Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Xochimilco

## DIVISIÓN DE CIENCIAS BIOLÓGICAS Y DE LA SALUD
## DEPARTAMENTO DE SISTEMAS BIOLÓGICOS

## LICENCIATURA EN QUÍMICA FARMACÉUTICA BIOLÓGICA

## INFORME DE ACTIVIDADES DE SERVICIO SOCIAL:

# Desarrollo de una herramienta bioinformática interactiva en lenguaje Python para visualizar asociaciones biológicas

**Correspondiente al proyecto genérico:**

Obtención de materias primas, principios activos, medicamentos y productos biológicos.

**Etapa:** Diseño y desarrollo de productos biológicos por métodos biotecnológicos o de ingeniería genética

PRESENTA:

**Herrera Jiménez Christian**
Matrícula: 2173082661

**Lugar de realización:** Laboratorio N-104, Edificio N. Departamento de Sistemas Biológicos, Universidad Autónoma Metropolitana Unidad Xochimilco, con Dirección: Calzada Del Hueso 1100. Col. Villa Quietud, Alcaldía de Coyoacán, C.P 04960, Ciudad de México, México.

**Periodo:** 17 noviembre 2021 al 17 mayo 2022

Ciudad de México, Agosto 2023

# ÍNDICE

**Introducción** ……03

**Marco teórico** ……..04

Tecnologías ómicas y sus principales herramientas ……04

Lenguajes de programación ……05

Jupyter Notebook ……06

Diagrama de acordes ……06

Curvas de Bézier ……08

**Justificación** ……08

**Objetivo general** ……09

**Objetivos específicos** ……09

**Desarrollo experimental** ……09

**Resultados y discusión** ……10

Librerías y módulos empleados en la elaboración del código ……10

Funciones creadas para la optimización del código ……11

Definición de variables con información para las conexiones ……15

Código para la visualización del gráfico ……17

Validación de la herramienta HIVAB ……20

**Metas y objetivos alcanzados** ……22

**Conclusiones** ……23

**Referencias Bibliográficas** ……24

**Resumen** ……30

## Introducción

Gracias a la secuenciación del genoma humano a finales del siglo XX, se realzó el interés por comprender la información presente en diversos conjuntos de moléculas, así como las interacciones entre ellas, dando lugar a las tecnologías ómicas (Bermúdez, 2020). La palabra ómica proviene del inglés y se refiere al estudio de un conjunto en su totalidad (Tabas, 2018). En el año 1980, se acuñó el término "Tecnologías ómicas", al estudio de un conjunto de moléculas (Frigolet & Gutiérrez, 2017). Sin embargo, hoy en día es una disciplina científica y tecnológica que se desglosa de las diferentes especialidades que conllevan principalmente el estudio del código genético (Gordon, 2022), en donde se integran diversas ciencias y herramientas como la bioinformática, estadística, física, química y matemáticas con un fin científico (Serrano et al., 2020). El auge de las tecnologías ómicas se ha generado por el desarrollo de técnicas de alto rendimiento en el área de la biología, las cuales, son capaces de generar grandes volúmenes de datos que contienen información suficiente del sistema como para estudiarlo. No obstante, el análisis del procesamiento y la representación de resultados obtenidos de un gran volumen de datos es ineficiente cuando se utilizan los medios tradicionales (Tabas, 2018). A raíz de esta problemática, entra en juego el rol de la bioinformática, la cual, es una herramienta sumamente importante debido al desarrollo de herramientas informáticas como equipos analíticos, diversos softwares, bases de datos y lenguajes de programación, así como la aplicación de estos en la generación de conocimientos biológicos para comprender mejor los sistemas vivos (Meneses et al., 2011), permitiendo identificar y medir un gran número de datos, así como almacenar y representar dicha información (Frigolet & Gutiérrez, 2017), además, su principal tarea consiste en entender las correlaciones, las estructuras y los patrones en los datos biológicos (Meneses et al., 2011). Es así, que la visualización de este tipo de datos ahora suele representarse con gráficas logrando una mejor ejemplificación de las asociaciones o interacciones que los científicos de datos desean. Una herramienta de la biotecnología que ha destacado últimamente por su forma de representar dichos resultados es el diagrama de acordes, el cual, es un gráfico interactivo con un formato circular usualmente utilizado he implementado en

muchas disciplinas para identificar y analizar patrones en diferentes tipos de redes (Jalali, 2016). Los diagramas de acordes son construidos a partir de matrices/listas de adyacencia (Koutrouli et al., 2020) donde a cada valor se le llama nodo y estos a su vez, están organizados y distribuidos a lo largo de un círculo, donde los nodos van a estar conectados entre sí mediante el uso de arcos o curvas de Bézier con el fin de crear las conexiones (Ribecca, 2017).

Por consiguiente, en este trabajo se planteó el desarrollo de una interfaz gráfica bioinformática e interactiva en lenguaje Python para visualizar asociaciones/interacciones biológicas entre datos provenientes de tecnologías Ómicas.

## Marco teórico

### Tecnologías ómicas y sus principales herramientas

Con el paso de los años los seres humanos hemos ido evolucionando y mejorando nuestra forma de estudiar la salud. Un ejemplo que marcó un antes y un después en las últimas décadas fue la innovación de las tecnologías ómicas, que desde los años 80´s, se empezaba a emplear para referirse al estudio de un conjunto de moléculas (Frigolet & Gutiérrez, 2017), pero no fue hasta el año 1990 que este término tuvo un alto renombre gracias al Proyecto Genoma Humano (PGH), donde se llevó a cabo la secuenciación del 99,99% del genoma humano, la identificación de genes portadores de enfermedades, así como parte del genoma del ratón (Sardi, 2020). Al cumplir el propósito de dicho proyecto, se implementó lo que hoy en día conocemos como tecnologías ómicas que son la integración de diversas ciencias y herramientas como las matemáticas, estadística, física, química, bioinformática, entre otras, con un fin científico (Serrano et al., 2020), y gracias a ellas, hemos podido determinar los genomas de una gran variedad de organismos, así como identificar y definir biomarcadores moleculares, describir nuevas funciones de genes no codificantes, identificar modificaciones postranscripcionales y postraduccionales, tener un mejor conocimiento sobre las rutas metabólicas e identificar nuevas vías, así como caracterizar las interacciones ADN-proteínas y ARN-proteínas, entre otras (Quiroga, 2016).

Como se mencionó anteriormente, la bioinformática es una herramienta sumamente importante de las tecnologías ómicas dado que el desarrollo de la tecnología mediante la generación de equipos analíticos, el desarrollo de diversos software, bases de datos y lenguajes de programación nos han permitido identificar y medir un gran número de moléculas, así como almacenar una gran cantidad de información, lo cual nos ha llevado a tener una mejor comprensión de los sistemas biológicos complejos (Frigolet & Gutiérrez, 2017).

**Lenguajes de programación**

Para el análisis bioinformático o la creación de nuevas herramientas es necesario utilizar distintos lenguajes de programación. Entre los cuales sobresalen Python, Java, C, Perl, R, entre otros. Todos estos lenguajes tienen ventajas de manera general, una de ellas es que permiten la colaboración abierta al proporcionar una plataforma central para que los investigadores desarrollen y mejoren metodologías de manera cooperativa y realicen análisis de datos, proporcionando un medio para la difusión transparente de un estudio o producto terminado. Sin embargo, en la actualidad, Python y R se han convertido en lenguajes especialmente populares en la ciencia de datos debido a la disponibilidad de bibliotecas de código completas, robustas y bien documentadas (módulos/paquetes). Muchos paquetes estadísticos y de aprendizaje automático están disponibles para estos lenguajes, además del hecho que presentan un fuerte apoyo por su comunidad para el análisis de datos (Mendez et al., 2019).

R es un software libre enfocado principalmente, para al análisis estadístico y gráfico, además dicho análisis se puede realizar de una manera reproducible, pero tiene la desventaja de que no es tan fácil de manejar como otros lenguajes y su principal uso es el análisis de datos (Català, 2017; Riquelme, 2018). Por otro lado, Python es un lenguaje de programación con paradigmas múltiples, compatible con todos los sistemas operativos y fácil de usar, lo cual lo convierte en una muy buena herramienta ya que, no se encierra en un único enfoque, sino que su versatilidad lo puede llevar a realizar aplicaciones web, analizar datos, automatizar operaciones, entre un sinfín de opciones (Challenger et al., 2014).

## Jupyter Notebook

Project Jupyter es un proyecto para desarrollar software de código abierto sin fines de lucro, surgido a partir de la evolución del Proyecto IPython en 2014 con el fin de admitir la ciencia de datos interactiva y la computación científica en todos los lenguajes de programación (Jupyter, 2022). Su producto principal es Jupyter Notebook, el cual permite una integración perfecta de código, texto narrativo y figuras en un solo documento ejecutable y editable en vivo, registrado en el formato de notación de objetos JavaScript (JSON) de estándar abierto e independiente del idioma. Los cuadernos se pueden escribir en un solo lenguaje de programación o en una combinación de varios lenguajes (Mendez et al., 2019).

Jupyter Notebook es el sistema más utilizado para la programación alfabetizada interactiva, fue diseñado para hacer que el análisis de datos sea más fácil de documentar, compartir y reproducir. El sistema se lanzó en 2013 y hoy en día hay más de 1 millón de portátiles en GitHub. Jupyter Notebook además de Python, admite una variedad de lenguajes de programación, como Julia, R, Javascript y C. (Pimentel et al., 2019). Actualmente se ha utilizado para computar, presentar, discutir y difundir hallazgos científicos, permitiendo que el registro y la compresión de los datos sea más sencilla (J. Wang et al., 2020).

Para ejecutar Jupyter Notebook y el código fuente asociado, el usuario debe descargar y ejecutar una copia local de los archivos, o cargar y ejecutar el notebook "en la nube". Por lo tanto, el proceso de habilitar el uso práctico de este recurso compartido puede requerir un nivel de experiencia computacional que puede ser un impedimento para los usuarios ocasionales y restringir la adopción por parte de científicos curiosos de datos no expertos (Mendez et al., 2019).

## Diagrama de acordes

En la mayoría de lenguajes de programación se pueden realizar gráficos para visualizar interacciones, uno de ellos es el diagrama de acordes, el cual es un gráfico circular que permite observar interacciones/intersecciones de una manera elegante y objetiva. El diagrama de acordes fue desarrollado por Martin Krzywinski

(Wang, 2018), quien popularizó este tipo de gráfico en el paquete de software Circos, el cual está basado en d3.js., y es una biblioteca de Javascript para construir gráficos interactivos con un diseño circular (Peterson, 2022).

El diagrama de acordes cuenta con 3 componentes principales (figura 1):

- *"Source"* (fuente): Son las entidades o nodos que marcan el inicio de una asociación, variable independiente.
- "*Target*" (objetivo): Son las entidades o nodos que marcan el final de una asociación, variable dependiente.
- "*Connection*" (conexión): Es la unión o asociación que existe entre el "*source*" y el "*target*".

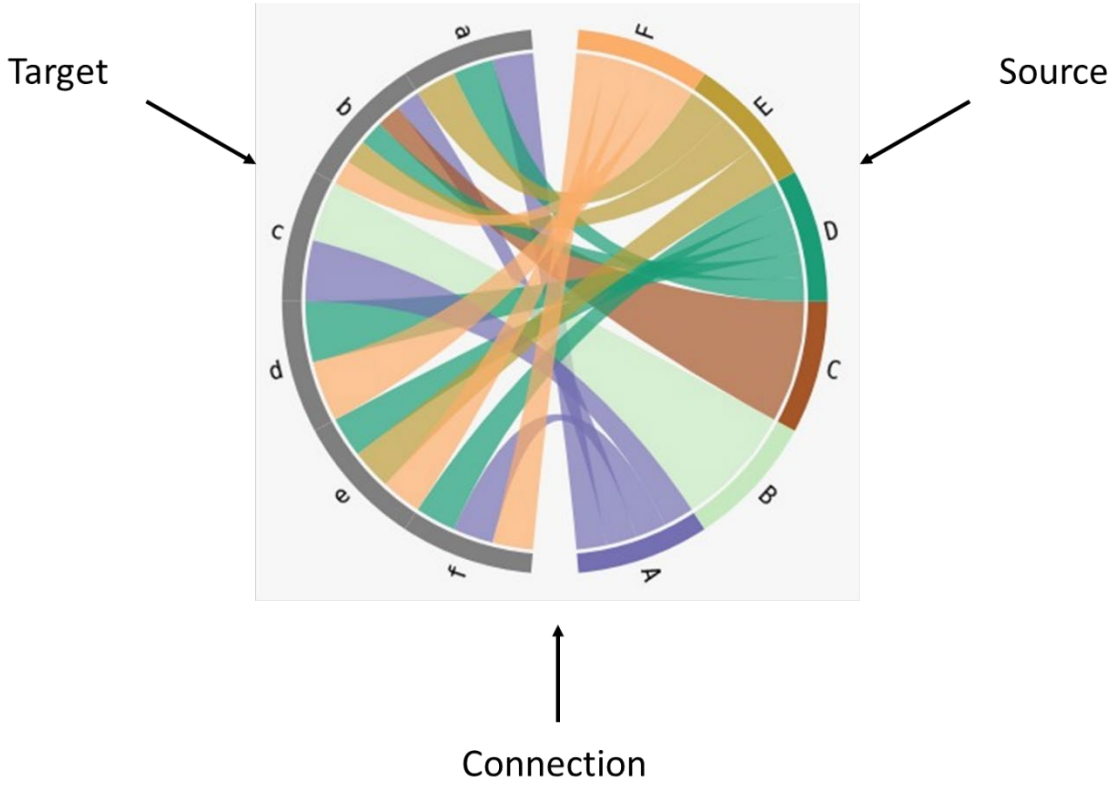



Figura 1. Esquema de un diagrama de acordes.

Este tipo de diagrama visualiza las interrelaciones mediante vértices o conexiones entre entidades designadas como nodos. Los nodos se organizan a lo largo de un círculo con las relaciones entre los puntos conectados entre sí mediante el uso de

arcos o curvas de Bézier con el fin de crear las conexiones entre los nodos. Esto hace que los diagramas de cuerdas sean ideales para comparar las similitudes dentro de un conjunto de datos o entre diferentes grupos de datos (Ribecca, 2017).

**Curvas de Bézier**

Las curvas de Bézier son necesarias para la elaboración de los diagramas de acordes y están definidas por puntos de control donde pueden existir 2, 3, 4 o más puntos, en dichas curvas los puntos no siempre están en la curva, pero es importante resaltar que la curva siempre va a estar dentro de los puntos de control, además de que el orden de la curva es igual al número de puntos menos uno, por ejemplo, para dos puntos tenemos una curva lineal (línea recta), para tres puntos es una curva cuadrática (parabólica) y así sucesivamente. Las curvas de Bézier se utilizan principalmente en gráficos para dibujar formas, en animación CSS, entre otras funciones (Kantor, 2022).

**Justificación**

Los diagramas de acordes se pueden realizar en un sinfín de lenguajes de programación, pero en la mayoría existen ciertas complicaciones que hacen que la experiencia para el usuario pueda ser tediosa o incluso molesta, pero Python y R sobresalen entre los demás por las amplias bibliotecas, módulos y paquetes que poseen (Mendez et al., 2019). Sin embargo, la principal desventaja de R es su grado de complejidad, lo que lo hace un lenguaje de programación que no mucha gente lo sabe utilizar (Català, 2017), convirtiendo a Python en un lenguaje bastante aceptable para este campo. No obstante, los diagramas de acordes no son un tipo de gráfico muy habitual, debido a que las bibliotecas más comunes como Matplotlib y Seaborn () no han sido útiles para realizar este tipo de gráfico. En este sentido existen algunas librerías como chord, Bokeh, Plotly y Mne destinadas a la construcción de diagramas de acordes para Python, pero pueden demandar mucha memoria RAM, además de que el usuario necesita un conocimiento intermedio sobre programación y se necesitan instalar de manera independiente otros programas o herramientas e incluso puede haber casos donde se necesite el

entorno de una plataforma o sitio web específico para que se pueda emplear dicha herramienta (Holtz, 2018).

Por este motivo en el presente proyecto se plantea el desarrollo del programa bioinformático HIVAB (herramienta interactiva para la visualización de asociaciones biológicas) diseñado para que cualquier usuario con conocimientos nulos o avanzados de bioinformática pueda utilizarlo para fines académicos o científicos.

**Objetivo general**

- Desarrollar una interfaz gráfica bioinformática e interactiva en lenguaje Python para visualizar asociaciones biológicas.

**Objetivos específicos**

- Adquirir conocimientos básicos y avanzados del lenguaje de programación Python.
- Desarrollar una interfaz gráfica en lenguaje Python para visualizar interacciones entre datos provenientes de tecnologías Ómicas.
- Realizar simulaciones con datos preestablecidos para evaluar la funcionalidad de la interfaz gráfica.
- Validar el funcionamiento de programa con un estudio de caso ya publicado (datos biológicos).

**Desarrollo experimental**

El programa bioinformático HIVAB: herramienta interactiva para la visualización de asociaciones biológicas, se desarrolló en el lenguaje de programación Python (versión 3.6.7). Se utilizaron diversas extensiones de paquetes de código abierto que representan altos niveles de aceptación y soporte por parte de la comunidad entre los cuales se incluyen: Collections, Numpy (versión 1.19.5) para cálculos basado en matrices (asociaciones ordenadas en filas y columnas) y Matplotlib (versión 3.3.4) para la visualización de datos (Mendez et al., 2019). Cabe destacar que esta herramienta fue desarrollada en Jupyter Notebook (versión 6.4.6). Esta

herramienta bioinformática será validada con información biológica proveniente de algún enfoque Ómico de alguna publicación científica existente.

## Resultados y discusión

### Librerías y módulos empleados en la elaboración del código

Para el desarrollo de HIVAB, como se mencionó anteriormente se utilizaron las librerías Matplotlib y Numpy.

a) Librería Matplotlib

Matplotlib.path es un módulo bastante útil para tratar con las polilíneas utilizadas en Matplotlib, debido a que la mayoría de los dibujos o trazos vectoriales utilizan rutas en algún lugar de la estructura del dibujo (Hunter et al., 2022b).

Matplotlib.patches es un módulo que se emplea para crear regiones bidimensionales arbitrarias donde se puede definir el contorno de cualquier forma bidimensional que se desee, sin importar cuán compleja sea (Rohrer, 2020).

Matplotlib.colors es un módulo para convertir números o argumentos de color a RGB o RGBA. En este módulo también se incluyen funciones y clases para conversiones de especificación de color, así como para asignar números a colores en una matriz de colores 1-D llamada mapa de colores (Hunter et al., 2022ª).

Matplotlib.plyplot es una interfaz que proporciona una forma de graficar similar a MATLAB, además de estar diseñado principalmente para gráficos interactivos y casos simples de generación de gráficos programáticos (Hunter et al., 2020).

b) Librería Numpy

Numpy es una librería de Python para computación científica, de igual manera puede ser utilizado como un eficiente contenedor multidimensional de datos genéricos, otra característica importante es que cuenta con su propia estructura de datos incorporada llamada arreglo que es similar a la lista normal de Python en la cual se pueden almacenar y operar con datos de manera mucho más eficiente (Cardellino, 2021).

c) Función Counter a partir de Collections

El módulo Counter se utiliza para contar elementos de un objeto iterable en Python, donde almacena cada elemento iterable como una clave de diccionario de Python, debido a que los diccionarios de Python solo permiten claves únicas, no hay repetición (Nitesh, 2021).

**Funciones creadas para la optimización del código**

Antes de explicar las funciones tenemos que dejar en claro lo que se definió como segmentos, ya que a partir de este punto este término se estará empleando a lo largo del trabajo. El segmento está conformado por vértices y ocupa un espacio en el gráfico donde en este caso, puede representar a un "*source"* o un "*target"* (figura 2).

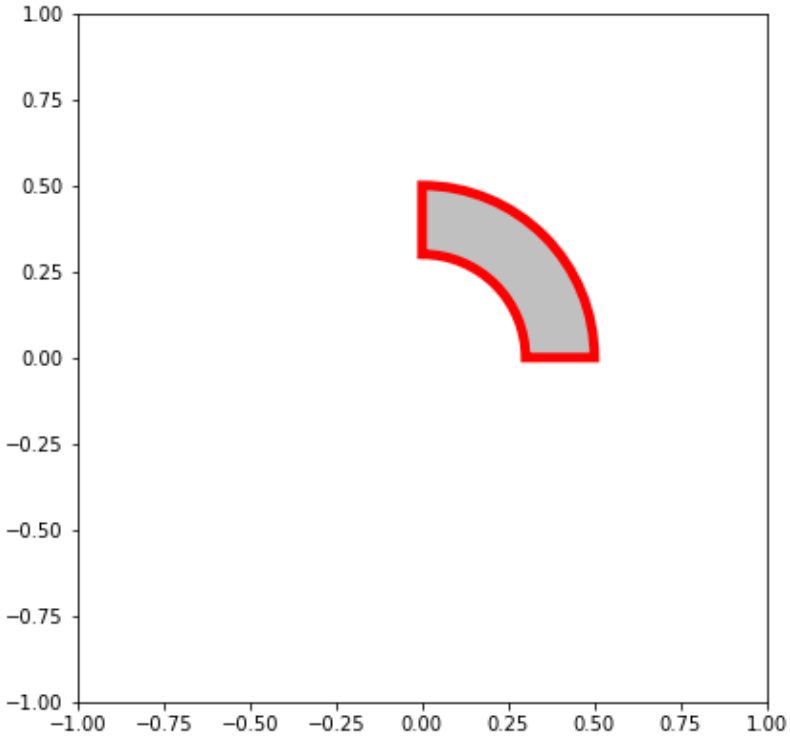



Figura 2. Representación de un segmento.

a) Función **seg**

La función definida como “seg” establece los parámetros de los segmentos tanto del “*source”* como del “*target”* que van a agregarse al gráfico para que a partir de estos se puedan colocar los propios segmentos, sus títulos y las conexiones entre el “source” y el “*target”*, por lo tanto, sin los segmentos y los parámetros de estos, el gráfico no podría tener un significado. Esta función recibe 7 argumentos de entrada: radio, ángulo inicial, ángulo final, color del segmento, ancho del segmento, tamaño del borde del segmento y color del borde del segmento (figura 3).

```
def seg(r = 0.5,
        ai = 50,
        af = 70,
        c = 'limegreen',
        w = 0.2,
        l = 1,
        ec = 'black'):
    """
    Esta función sirve para establecer los parámetros de los segmentos
    tanto del source como del target que van a agregarse al gráfico
    """
    sour = mpatches.Wedge((0, 0), # Posición central (x , y), en este caso `x = 0`, `y = 0`
                          r, # radio
                          ai, # ángulo inicial
                          af, # ángulo final
                          width = w, # ancho del segmento
                          edgecolor = ec, # color del borde del segmento
                          facecolor= c, # color del segmento
                          lw  = l) # tamaño del borde del segmento
    return sour
```

Figura 3. Código de la función **seg**.

b) Función **angint**

La función que se definió como “angint” arroja las coordenadas del segmento dentro del plano cartesiano que se necesitan para conocer lo que se determinó como “ángulos internos” (figura 4A), Esto se debe a que un segmento cuenta con vértices externos e internos. Por lo tanto, la función angint nos facilita el trabajo ya que nos permite conocer los vértices o coordenadas internas del segmento (figura 4B) y así establecer conexiones entre “*sources*“ y “*targets”* como se describe posteriormente.

A)

```
def angint (entrada = ""):
    """
    Función que se encarga de definir los puntos internos de cualquier segmento (fuente)
    """
    angint = int(len(entrada.get_path().vertices[0:-2])/2) #Mediante el .get_path() podemos convertir las coordenas que
    #deseamos utilizar en formato de "arreglo"
    #.vertices[0:-2] nos permite delimitar los puntos con los que vamos a trabajar, además de que eliminamos aquellos puntos
    #que se repiten al principio y al final
    #/2 Aquí dividimos los puntos en 2 partes, una de estas partes son los puntos externos del segmento y la otra son los
    #puntos internos del segmento
    datos = entrada.get_path().vertices[0:-2][angint:] #datos definimos una variable la cual nos va a arrojar los puntos
    #internos, el cual es el propósito de la función
    #[angint:] Aquí nosotros establecemos que queremos que nos arroje los puntos que se encuentran a partir de la mitad hasta
    #el final de los puntos del segmento
    return (datos) #Esto sirve para que nos devuelva el valor de los puntos internos del segmento
```

B)

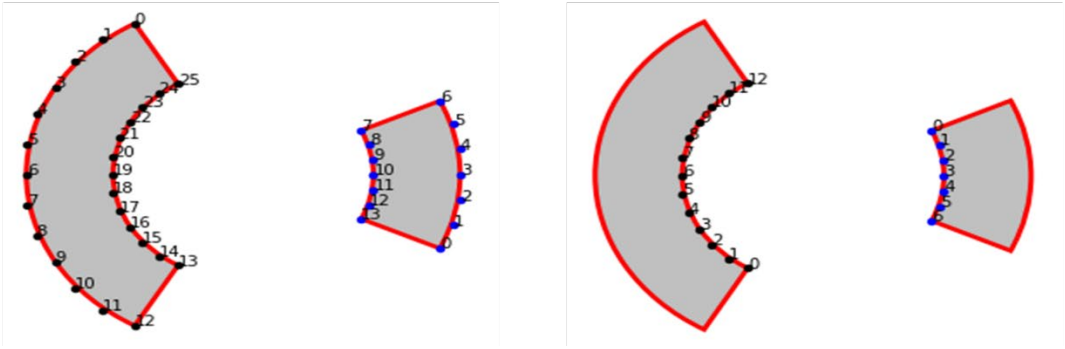
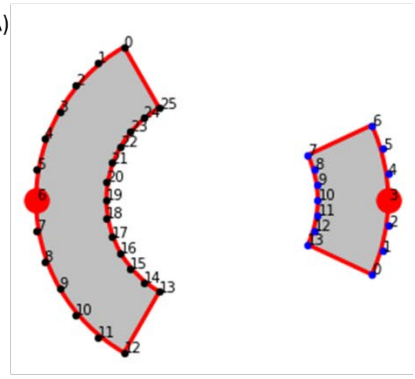


Figura 4. A) Código de la función **angint**. B) Comparación entre los vértices completos de un segmento con los vértices internos de un segmento.

c) Función **centro**

Esta función tiene el propósito de encontrar el centro de los puntos externos (vértices externos) de los segmentos (figura 5A) con el fin de guardar estas posiciones y agregar etiquetas o títulos a los segmentos y obtener una visualización ordenada (figura 5B).

A)



B)

```
def centro(entrada = ""):
    """
    Esta función sirve para conocer el centro de los puntos externos de los segmentos
    """
    angint = int(len(entrada.get_path().vertices[0:-2])/2) #la función angint nos permite conocer los puntos internos de los
    #segmentos
    datos = entrada.get_path().vertices[0:-2][0:angint] #En esta variable guardamos los puntos externos del segmento al
    #establecer que tome los puntos [0:angint]
    ext = int (len(datos)/2) #Esta variable guarda las coordenadas del punto medio de los puntos externos
    cenext = datos[ext] #Esta variable guarda el punto medio de los puntos externos
    return (cenext) #Aquí solicitamos que nos aparazca el punto medio de los puntos externos del segmento en el gráfico
```

Figura 5. A) Visualización del vértice central de la parte externa de un segmento. B) Código de la función **centro**.

d) Función **LOCATIONS**

La función definida como “LOCATIONS” se encarga de crear las conexiones entre el “*source”* y el “*target”*. Para crear estas uniones es necesario implementar las opciones “MOVETO”, “LINETO” y “CURVE3” de la función mpath, debido a que estas son las que van a definir los puntos que se necesitan para crear y darle forma a las conexiones (figura 6). De manera detallada el “MOVETO”, define las coordenadas que establecen el punto de partida de la conexión, mientras que el comando “LINETO” se encarga de dibujar o trazar líneas rectas desde el punto actual hasta el siguiente punto, por último, el comando “CURVE3” se encarga de dibujar o trazar una curva de Bézier cuadrática desde la posición actual, con el punto de control dado, hasta el punto final dado, en otras palabras, traza una curva de un punto X al punto Y dándole una inflexión hacia un punto Z creando una parábola (Hunter et al., 2021).

```python
def LOCATIONS(source, target): #Función que se encarga de crear las conexiones entre el source y el target
    #print(source, target)
    """
    source: arreglo con posiciones del origen
    target: arreglo con posiciones hacia el destino
    """
    Path = mpath.Path #--------variable que guarda en formato de arreglo los datos que se agregaran posteriormente
    path_data = [] #Se define una lista en la cual se van a ir añadiendo las coordenadas para las conexiones
    path_data.append(tuple([Path.MOVETO, tuple(source[0])])) #Coordenadas que definen el inicio de la conexion del
    #source
    for i in source[1:-1]: #Se crea un bucle para agregar los puntos siguientes de la conexión del source
        path_data.append(tuple([Path.CURVE3, tuple(i)])) #---------Aquí se define que la forma de la conexión debe
        #ser curveada
    path_data.append(tuple([Path.LINETO, tuple(source[-1])])) #--------
    path_data.append(tuple([Path.CURVE3, (0,  0)])) #Esto sirve para que la conexión tenga una curvatura orientada
    #hacia el centro del gráfico
    path_data.append(tuple([Path.LINETO, tuple(target[0])])) #Último punto que indica el fin de la conexión
    for j in target[1:-1]: #Se crea un bucle para agregar los puntos siguientes de la conexión del target
        path_data.append(tuple([Path.CURVE3, tuple(j)])) #---------Aquí se define que la forma de la conexión debe
        #ser curveada
    path_data.append(tuple([Path.LINETO, tuple(target[-1])])) #-----------
    path_data.append(tuple([Path.CURVE3, (0,  0)])) #Esto sirve para que la conexión tenga una curvatura orientada
    #hacia el centro del gráfico
    path_data.append(tuple([Path.LINETO, tuple(source[0])])) #Último punto que indica el fin de la conexión
    return path_data #Esto sirve para que nos muestre la conexión en el gráfico
```

Figura 6. Código de la función **LOCATIONS**.

## Definición de variables con información para las conexiones

Para el desarrollo del código se utilizó un ejemplo que demuestra asociaciones biológicas, específicamente, la relación entre procesos biológicos y proteínas que participan en dichos procesos (figura 7), para lo cual dichas interacciones se guardaron en una variable llamada neti.

```
neti = [("Endopeptidase inhibitor activity", "P06684:C5"),
("Endopeptidase inhibitor activity", "P29699:Ahsg"),
("Endopeptidase inhibitor activity", "P01027:C3"),
("Endopeptidase inhibitor activity", "Q61838:Pzp"),
("Endopeptidase inhibitor activity", "Q00897:Serpina1d"),
("Negative regulation of platelet", "O55042:Snca"),
("Negative regulation of platelet", "Q60994:Adipoq"),
("Negative regulation of platelet", "Q6P549:Inppl1"),
("Collagen-containing extracellular matrix", "Q8CIZ8:Vwf"),
("Collagen-containing extracellular matrix", "P19221:f2"),
("Collagen-containing extracellular matrix", "P29788:Vtn"),
("Collagen-containing extracellular matrix", "P07759:Serpina3k"),
("Collagen-containing extracellular matrix", "P29699:Ahsg"),
("Collagen-containing extracellular matrix", "O35639:Anxa3"),
("Collagen-containing extracellular matrix", "Q61838:Pzp"),
("Collagen-containing extracellular matrix", "P48036:Anxa5"),
("Collagen-containing extracellular matrix", "Q3U962:Col5a2"),
("Mitochondrion", "O55042:Snca"),
("Mitochondrion", "O88986:Gcat"),
("Mitochondrion", "Q8VCM5:Mul1"),
("Mitochondrion", "Q61578:Fdxr"),
("Mitochondrion", "P05064:Aldoa"),
("Mitochondrion", "P52503:Ndufs6"),
("Mitochondrion", "O08528:Hk2"),
("Mitochondrion", "Q8BJ03:Cox15"),
("Mitochondrion", "Q7TSQ8:Pdpr"),
("Mitochondrion", "Q3ULD5:Mccc2"),
("Mitochondrion", "Q99MR8:Mccc1"),
("Mitochondrion", "Q9CZ13:Uqcrc1"),
("Mitochondrion", "Q9D1G1:Rab1b"),
("Mitochondrion", "P55096:Abcd3"),
("Mitochondrion", "P05132:Prkaca"),
("Mitochondrion", "Q9CQS4:Slc25a46"),
```

Figura 7. Variable que guarda una lista con las relaciones (tuplas) entre procesos biológicos y proteínas.

Posteriormente, se guardaron en dos variables diferentes (orígenes y destinos) los valores únicos del "*source*" y del "*target*". En el "*source*" se van a localizar los procesos biológicos y en el "*target*" se van a encontrar las proteínas. Una vez que se realizó lo anterior, se guardaron 2 variables más que guardan el número de veces que se repite un elemento del "*source*" y del "*target*" respectivamente, dicho en otras palabras, nos indica en cuantas interacciones se encuentran implicados dichos elementos. Posteriormente se cuentan y se guardan en 2 variables más ("Lsource" y "Ltarget") los valores tanto del "*source*" como del "*target*" que establecen la longitud o el número de elementos únicos.

Para concluir esta parte, se guardaron 3 variables, una de ellas define el color que se le va a asignar a los segmentos y las conexiones o interacciones que tenga el "*source"* (proceso biológico) con el "*target"* (proteínas), mientras que las otras 2 se emplean para establecer el tamaño del título del "*source"* y del "*target"*.

**Código para la visualización del gráfico**

a) Parámetros del gráfico

Lo primero que se define cuando se construye un gráfico son los parámetros generales, por ejemplo, el tamaño, escalas de los ejes X y Y, el aspecto (en este caso la relación de aspecto está bloqueado), así como parámetros constantes como el radio del "source" y del "target", el ángulo que separa los segmentos del "source" de los del "target", el grosor que tendrán los segmentos tanto del "target" como del "source", así como el color y el espesor del espacio entre los segmentos y las conexiones (figura 8).

```
fig = plt.figure(figsize=(7, 7)) #Tamaño de la figura

ax = fig.add_axes([0, 0, 1, 1]) #Delimita el tamaño del gráfico
ax.set_xlim(-1, 1) #Modificación de las escalas pero el tamaño del recuadro sigue teniendo las mismas dimensiones
ax.set_ylim(-1, 1) #Modificación de las escalas pero el tamaño del recuadro sigue teniendo las mismas dimensiones

ax.set_aspect('equal', 'box') #Definición de la forma del gráfico

Fijo = 90 #Ángulo que separa el source del target
RS = 0.7 # radio del source
RT = 0.7 # radio del target
Ancho_seg = 0.05 #Indica el grosor que tendrán los segmentos
C = 'none' # color del espacio entre el segmento y la conexion
Wid = 0.1 # espesor del espacio entre el segmento y la conexion
```

Figura 8. Parámetros generales del gráfico.

b) Configuración de la zona "*target"*

Definimos 3 variables que nos ayudan a delimitar la zona "*target"* y sus segmentos. La primera (llamada des) establece el valor que se le agregará al ángulo de inicio (90 ° en este caso, este valor debe encontrarse entre 30° y 150°) de la zona "*target"* para poder diferenciar entre la zona "*target"* de la zona "*source"*. La siguiente (llamada n) define el sitio donde empezarán a localizarse los segmentos del "*target"* (obtenida a partir de la suma del valor de la variable Fijo + el valor de la variable des). Y la última (llamada sec) define los grados para cada segmento dentro de la

zona “*target*”. La variable sec contiene una constante la cual corresponde al cociente de los ángulos de la zona “*target*” y el número total de elementos únicos del “*target*” (figura 9).

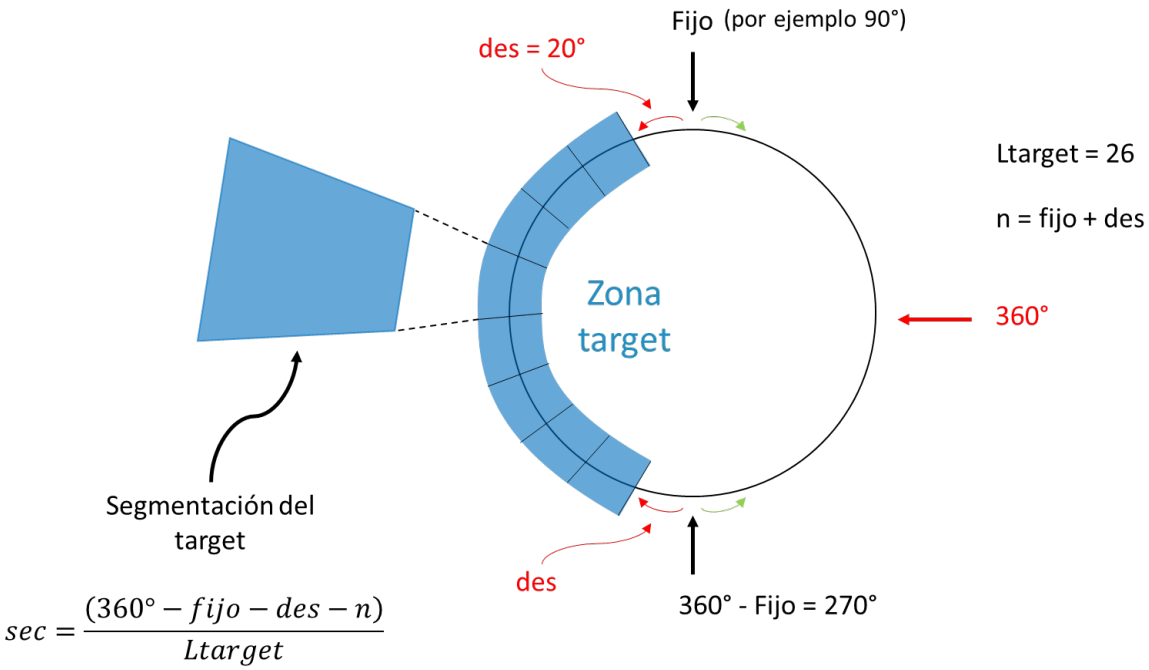



Figura 9. Configuración de la zona “*target*”.

Una vez calculado el valor de la variable sec, los elementos del “*target*” son iterados mediante un bucle for. En este punto se define la variable TARCONEC con una lista vacía a la cual se agregarán las etiquetas y los vértices de cada segmento de manera individual, y la subdivisión de los segmentos del “*target*”. Posteriormente, se implementó un bucle interno donde se definieron 3 variables, la primera de ellas guarda la posición en la cual se va a encontrar el inicio del segmento del “*target*”, la segunda variable nos indica el número de ángulos que va a ocupar cada subsegmento del “*target*” dependiendo del número de asociaciones que tenga ese “*target*” con el “*source*”, y la última variable nos va a indicar los parámetros necesarios para poder observar la subdivisión de los segmentos del “*target*”. Además, en este mismo bucle se añadieron los comandos para poder visualizar la subdivisión de los segmentos del “*target*” en el gráfico, los valores únicos del “*target*” (valores no repetibles) y la subdivisión de dichos segmentos que se añadirán a la

variable TARCONEC anteriormente mencionada. Al mismo tiempo que se adicionan los segmentos del "*target"* se capturan los vértices externos de los segmentos para obtener el punto central con la función centro y agregar los títulos. En esta misma sección se incluyeron 3 condicionales con la finalidad de colocar los títulos de los segmentos del "*target"* con los ángulos correctos, específicamente uno coloca los títulos de los segmentos si se encuentran entre los ángulos 0 y 90, el segundo los coloca si se encuentran entre los ángulos 90 y 270 y el tercero los coloca si se encuentra entre los ángulos 270 y 360 (figura 10).

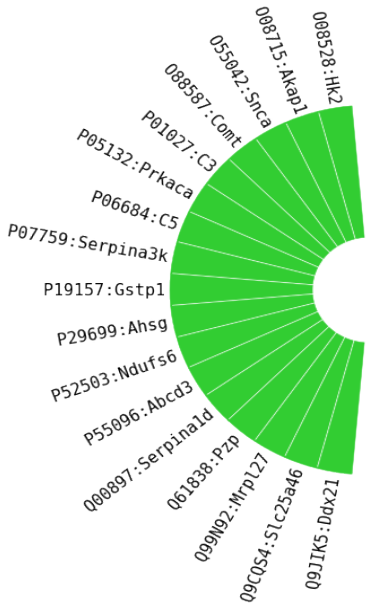



Figura 10. Ejemplificación de la subdivisión del segmento del "*target"*.

c) Configuración de la zona "*source"*

La zona "*source"* sigue los mismos pasos que la zona "*target"* solo que los nombres de las variables son diferentes para no sobrescribir la información dentro del código. Por lo tanto, de igual manera comenzamos definiendo 3 variables que nos ayudan a delimitar únicamente los segmentos del "*source"* donde se establece el valor que se le agregará al ángulo de inicio de la zona "*source"*, el sitio donde empezarán a localizarse los segmentos del "*source"* y el valor en grados que va a ocupar cada segmento la zona "*source*".

d) Configuración de las conexiones

Para poder realizar las conexiones y que estas se puedan visualizar en el gráfico, primero elaboramos una variable denominada conexiones en la cual hay una lista vacía donde se guardarán las conexiones entre el "*source"* y el "*target"*. Las conexiones se obtuvieron a partir de las variables SOURCONEC y TARCONEC, y al mismo tiempo que se construyen las conexiones se eliminan de las variables SOURCONEC y TARCONEC con el fin de evitar redundancias. Sucesivamente se adicionó una variable que guarda los resultados de la función "LOCATIONS" que crea las conexiones entre el "*source"* y el "*target"* (figura 11).

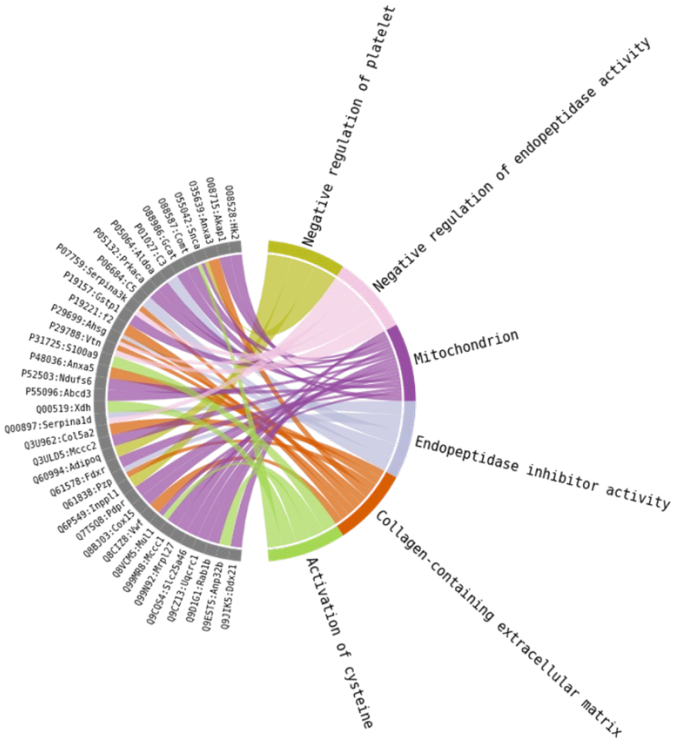



Figura 11. Visualización completa de un diagrama de acordes realizado con HIVAB.

**Validación de la herramienta HIVAB**

Además de la prueba presentada en la figura 11, se procedió a realizar la validación del código utilizando la información funcional de dos artículos científicos. En primer lugar, se utilizó la información del inciso A de la figura 6 de Zhong et al., (2020) en

su trabajo “Whole-genome sequence association of blood proteins in a longitudinal wellness cohort” donde se observa un diagrama con 50 proteínas significativas relacionadas/asociadas con la composición corporal (bioimpedancia grasa, bioimpedancia muscular, bioimpedancia ósea, peso, cintura e IMC) (figura 12). En segundo lugar, se usó la información de la figura 6 de Wang et al., (2021) en su trabajo “Extensive mitochondrial proteome disturbance occurs during the early stages of acute myocardial ischemia” donde se puede apreciar un análisis de enriquecimiento usando la Ontología Genética (“Gene Ontology”: GO) (figura 13). La información funcional de las dos publicaciones fue usada en HIVAB para comparar las visualizaciones.

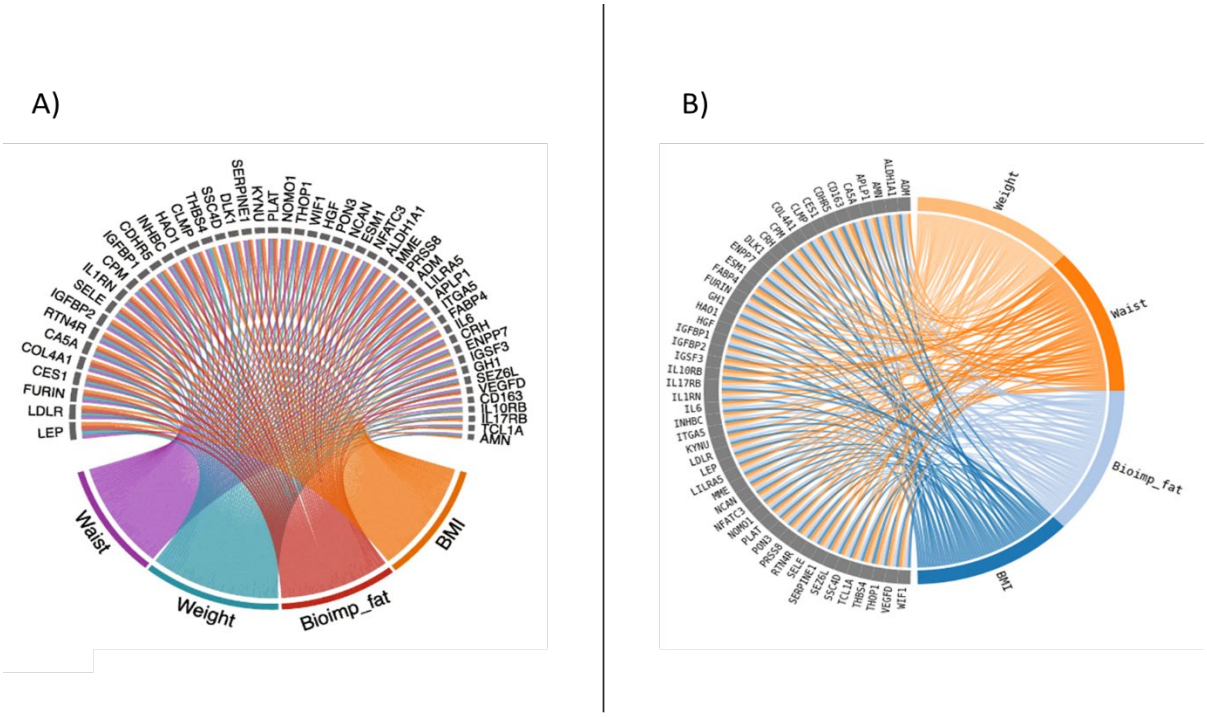



Figura 12. A) Diagrama de acordes de las 50 proteínas más significativas relacionadas con la composición corporal (bioimpedancia grasa, bioimpedancia muscular, bioimpedancia ósea, peso, cintura e IMC) de Zhong et al., (2020). B) Misma información de la figura A procesada con HIVAB.

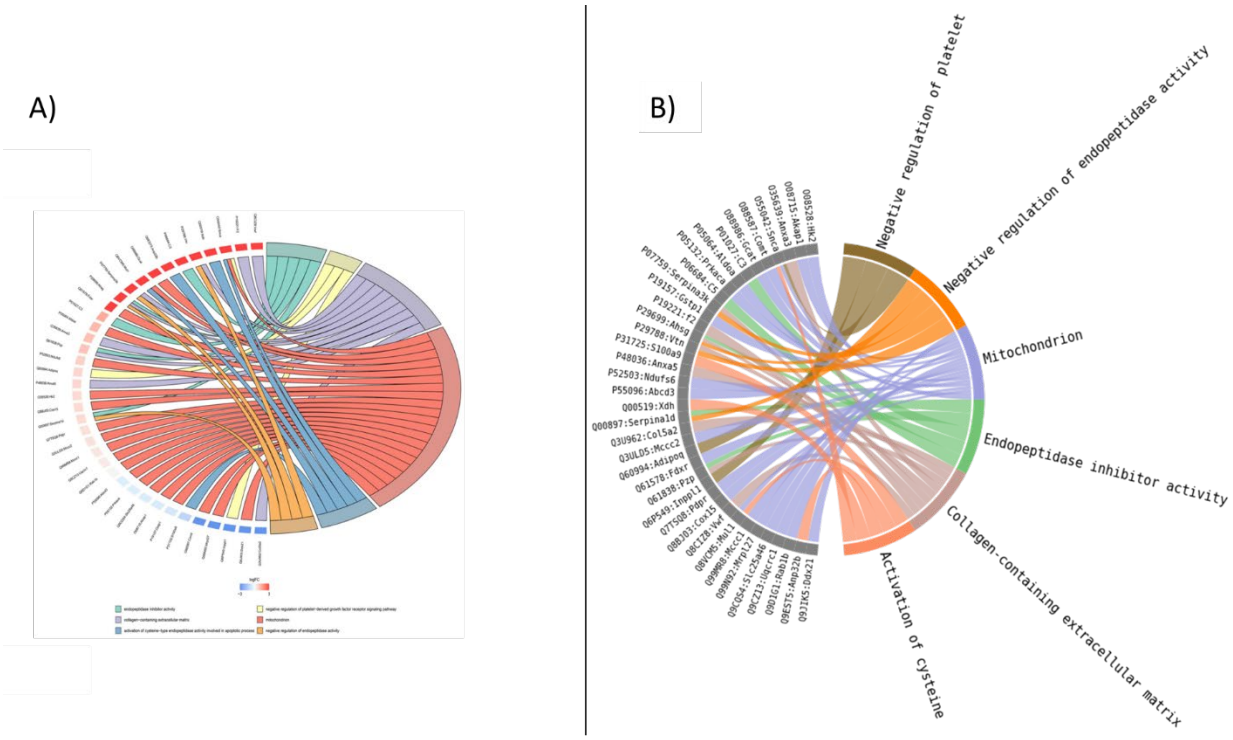



Figura 13. A) Diagrama de cuerdas del análisis de enriquecimiento GO de Wang et al., (2021). B) Misma información de la figura A procesada con HIVAB.

Después de realizar y analizar los gráficos construidos por la herramienta “HIVAB”, se determinó que esta puede replicar las visualizaciones de ambos trabajos de forma satisfactoria, adicionando un valor a la estética del diagrama, ya qué, cuenta con una amplia variedad de tonalidades para poder representar las conexiones de las interrelaciones, acomodando alfabética o numéricamente los segmentos tanto del “*source”* como del “*target”* ocasionando que la visualización del gráfico sea más familiar, además de que todos los parámetros del código se pueden ajustar para cualquier necesidad que requiera el usuario.

**Objetivos y metas alcanzadas**

- Se logra desarrollar una interfaz gráfica bioinformática e interactiva en lenguaje Python para visualizar asociaciones biológicas.
- Se logra adquirir conocimientos básicos y avanzados del lenguaje de programación Python.

- Se logra desarrollar una interfaz gráfica en lenguaje Python para visualizar interacciones entre datos provenientes de tecnologías Ómicas.
- Se logra realizar simulaciones con datos preestablecidos para evaluar la funcionalidad de la interfaz gráfica.
- Se logra validar el funcionamiento de programa con un estudio de caso ya publicado (datos biológicos).

## Conclusiones

Gracias a la constante evolución de las tecnologías ómicas como lo son la genómica, transcriptómica, proteómica, metabolómica, entre otras., se han tomado en cuenta nuevas opciones para comunicar sus resultados, implementando el uso de disciplinas como la estadística y la bioinformática con el objetivo de lograr un fin científico. Es por este motivo que actualmente los diagramas de acordes se están haciendo notar en el ámbito científico, ya qué, permiten visualizar diversas interrelaciones mediante vértices con nodos, los cuales se distribuyen y conectan entre sí a lo largo de un círculo en forma de arcos creando una serie de conexiones. Es por este motivo que se creó la herramienta HIVAB la cual no solo cumple con estos criterios, sino también logra adicionar una estructura elegante, concisa, heterogénea y amigable para que los científicos de datos puedan representar cualquier tipo de interacción que requieran.

## Referencias Bibliográficas

Bermúdez, M. (2020). Revisión: Impacto de la aplicación de técnicas “ómicas” en microalgas en el desarrollo de nuevas aplicaciones biotecnológicas. Análisis de las potenciales aplicaciones de “Nannochloropsis gaditana”. Consultado el 06 de junio del 2023 en: https://rodin.uca.es/handle/10498/23838

Cardellino, F. (2021). La guía definitiva del paquete NumPy para computación científica en Python. Consultado el 29 de julio del 2022 en: https://www.freecodecamp.org/espanol/news/la-guia-definitiva-del-paquete-numpy-para-computacion-cientifica-en-python/

Català, F. (2017). Uso de datos genómicos par a la identificación de miRNAs predictores de supervivencia en adenocarcinoma. Consultado el 10 de abril del 2022 en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/73266/6/fnayacTFM0117memoria.pdf

Challenger, I., Díaz, Y., & Becerra, R. (2014). El lenguaje de programación Python. Ciencias Holguín, 20(2), 1–13.

Frigolet, M., & Gutiérrez, R. (2017). Ciencias “ómicas”, ¿cómo ayudan a las ciencias de la salud? . Revista Digital Universitaria, 18(7), 1–16. Consultado el 02 de abril del 2022 en: https://doi.org/10.22201/codeic.16076079e.2017.v18n7.a3

Gordon, R. (2022). Inteligencia artificial: La caja de herramientas virtuales al servicio de la bioinformatica. Tecnociencia, 24(2), 48–65. Consultado el 08 de octubre del 2022 en: https://revistas.up.ac.pa/index.php/tecnociencia/article/view/3070

Holtz, Y. (2018). Chord diagram – from Data to Viz. Consultado el 03 de abril del 2022 en: https://www.data-to-viz.com/graph/chord.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & Matplotlib. (2020). matplotlib.pyplot — documentación de Matplotlib - 3.1.2. Consultado el 29 de julio del 2022 en: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & Matplotlib. (2021). Tutorial de rutas — documentación de Matplotlib - 3.5.0. Consultado el 29 de julio del 2022 en: https://matplotlib.org/3.5.0/tutorials/advanced/path_tutorial.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & Matplotlib. (2022a). matplotlib.colors — documentación de Matplotlib - 3.5.2. Consultado el 29 de julio del 2022 en: https://matplotlib.org/stable/api/colors_api.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & Matplotlib. (2022b). matplotlib.path — documentación de Matplotlib - 3.5.2. Consultado el 29 de julio del 2022 en: https://matplotlib.org/stable/api/path_api.html

Jalali, A. (2016). Supporting social network analysis using chord diagram in process mining. Lecture Notes in Business Information Processing, 261, 16–32. Consultado el 02 de abril del 2023 en: https://doi.org/10.1007/978-3-319-45321-7_2/COVER

Jupyter. (2022). Proyecto Jupyter. Consultado el 12 de febrero del 2023 en: https://jupyter.org/about

Kantor, I. (2022). Curva de Bézier. Consultado el 23 de junio del 2022 en: https://es.javascript.info/bezier-curve

Koutrouli, M., Karatzas, E., Paez-Espino, D., & Pavlopoulos, G. A. (2020). A Guide to Conquer the Biological Network Era Using Graph Theory. Frontiers in Bioengineering and Biotechnology, 8, 504360. Consultado el 02 de abril del 2022 en: https://doi.org/10.3389/FBIOE.2020.00034/BIBTEX

Mendez, K., Pritchard, L., Reinke, S., & Broadhurst, D. (2019). Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing. Metabolomics, 15(125), 1–16. Consultado el 10 de abril del 2022 en: https://doi.org/10.1007/S11306-019-1588-0/FIGURES/4

Meneses, C. A., Rozo, L. V, & Franco, J. (2011). Tecnologías bioinformáticas para el análisis de secuencias de ADN. Scientia et Technica Año XVI, 49.

Nitesh, K. (2021). How to Use the Counter Module in Python. Consultado el 29 de julio del 2022 en: https://linuxhint.com/counter-module-python/

Peterson, B. (2022). Generador gratuito de diagramas de acordes en línea. Consultado el 17 de enero del 2023 en: http://www.datasmith.org/2018/06/02/a-bold-chord-diagram-generator/

Pimentel, J., Murta, L., Braganholo, V., & Freire, J. (2019). A large-scale study about quality and reproducibility of jupyter notebooks. IEEE International Working Conference on Mining Software Repositories, 507–517. Consultado el 30 de abril del 2022 en: https://doi.org/10.1109/MSR.2019.00077

Quiroga, C. (2016). Las tecnologías «ómicas»: situación actual y desafíos futuros. Revista Argentina de Microbiología, 48(4), 265–266. Consultado el 05 de abril del 2022 en: https://doi.org/10.1016/J.RAM.2016.12.001

Ribecca, S. (2017). Chord Diagram. Consultado el 02 de abril del 2022 en: https://datavizcatalogue.com/methods/chord_diagram.html

Riquelme, T. (2018). Modelos estadísticos para el análisis integrativo de experimentos multi-ómicos. Consultado el 10 de abril del 2022 en: https://riunet.upv.es/handle/10251/114957

Rohrer, B. (2020). Patches in Matplotlib. Consultado el 29 de julio del 2022 en: https://e2eml.school/matplotlib_patches.html

Sardi, A. (2020). El impacto del Proyecto Genoma Humano y la discriminación genética: aspectos éticos, sociales y jurídicos. Revista de Bioética y Derecho, 48, 209–226. Consultado el 05 de abril del 2022 en: https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1886-58872020000100015

Serrano, O., Feria, H., & Marcheco, B. (2020). Conocimientos sobre las tecnologías ómicas y medicina personalizada en estudiantes de Medicina. EDUMECENTRO, 12(2), 59–75. Consultado el 08 de octubre del 2022 en: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2077-28742020000200059

Tabas, D. (2018). Herramientas eficientes para el análisis masivo de datos ómicos. In Tesis doctoral. Consultado el 02 de abril del 2022 en: https://eprints.ucm.es/id/eprint/49798/1/T40488.pdf

Wang, J., He, J., Fan, Y., Xu, F., Liu, Q., He, R., & Yan, R. (2021). Extensive mitochondrial proteome disturbance occurs during the early stages of acute myocardial ischemia. Experimental and Therapeutic Medicine, 23(1), 1–14. Consultado el 07 de junio del 2023 en: https://doi.org/10.3892/ETM.2021.11008

Wang, J., Kuo, T., Li, L., & Zeller, A. (2020). Assessing and Restoring Reproducibility of Jupyter Notebooks. 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, 138–149. Consultado el 30 de abril del 2022 en: https://doi.org/10.1145/3324884.3416585

Wang, X. (2018, September 12). A History of Chords Diagrams. Consultado el 30 de abril del 2022 en: https://studentwork.prattsi.org/infovis/labs/a-history-of-chords-diagrams/

Zhong, W., Gummesson, A., Tebani, A., Karlsson, M. J., Hong, M. G., Schwenk, J. M., Edfors, F., Bergström, G., Fagerberg, L., & Uhlén, M. (2020). Whole-genome sequence association analysis of blood proteins in a longitudinal wellness cohort. Genome Medicine, 12(53), 1–16. Consultado el 07 de junio del 2023 en: https://doi.org/10.1186/S13073-020-00755-0/FIGURES/6

Vo. Bo. DE LOS ASESORES

DR. JUAN ESTEBAN BARRANCO FLORIDO

DR. JESÚS EDUARDO ZÚÑIGA LEÓN

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Xochimilco

**DIVISIÓN DE CIENCIAS BIOLÓGICAS Y DE LA S ALUD DEPARTAMENTO DE SISTEMAS BIOLÓGICOS**

**LICENCIATURA EN QUÍMICA FARMACÉUTICA BIOLÓGICA**

**INFORME DE ACTIVIDADES DE SERVICIO SOCIAL:**

# Desarrollo de una herramienta bioinformática interactiva en lenguaje Python para visualizar asociaciones biológicas

**Proyecto genérico:** Obtención de materias primas, principios activos, medicamentos y productos biológicos.

**Etapa:** Diseño y desarrollo de productos biológicos por métodos biotecnológicos o de ingeniería genética

**Licenciatura en Química Farmacéutica Biológica**

**Alumno:** Herrera Jiménez Christian
**Matrícula:** 2173082661

**Lugar de realización:** Laboratorio N-104, Edificio N. Departamento de Sistemas Biológicos, Universidad Autónoma Metropolitana Unidad Xochimilco, con Dirección: Calzada Del Hueso 1100. Col. Villa Quietud, Alcaldía de Coyoacán, C.P 04960, Ciudad de México, México.

**Periodo:** 17 noviembre 2021 al 17 mayo 2022

## Resumen

El auge de las tecnologías ómicas se ha generado por el desarrollo de técnicas de alto rendimiento en el área de la biología, las cuales, son capaces de generar grandes volúmenes de datos que contienen información suficiente del sistema como para estudiarlo. No obstante, el análisis del procesamiento y la representación de resultados obtenidos de un gran volumen de datos es ineficiente cuando se utilizan los medios tradicionales (Tabas, 2018). A raíz de esta problemática, entra en juego el rol de la bioinformática, la cual, es una herramienta sumamente debido a que permite identificar y medir un gran número de datos, así como almacenar y representar dicha información (Frigolet & Gutiérrez, 2017). Una herramienta de la biotecnología que ha destacado últimamente por su forma de representar dichos resultados es el diagrama de acordes, el cual, es un gráfico interactivo con un formato circular usualmente utilizado he implementado en muchas disciplinas para identificar y analizar patrones en diferentes tipos de redes (Jalali, 2016). Los diagramas de acordes son construidos a partir de matrices/listas de adyacencia (Koutrouli et al., 2020) donde a cada valor se le llama nodo y estos a su vez, están organizados y distribuidos a lo largo de un círculo, donde los nodos van a estar conectados entre sí mediante el uso de arcos o curvas de Bézier con el fin de crear las conexiones (Ribecca, 2017), sin embargo, en la mayoría de éstos gráficos existen ciertas complicaciones que hacen que la experiencia para el usuario pueda ser tediosa o incluso molesta (Mendez et al., 2019).

Por consiguiente, en este trabajo se planteó el desarrollo de una interfaz gráfica bioinformática e interactiva en lenguaje Python para visualizar asociaciones/interacciones biológicas entre datos provenientes de tecnologías Ómicas, con el fin de que cualquier usuario con conocimientos nulos o avanzados de bioinformática pueda utilizarla para fines académicos o científicos.

Bibliografía

Frigolet, M., & Gutiérrez, R. (2017). Ciencias “ómicas”, ¿cómo ayudan a las ciencias de la salud? . Revista Digital Universitaria, 18(7), 1–16. Consultado el 02 de abril del 2022 en: https://doi.org/10.22201/codeic.16076079e.2017.v18n7.a3

Jalali, A. (2016). Supporting social network analysis using chord diagram in process mining. Lecture Notes in Business Information Processing, 261, 16–32. Consultado el 02 de abril del 2023 en: https://doi.org/10.1007/978-3-319-45321-7_2/COVER

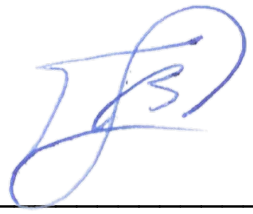
Koutrouli, M., Karatzas, E., Paez-Espino, D., & Pavlopoulos, G. A. (2020). A Guide to Conquer the Biological Network Era Using Graph Theory. Frontiers in Bioengineering and Biotechnology, 8, 504360. Consultado el 02 de abril del 2022 en: https://doi.org/10.3389/FBIOE.2020.00034/BIBTEX

Mendez, K., Pritchard, L., Reinke, S., & Broadhurst, D. (2019). Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing. Metabolomics, 15(125), 1–16. Consultado el 10 de abril del 2022 en: https://doi.org/10.1007/S11306-019-1588-0/FIGURES/4

Ribecca, S. (2017). Chord Diagram. Consultado el 02 de abril del 2022 en: https://datavizcatalogue.com/methods/chord_diagram.html

Tabas, D. (2018). Herramientas eficientes para el análisis masivo de datos ómicos. In Tesis doctoral. Consultado el 02 de abril del 2022 en: https://eprints.ucm.es/id/eprint/49798/1/T40488.pdf

Vo. Bo. DE LOS ASESORES



DR. JUAN ESTEBAN BARRANCO FLORIDO

DR. JESÚS EDUARDO ZÚÑIGA LEÓN