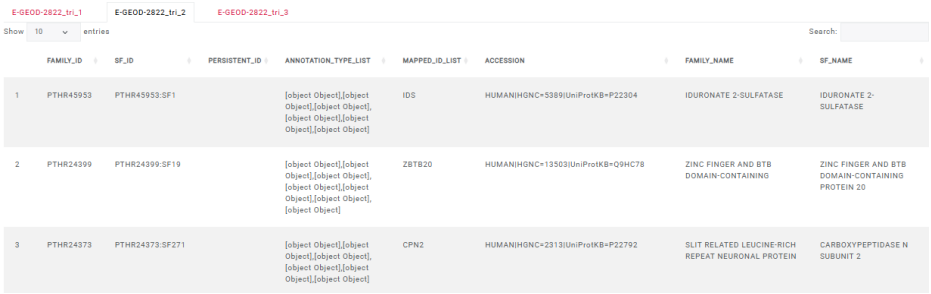


Figura 8.8: Consulta genes para uno de los triclústeres solución del experimento E-GEOD-2822.

8.4.2. Funcionalidades basadas en los resultados de TriGen

Este segundo bloque se ha desarrollado con el objetivo de facilitar la exploración e interpretación de los resultados mediante gráficas obtenidas así como su validación biológica y técnica mediante la representación de las métricas asociadas. Para ello, se ha desarrollado el código que se muestra a continuación.

```
### FUNCION OBTENER GRAFICAS VARIACION NIVELES
### DE EXPRESION POR PUNTO DE TIEMPO SEGUN CONDICION.
geneExpInteractive <- function(accesion_number, base_dir) {
  coords_dir <- file.path(base_dir,
    "TrLab5/OutputTriGen",
    accesion_number,
    accesion_number,
    "coordinates"
  )

  csvs <- list.files(coords_dir, pattern = "\\.csv$", full.names = TRUE)
  if (length(csvs) == 0) {
    warning("No hay archivos CSV en ", coords_dir)
    return(list())
  }

  all_plots <- list()

  for (path in csvs) {
    df <- read.csv(path, sep = ";", stringsAsFactors = FALSE)
    fname <- tools::file_path_sans_ext(basename(path))
    conds <- unique(df$s)

    plots_per_cond <- lapply(conds, function(cond) {
      dsub <- df[df$s == cond, ]
      plot_ly(dsub, x = ~t, y = ~el, color = ~factor(g),
              colors = "Set1", mode = "lines+markers", type = "scatter",
              hoverinfo = "text",
              text = ~paste("Gen:", g,
                            "<br>Tiempo:", t,
                            "<br>Expr:", el)
      ) %>%
        layout(
          title = paste0(fname, " condicion s=", cond),
          xaxis = list(title = "Tiempo (t)"),
          yaxis = list(title = "Expresion (el)"),
          legend = list(title = list(text = "<b>Gen</b>"))
        )
    })
    names(plots_per_cond) <- paste0(fname, "_s", conds)
    all_plots[[fname]] <- plots_per_cond
  }
  return(all_plots)
}
```

Extracto de código 8.9: Función para generación de gráficas variación nivel de expresión génica a lo largo del tiempo según la condición.

Se ha desarrollado la función **geneExpInteractive()** para visualizar la variación de expresión génica a lo largo del tiempo para cada condición experimental, utilizando los archivos de coordenadas generados por el algoritmo y poder evaluar la coherencia de patrones descubiertos.

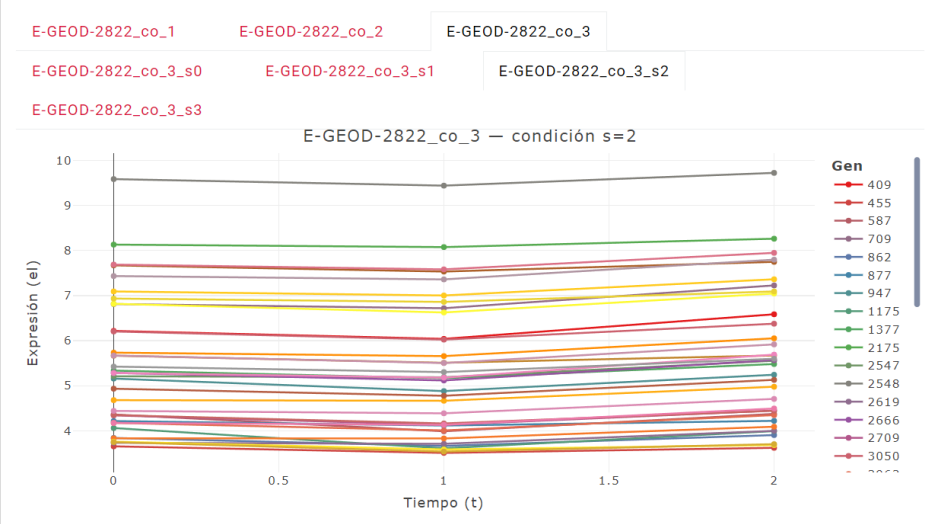


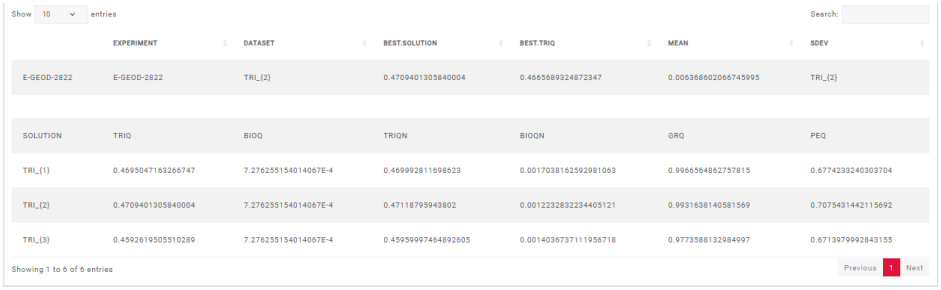
Figura 8.9: Gráfico variación nivel de expresión respecto al tiempo para una condición de uno de los triclústeres solución del experimento E-GEOD-2822.

```
showMetrics <- function(accesion_number, base_dir){
  metricas_path <- file.path(base_dir,
    "TrLab5/OutputTriGen",
    accesion_number,
    accesion_number,
    paste0(accesion_number,"_triq.csv")
  )

  csv <- read.csv(metricas_path, sep = ";")
  csv <- csv %>%
    dplyr::select(-BEST.SOLUTIONN, -BEST.TRIQN, -MEANN, -SDEVN)
}
```

Extracto de código 8.10: Función obtención métricas de calidad generadas por Trigen.

Se ha desarrollado la función **showmetrics()** para leer y mostrar las métricas de calidad generadas por el algoritmo para cada tricluster.



Show 10 entries Search:

	EXPERIMENT	DATASET	BEST.SOLUTION	BEST.TRIQ	MEAN	SDEV
E-GEOD-2822	E-GEOD-2822	TRI_(2)	0.4709401305840004	0.4665689324872347	0.006368602066745995	TRI_(2)

SOLUTION	TRIQ	BIOQ	TRIQN	BIOQN	GRQ	PEQ
TRI_(1)	0.4695047163266747	7.2762551540140 67E-4	0.469992811698623	0.0017038162592981063	0.9966564862757815	0.6774233240303704
TRI_(2)	0.4709401305840004	7.2762551540140 67E-4	0.47118795943802	0.0012232832234405121	0.9931638140581569	0.7075431442115692
TRI_(3)	0.4592619505510289	7.2762551540140 67E-4	0.45959997464892605	0.0014036737111956718	0.9773588132984997	0.6713979992843155

Showing 1 to 6 of 6 entries Previous 1 Next

Figura 8.10: Metricas TriGen para experimento E-GEOD-2822

8.5. Lanzamiento ArrayExpress Processor for TriGen

Por último, para el desarrollo de la interfaz gráfica que permite la interacción con toda la herramienta bioinformática construida, se ha utilizado Shiny App.

Figura 8.11: Logo Shiny

Shiny es un paquete que permite crear aplicaciones web interactivas directamente desde código en R, sin necesidad de conocimientos en desarrollo web. Gracias a esta tecnología, se ha diseñado una interfaz intuitiva que integra todas las funcionalidades descritas en la sección anterior. Una aplicación de este tipo se estructura principalmente en tres componentes clave: **UI(user interface)** donde se define qué verá el usuario, **server** donde se define la lógica de la aplicación y **shinyApp()** para la combinación de las dos anteriores y el

lanzamiento. A continuación, mostramos el código de cada uno.

```
theme <- bs_theme(
  version    = 5,
  bootswatch = "lux",
  primary    = "#DC143C",
  secondary  = "#8B0000"
)

ui <- fluidPage(
  useShinyjs(),
  theme = theme,
  add_busy_spinner(
    spin     = "fading-circle",
    position = "bottom-right",
    margins  = c(20,50),
    color    = "#DC143C"
  ),
  tags$head(
    tags$link(rel = "icon", type = "image/png", href = "logoAE4TriGen.png
   "),
    tags$script(HTML("
      document.addEventListener('DOMContentLoaded', function() {
        var t = [].slice.call(
          document.querySelectorAll('[data-bs-toggle=\"tooltip\"]')
        );
        t.forEach(el => new bootstrap.Tooltip(el));
      });
      .container-fluid {
        padding-left: 0;
        padding-right: 0;
      };
    "))
  ),

fluidRow(
  style = "background: linear-gradient(90deg, #DC143C, #8B0000);",
  column(8,
         div(
           style = "text-align:center; margin:20px 0;padding: 15px;
   border-radius: 10px;",
           tags$h1(
             "ARRAYEXPRESS PROCESSOR for TriGen Algorithm",
             style = "
              font-size:1.2rem;
              font-weight:bold;
              white-space: nowrap;
```

```
              overflow: hidden;
              color: white;
              font-family: 'Montserrat', sans-serif;
            "
          )
        )
  ),
  column(4,
         div(
           style = "text-align:right; margin:20px 0;",
           actionButton(
             inputId = "open_search",
             label   = NULL,
             icon    = icon("search"),
             class   = "btn btn-link search-icon"
           )
         )
  )
),


tags$style("
  .search-icon:hover {
    transform: scale(1.1);
    transition: 0.3s ease-in-out;
  }

  .btn-link:hover {
    color: #DC143C;
    transition: 0.3s;
  }




"),


  hr(),

  uiOutput("search_table"),

  # SELECTOR DE DIRECTORIO
  fluidRow(
    column(6,
           shinyDirButton(
             "choose_dir", "Seleccionar directorio",
```

```
            "Selecciona carpeta base", FALSE
          )
    ),
    column(6,
          verbatimTextOutput("chosen_dir")
    )
  ),
  hr(),

  # MAIN LAYOUT
  fluidRow(
    column(4,
          wellPanel(
            selectizeInput("accession", "Experimento:", choices=NULL,
                           options=list(create=TRUE), width="100%"),
            actionButton("download_btn","Descargar datos",
                         icon=icon("download"),class="btn btn-danger w
    -100 mb-2"),
            actionButton("process_btn","Crear archivos",
                         icon=icon("cogs"),class="btn btn-danger w-100
    mb-2"),
            actionButton("runtrigen_btn","Ejecutar TriGen",
                         icon=icon("play"),class="btn btn-danger w-100
    mb-2"),
            selectInput("genome","Genoma (PANTHER):",choices=NULL),
            selectInput("annot_dataset","Dataset anotacion:",choices=
    NULL),
            actionButton("enrich_btn","Enriquecimiento",
                         icon=icon("chart-bar"),class="btn btn-danger w
    -100 mb-2"),
            actionButton("consultar_genes","Info genes",
                         icon=icon("dna"),class="btn btn-danger w-100 mb
    -2"),
            actionButton("expr_btn","Expr. interactiva",
                         icon=icon("chart-line"),class="btn btn-danger w
    -100 mb-2"),
            actionButton("metrics_btn","Metricas",
                         icon=icon("table"),class="btn btn-danger w-100"
    )
          )
    ),
    column(8,
          navset_pill(
            tabPanel("Estado",
                     verbatimTextOutput("download_status"),
                     verbatimTextOutput("process_status")
            ),
```

```
            tabPanel("Archivos",
                     verbatimTextOutput("archivos_creados")
            ),
            tabPanel("Enriquecimiento",
                     uiOutput("plots_enrichment")
            ),
            tabPanel("Genes",
                     uiOutput("geneinfo_ui")
            ),
            tabPanel("Expresion",
                     uiOutput("expr_plots")
            ),
            tabPanel("Metricas",
                     dataTableOutput("metrics_table")
            )
          )
    )
  ),
  fluidRow(
    class = "footer bg-light border-top",
    style = "padding: 1rem 0; font-size: 0.85rem;",


    column(
      4,
      tags$ul(
        class = "list-unstyled mb-0",
        tags$li(
          class = "text-muted",
          "Desarrollado por Alejandro Marquez Gonzalez"
        )
      )
    ),


    column(
      4,
      div(
        class = "text-center",
        tags$a(
          href   = "https://www.ebi.ac.uk/arrayexpress/",
          target = "_blank",
          class  = "text-primary fw-bold text-decoration-none",
          "Visita ArrayExpress"
        )
      )
    ),
```

```
177
178
179       column(
180         4,
181         div(
182           class = "d-flex justify-content-end align-items-center",
183           img(src = "logo2.png", height = "40px")
184         )
185       )
186     )
187
188 )
```

Extracto de código 8.11: Código interfaz y lógica de ShinyApp

El objetivo ha sido facilitar la interacción del usuario. La interfaz gráfica se presenta a continuación:

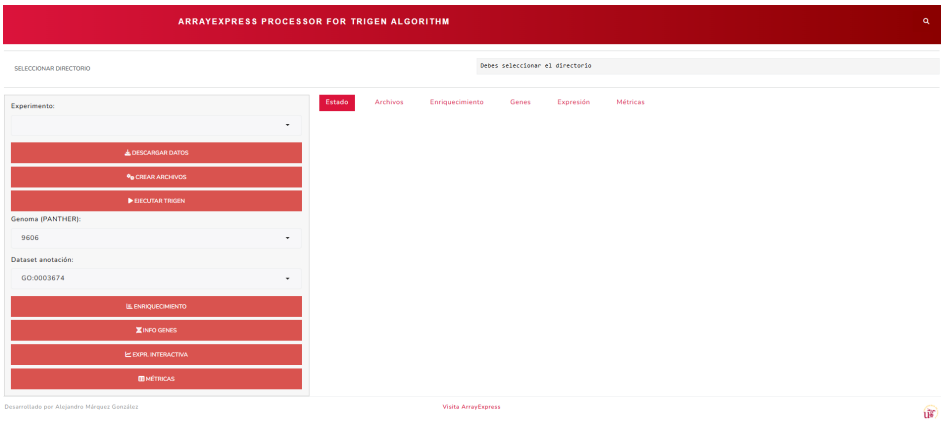


Figura 8.12: Interfaz gráfica para el usuario ArrayExpress Processor for TriGen Algorithm.

```
1
2 server <- function(input, output, session) {
3   roots <- c(Home = normalizePath("~"))
4
5   shinyDirChoose(
6     input,
7     id    = "choose_dir",
8     roots = roots,
9     session = session,
10    filetypes = c("", "txt", "csv")
11  )
12
13  baseDir <- reactive({
14    if (is.null(input$choose_dir) || length(input$choose_dir) == 0)
    return(NULL)
```

```
    normalizePath(
      parseDirPath(roots, input$choose_dir),
      winslash = "/")
  })

  output$chosen_dir <- renderText({
    bd <- baseDir()
    if (!shiny::isTruthy(bd)) {
      "Debes seleccionar el directorio"
    } else {
      paste("Directorio seleccionado:", bd)
    }
  })

  # 1) Poblar accesiones
  observe({
    acc <- list.dirs(file.path(baseDir(),"DatosExp"),
                     full.names=FALSE, recursive=FALSE)
    acc <- acc[nzchar(acc) & acc!="."]
    updateSelectInput(session, "accession", choices=acc, selected=acc[1])
  })

  # 2) PANTHER dropdowns
  observe({
    resp <- POST("http://pantherdb.org/services/oai/pantherdb/
   supportedgenomes",
                 accept_json())
    dat  <- fromJSON(httr::content(resp, "text", encoding="UTF-8"))
    gen  <- dat$search$output$genomes$genome$taxon_id
    updateSelectInput(session, "genome", choices=gen, selected=gen[1])
  })
  observe({
    resp <- GET("http://pantherdb.org/services/oai/pantherdb/
   supportedannotdatasets",
                accept_json())
    dat  <- fromJSON(httr::content(resp, "text", encoding="UTF-8"))
    ds   <- dat$search$annotation_data_sets$annotation_data_type$id
    updateSelectInput(session, "annot_dataset", choices=ds, selected=ds[1
   ])
  })

  # 3) Descargar datos
  observeEvent(input$download_btn, {
    req(baseDir(), input$accession)
    output$download_status <- renderText("Comprobando y descargando...")
    result <- downloadData(input$accession, base_dir = baseDir())
    output$download_status <- renderText({
```

```
59      if (identical(result, "already_downloaded")) {
60        files <- list.files(file.path(baseDir(),"DatosExp",
    input$accession),
61                            full.names=TRUE, recursive=TRUE)
62        paste0("Este experimento fue descargado previamente. Se
    encontraron los siguientes archivos:\n",
63               paste(files, collapse="\n"))
64      } else if (is.null(result)) {
65        paste("Error al descargar:", input$accession)
66      } else {
67        files <- list.files(file.path(baseDir(),"DatosExp",
    input$accession),
68                            full.names=TRUE, recursive=TRUE)
69        paste0("Descarga completa:\n", paste(files, collapse="\n"))
70      }
71    })
72  })
73
74  # 4) Crear archivos
75  observeEvent(input$process_btn, {
76    req(baseDir(), input$accession)
77    archivos <- tryCatch(fileCreationFinal(input$accession,
78                                           base_dir = baseDir()),
79                         error=function(e) NULL)
80    archivos <- list.files(file.path(baseDir(),"TrLab5/Resources",
    input$accession),
81                           full.names=TRUE, recursive=TRUE)
82    output$archivos_creados <- renderText(paste(archivos, collapse="\n"))
83  })
84
85  # 5) Ejecutar TriGen
86  observeEvent(input$runtrigen_btn, {
87    req(baseDir(),input$accession)
88    output$process_status <- renderText("Ejecutando TriGen...")
89    out <- tryCatch(runTriGen(input$accession,
90                              base_dir = baseDir()), error=function(e)
    NULL)
91    output$process_status <- renderText(
92      if (is.null(out)) "Error al ejecutar TriGen"
93      else paste("TriGen ejecutado. Resultados en:", out$dir)
94    )
95  })
96
97  # 6) Enriquecimiento funcional
98  observeEvent(input$enrich_btn, {
99    req(baseDir(),input$accession, input$genome, input$annot_dataset)
100   show_modal_spinner(text="Analizando...", color="#DC143C")
```

```
    genes_dir <- file.path(baseDir(), "TrLab5/OutputTriGen",
                           input$accession, input$accession, "genes")
    txts <- list.files(genes_dir, pattern="\\.txt$", full.names=TRUE)
    if (!length(txts)) {
      remove_modal_spinner()
      output$plots_enrichment <- renderUI(div(class="alert alert-warning"
    ,
                                              "No se encontraron archivos
     de genes."))
      return()
    }
    plots <- lapply(txts, function(f) {
      name <- tools::file_path_sans_ext(basename(f))
      genes <- readLines(f, warn=FALSE)
      df    <- runOverrepAnalysis(genes, input$genome,
    input$annot_dataset)
      top   <- getTopTerms(df, n=10)
      plotEnrichment(top, title=name)
    })
    remove_modal_spinner()
    output$plots_enrichment <- renderUI({
      tabs <- lapply(seq_along(plots), function(i) {
        nm <- tools::file_path_sans_ext(basename(txts[i]))
        tabPanel(nm, plotlyOutput(paste0("plt_", i)))
      })
      do.call(tabsetPanel, tabs)
    })
    for (i in seq_along(plots)) {
      local({
        idx <- i
        output[[paste0("plt_", idx)]] <- renderPlotly(plots[[idx]])
      })
    }
  })

  # 7) Consulta de genes
  observeEvent(input$consultar_genes, {
    req(baseDir(), input$accession, input$genome)

    res_list <- queryGeneInfo(input$accession,
                              input$genome, base_dir = baseDir())


    if (all(sapply(res_list, function(x) is.null(x) || nrow(x) == 0))) {
      output$geneinfo_ui <- renderUI({
        div(class="alert alert-warning",
            "No se encontraron resultados validos.")
```

```
    })
    return()
  }

  output$geneinfo_ui <- renderUI({
    tabs <- lapply(names(res_list), function(name) {
      tabPanel(
        title = name,
        dataTableOutput(outputId = paste0("geneinfo_tbl_", name))
      )
    })
    do.call(tabsetPanel, tabs)
  })

  for (name in names(res_list)) {
    local({
      nm <- name
      df <- res_list[[nm]]
      out_id <- paste0("geneinfo_tbl_", nm)
      output[[out_id]] <- renderDataTable({
        df
      }, options = list(pageLength = 10, scrollX = TRUE))
    })
  }
})

observeEvent(input$expr_btn, {
  req(baseDir(),input$accession)
  plots <- geneExpInteractive(input$accession, base_dir = baseDir())
  if (!length(plots)) {
    output$expr_plots <- renderUI(
      div(class="alert alert-warning","No hay archivos de coordenadas."
 )
    )
    return()
  }
  output$expr_plots <- renderUI({
    top_tabs <- lapply(names(plots), function(fn) {
      inner <- lapply(names(plots[[fn]]), function(cond) {
        pid <- paste0("expr_", fn, "_", cond)
        tabPanel(cond, plotlyOutput(pid))
      })
      tabPanel(fn, do.call(tabsetPanel, inner))
    })
    do.call(tabsetPanel, top_tabs)
  })
  for (fn in names(plots)) {
```

```
    for (cond in names(plots[[fn]])) {
      local({
        pod <- paste0("expr_", fn, "_", cond)
        pobj <- plots[[fn]][[cond]]
        output[[pod]] <- renderPlotly(pobj)
      })
    }
  }
})

# 9) Metricas TriGen
observeEvent(input$metrics_btn, {
  req(baseDir(), input$accession)
  df <- showMetrics(input$accession, base_dir = baseDir())
  output$metrics_table <- renderDataTable(df,
                                          options=list(pageLength=10,
                                                       scrollX=TRUE))
})

observeEvent(input$open_search, {
  showModal(modalDialog(
    title = "Buscar experimentos (BioStudies)",
    textInput("search_term", "Experimental design:", value = ""),
    actionButton("search_btn", "Buscar", icon = icon("search")),
    hr(),
    tableOutput("search_table"),
    footer = modalButton("Cerrar")
  ))
})


observeEvent(input$search_btn, {
  req(input$search_term)
  resultados <- consultaExp(input$search_term)

  if (nrow(resultados) == 0) {
    output$search_table <- renderTable({
      data.frame(Mensaje = "No se encontraron experimentos.")
    }, rownames = FALSE, colnames = FALSE)
  } else {
    output$search_table <- renderTable({
      resultados[, c("accession", "title", "files"), drop = FALSE]
    }, rownames = FALSE,
    hover = TRUE,
    border = TRUE,)
  }
})
```

```
238
239
240 }
```

```
1 shinyApp(ui, server)
```

Extracto de código 8.12: Función creación y ejecución ShinyApp

La barra lateral de la interfaz permite acceder y ejecutar de forma sencilla todas las funcionalidades previamente explicadas. Desde ella, el usuario puede descargar los datos necesarios, generar los recursos de entrada y lanzar la ejecución del algoritmo principal. Su diseño busca centralizar el control del flujo de trabajo, facilitando una navegación ordenada y eficiente a lo largo de todas las etapas del proceso.

Se ha integrado un buscador de experimentos que permite acceder a la base de datos y filtrar los resultados según el término de .[ex]perimental design.[es]pecificado por el usuario. En el contexto de esta aplicación, el principal interés se centra en aquellos experimentos clasificados como ”time series design”.

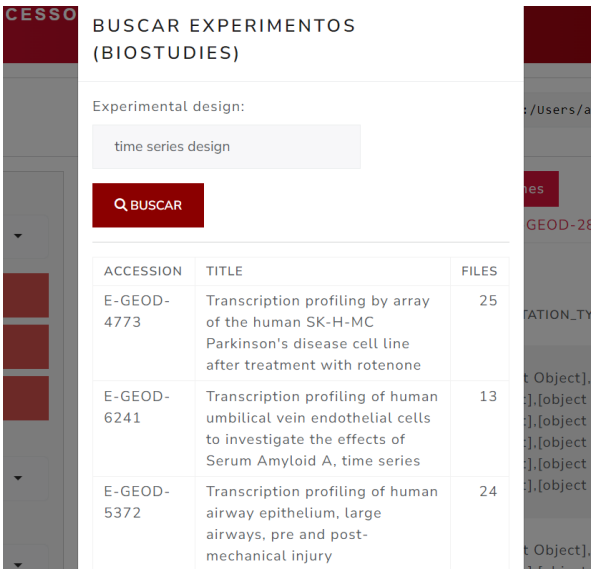


Figura 8.13: Buscador de experimentos según característica .[ex]perimental design”.

Se ha integrado un selector de directorio base, lo que permite al usuario indicar fácilmente la ubicación en su sistema donde se encuentran almacenados los experimentos descargados y el archivo ejecutable del algoritmo. Esto facilita que la aplicación sea adaptable a diferentes entornos o rutas de trabajo.

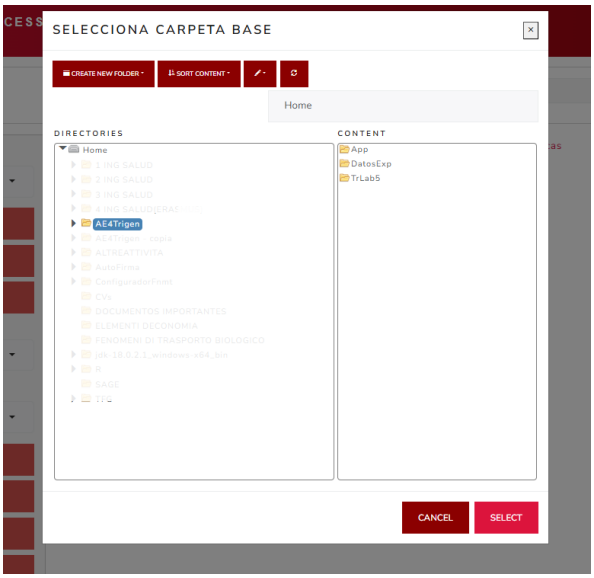


Figura 8.14: Selector directorio base de ArrayExpress Processor for TriGen algorithm

Además, el selector de experimentos se puebla automáticamente con los identificadores de los experimentos que ya han sido descargados previamente en el directorio seleccionado.

Por otro lado, tanto el selector de organismo como el selector de conjunto de anotación han sido vinculados dinámicamente a la API pública de PANTHER DB. En tiempo de ejecución, se realiza una llamada a esta API para obtener el listado actualizado de organismos y conjuntos de anotación disponibles.

8.5.1. Manual de despliegue y guía de uso.

Para el despliegue de la aplicación, únicamente es necesario tener R instalado en el sistema y contar con conexión a Internet. El desarrollo se ha realizado utilizando la versión R–4.4.3. No se han llevado a cabo pruebas de compatibilidad con otras versiones de R.

Se ha desarrollado un archivo `.bat` básico con el objetivo de facilitar el lanzamiento rápido y sencillo de la aplicación. Este archivo permite al usuario ejecutar la interfaz con un doble clic. A continuación, se explica detalladamente el despliegue de la herramienta:

1. Descargar y descomprimir el archivo `.ZIP`.

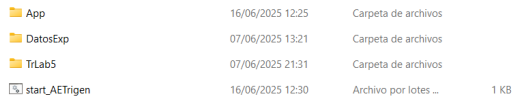


Figura 8.15: Paso 1. Descargar y descomprimir.

2. Hacer doble clic en el archivo `start_AETrigen.bat`.

3. Se abrira el terminal. Introducir la ruta al archivo ejecutable `Rscript.exe` en tu sistema.

Figura 8.16: Paso 3. Introducir ruta al archivo ejecutable `Rscript.exe`.

4. Una vez ya en la interfaz gráfica, seleccionar el directorio base(la carpeta de descarga descomprimida).

5. Tras ello, hacer uso en el orden lógico de trabajo:

   a) Descarga del experimento.

   b) Generación de los recursos.

   c) Ejecución del algoritmo.

6. Utilizar todas las funcionalidades de la herramienta como se desee.

9. Mejoras y líneas de trabajo futuras

A modo de conclusión y reflexión propia, conforme he ido desarrollando todo este proyecto se me han ocurrido varias ideas y mejoras que podrían implementarse con un poco mas de trabajo y tiempo en el futuro para ampliar la utilidad de esta herramienta y que dejo como sugerencia como posibles Trabajos Finales de Grado:

1. Generalizar aún mas el preprocesamiento permitiendo completar automáticamente metadatos faltantes, como los puntos de tiempo o las etiquetas de muestra, cuando no estén claramente definidos en los archivos SDRF.

2. Agregar una visualización previa de los metadatos del experimento antes de su descarga.

3. Permitir ajustar todos los parámetros del algoritmo TriGen directamente desde la interfaz, de forma que el usuario pueda modificar fácilmente los valores de población, generaciones o pesos de las métricas sin editar manualmente el archivo de configuración.

4. Poder generar varias ventanas para comparar los resultados de múltiples experimentos simultáneamente.

5. La incorporación de técnicas de inteligencia artificial o sistemas multiagente, que permitan guiar o ejecutar todo el flujo de trabajo mediante lenguaje natural.

6. Integrar los demás servicios que ofrece PantherDB, como la exploración de vías, análisis de evolución o comparaciones entre especies.

7. Explorar el despliegue en un servidor Linux propio, facilitando así el acceso y uso de la herramienta por parte de otros usuarios. Se intentó desplegar la aplicación Shiny en la plataforma `https://www.shinyapps.io/` para facilitar su acceso desde cualquier navegador, pero surgieron problemas de compatibilidad con algunas dependencias relacionadas con BioConductor.

Bibliografía

[1] Supratim Choudhuri. *Bioinformatics for Beginners: Genes, Genomes, Molecular Evolution, Databases and Analytical Tools*. Academic Press, Amsterdam, 2014. ISBN 9780124104716.

[2] John Walker. Frederick sanger (1918–2013). *Nature*, 505(7481):27, 2014. doi: 10.1038/505027a. Obituary commentary.

[3] Jin Xiong. *Essential Bioinformatics*. Cambridge University Press, Cambridge, 2006. ISBN 9780521600828.

[4] Elmer A. Fernández, Mariano J. Alvarez, Osvaldo L. Podhajcer, and Gustavo Stolovitzky. Genómica funcional: En busca de la función de los genes. *Actas de la Academia Nacional de Ciencias (Córdoba, Argentina)*, 13:64–75, 2007. Basado en el curso "Genómica Funcional" del evento Biomat 2005.

[5] David A. Wheeler, Maithreyan Srinivasan, Michael Egholm, Yufeng Shen, Lei Chen, Amy McGuire, Wen He, Yi-Ju Chen, Vinod Makhijani, G. Thomas Roth, Xavier Gomes, Karrie Tartaro, Faheem Niazi, Cynthia L. Turcotte, Gerard P. Irzyk, James R. Lupski, Craig Chinault, Xing zhi Song, Yue Liu, Ye Yuan, Lynne Nazareth, Xiang Qin, Donna M. Muzny, Marcel Margulies, George M. Weinstock, Richard A. Gibbs, and Jonathan M. Rothberg. The complete genome of an individual by massively parallel dna sequencing. *Nature*, 452(7189):872–876, 2008. doi: 10.1038/nature06884.

[6] Universidad Rey Juan Carlos. *RNA-Seq: Introducción y análisis bioinformático con R y Bioconductor*. Universidad Rey Juan Carlos, 2022. Guía educativa para análisis transcriptómico.

[7] Miroslav Blumenberg. Introductory chapter: Transcriptome analysis. In *Transcriptome Analysis*. IntechOpen, 2019. doi: 10.5772/intechopen.85980. Capítulo introductorio de libro de acceso abierto.

[8] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009. doi: 10.1038/nrg2484.

[9] Instituto de Biología Molecular y Celular de Rosario (IBR-CONICET). *Transcriptómica*, 2020. Material educativo sobre análisis transcriptómico.

[10] Laurent Gautier, Leslie Cope, Benjamin M. Bolstad, and Rafael A. Irizarry. affy—analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004. doi: 10.1093/bioinformatics/btg405.

[11] Oscar M. Rollano Peñaloza and Patricia Mollinedo Portugal. Análisis bioinformático de arn-seq con una perspectiva para bolivia. *Revista Boliviana de Química*, 34(2):50–55, 2017. ISSN 0250-5460.

[12] John Morris, Helen Parkinson, Ugis Sarkans, Tim Rayner, and Norman Morrison. Mage-tab specification version 1.1. Technical report, Functional Genomics Data Society (FGED), May 2011. Standard specification for representing microarray and sequencing experiment metadata.

[13] John M. Gottman, Richard M. McFall, and Jean T. Barnett. Design and analysis of research using time series. *Psychological Bulletin*, 72(4):299–306, 1969. doi: 10.1037/h0028431.

[14] Priyakshi Mahanta, Hasin Ahmad Ahmed, D. K. Bhattacharyya, and Jugal K. Kalita. Triclustering in gene expression data analysis: A selected survey. In *Proceedings of the 2011 IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW)*, pages 258–265. IEEE, 2011. doi: 10.1109/BIBMW.2011.6112414.

[15] I. P. Androulakis, E. Yang, and R. R. Almon. Analysis of time-series gene expression data: Methods, challenges, and opportunities. *Annual Review of Biomedical Engineering*, 9:205–228, 2007. doi: 10.1146/annurev.bioeng.9.060906.151904.

[16] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm – a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con)*, pages 380–384, Noida, India, 2019. IEEE. ISBN 978-1-7281-0211-5. doi: 10.1109/COMITCON.2019.8862450.

[17] David Gutiérrez-Avilés, Cristina Rubio-Escudero, Francisco Martínez-Álvarez, and José C. Riquelme. Trigen: A genetic algorithm to mine triclusters in temporal gene expression data. *Neurocomputing*, 132:42–53, 2014. doi: 10.1016/j.neucom.2013.03.061.

[18] Purvesh Khatri and Sorin Drăghici. Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, 21(18):3587–3595, 2005. doi: 10.1093/bioinformatics/bti565.

[19] Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004. doi: 10.1186/gb-2004-5-10-r80.

[20] Seth Falcon, Martin Morgan, and Robert Gentleman. *An Introduction to Bioconductor's ExpressionSet Class*. Bioconductor Project, 2006. Revisado en febrero de 2007.

[21] Marc Carlson. *AnnotationDbi: Introduction To Bioconductor Annotation Packages*. Bioconductor Project, 2025. Versión del 15 de abril de 2025.

[22] Audrey Kauffmann, Ibrahim Emam, Michael Schubert, and Jose Marugan. *ArrayExpress: Access the ArrayExpress Collection at EMBL-EBI Biostudies and build Bioconductor data structures: ExpressionSet, AffyBatch, NChannelSet*. Bioconductor, 2025.

[23] Rafael A. Irizarry, Bridget Hobbs, Francois Collin, Yasmin D. Beazer-Barclay, Kristen J. Antonellis, Uwe Scherf, and Terence P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, 2003. doi: 10.1093/biostatistics/4.2.249.

[24] David Gutiérrez-Avilés. Github profile: davgutavi. https://github.com/davgutavi.

[25] David Gutiérrez-Avilés and Cristina Rubio-Escudero. Trlab: Una metodología para la extracción y evaluación de patrones de comportamiento de grandes volúmenes de datos biológicos dependientes del tiempo. *Neurocomputing*, 2017.

[26] Moosa Mahmoudi. rbioapi: R interface to the panther and other bioinformatics apis – file: panther.r. https://github.com/moosa-r/rbioapi/blob/master/R/panther.R, 2020.

[27] PANTHER Database. Panther api openapi specification, 2025.