Bioinformática y Tratamiento de Datos (BIF)

Máster en Biotecnología Avanzada – Universidad Internacional de Andalucía & Universidad de Málaga

James R Perkins M. Gonzalo Claros Díaz
Juan Antonio Garcia Ranea Enrique Viguera Mínguez
Carlos Rodriguez Caso

2025-04-23

Table of contents

Prefacio

Este libro ha sido creado como un recurso complementario para la asignatura Bioinformática y Tratamiento de Datos (BIF), que forma parte del Máster en Biotecnología Avanzada, impartido conjuntamente por la Universidad Internacional de Andalucía (UNIA) y la Universidad de Málaga.

El contenido del libro se organiza en ocho capítulos, cada uno de los cuales corresponde a uno de los ocho temas fundamentales que se imparten en la asignatura. Este material no pretende sustituir a las clases ni a las presentaciones, sino servir como complemento. En cada tema, encontrarás explicaciones adicionales, ejemplos prácticos y demostraciones con código que te ayudarán a comprender mejor los conceptos presentados en clase.

Este recurso ha sido desarrollado con Quarto, una plataforma que permite integrar texto, código y resultados de manera fluida. Este formato favorece una experiencia de aprendizaje interactiva y reproducible, ayudando a conectar la teoría con la práctica en bioinformática.

Te animamos a experimentar con los ejemplos, modificar el código y explorar los datos. La bioinformática se aprende haciendo, y esperamos que este libro te acompañe y te apoye en ese proceso.

Chapter 1

Introducción a la terminal de Linux

Chapter 2

Introducción a R y redes

2.1 Repaso: descargar archivos

Antes de empezar con los análisis en R, repasaremos cómo obtener y preparar los datos. Aquí tienes los pasos esenciales para descargar y procesar un archivo desde UniProt utilizando la terminal.

2.1.1 Crear carpeta de trabajo

Abre tu terminal y ejecuta:

```
mkdir redes
cd redes
```

Esto crea una nueva carpeta llamada `redes` y cambia el directorio de trabajo a ella. Así mantenemos todos los archivos organizados.

2.1.2 Descargar archivo desde UniProt

Utiliza `wget` para descargar información sobre hexoquinasas:

```
wget 'https://rest.uniprot.org/uniprotkb/stream?query=hexokinase+AND+(reviewed:true)&format=tsv'
```

> ¿Qué hace este comando?
>
> - Consulta UniProt por proteínas que contienen el término **hexokinase** y están **revisadas** (reviewed:true).
> - Descarga los resultados en formato **TSV** (valores separados por tabulaciones).
> - Guarda el archivo como `hexokinase.tsv`.

2.2 Crear archivo de red para la práctica

Vamos a procesar ese archivo para obtener un archivo más sencillo que podremos usar para construir redes en R.

```
grep 'EC 2.7.1.1' hexokinase.tsv | cut -f 2,6 | tr '_' '    ' | cut -f 1,3 | cut -f 1,2
```

¿Qué hace este comando complejo?

1. `grep 'EC 2.7.1.1'`: selecciona solo las líneas con la enzima hexoquinasa (EC 2.7.1.1).
2. `cut -f 2,6`: extrae las columnas 2 y 6 (identificador y organismo).
3. `tr '_' '\t'`: reemplaza guiones bajos por tabulaciones.
4. `cut -f 1,3`: selecciona las columnas resultantes (id de proteína y especie).
5. `cut -f 1,2 -d ' '`: elimina cualquier texto adicional tras el segundo campo.
6. `> net`: guarda el resultado final en un archivo llamado `net`.

Este archivo contendrá dos columnas: el identificador de proteína y la especie correspondiente. Es un formato simple que podemos usar como entrada en R para crear redes de relaciones entre proteínas y organismos.

2.3 ¿Por qué hacemos esto?

La idea es tener un conjunto de datos **realista y sencillo** que nos permita centrarnos en aprender R y cómo manejar estructuras de red.

En esta primera sesión trabajaremos con `net`, el archivo que acabamos de generar. En la próxima sesión añadiremos más datos, como un archivo de **conteo**, y aplicaremos técnicas estadísticas y visualización en R para completar el análisis.

¿Sabías que…?

Este pequeño archivo simula una **red bipartita** entre proteínas y organismos. Más adelante exploraremos cómo convertir este tipo de datos en grafos utilizando R.

Ahora que tenemos nuestro fichero listo, podemos empezar de mirar el analisis con R.

2.4 ¿Qué es R?

R es un lenguaje y entorno de programación orientado al análisis estadístico y la representación gráfica de datos. Desarrollado originalmente por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, R ha evolucionado hasta convertirse en una herramienta esencial en campos como la bioinformática.

2.4.1 Componentes principales de R

- **Terminal propia**: Permite la ejecución directa de comandos y scripts.
- **Lenguaje de programación**: Ofrece estructuras y funciones para el desarrollo de algoritmos personalizados.
- **Gestor de ventanas gráficas**: Facilita la creación y visualización de gráficos y representaciones visuales de datos.

2.4.2 Iniciando la terminal de R

Para comenzar a utilizar R:

1. **Abrir la terminal**: Dependiendo de tu sistema operativo, accede a la terminal o consola.
2. **Iniciar R**: Escribe `R` y presiona Enter. Verás un prompt que indica que estás en el entorno de R.
3. **Salir de R**: Para finalizar la sesión, escribe `q()` y confirma si deseas guardar el espacio de trabajo.

```
> R
# Inicia la sesión de R

> q()
# Sale de R
# Save workspace image? [y/n/c]: n
```

En las terminales de sistemas Linux/bash, los comandos se ingresan en texto blanco sobre fondo negro. Al iniciar R, el fondo cambia a gris y el prompt cambia a `>`, indicando que ahora estás en el entorno de R.

2.5 Aprendiendo R: Variables

En R, las variables son contenedores que almacenan datos. Se utilizan para guardar y manipular información durante el análisis.

2.5.1 Creación y uso de variables

```
> texto <- 'ejemplo'
> texto
```

```
[1] "ejemplo"

> num <- 7
> num
[1] 7

> num_b <- 2
> num / num_b
[1] 3.5
```

- **Asignación**: Se utiliza el operador `<-` para asignar valores a las variables.
- **Cadenas de texto**: Se encierran entre comillas simples o dobles.
- **Operaciones**: R permite realizar operaciones matemáticas básicas con variables numéricas.

2.6 Trabajando con datos: Valores especiales y comparaciones

R maneja valores especiales que representan datos faltantes o indefinidos.

2.6.1 Valores especiales

```
> 5 + NA
[1] NA

> is.na(5 + NA)
[1] TRUE

> 10 + NULL
[1] NULL

> is.null(NULL)
[1] TRUE
```

- **NA**: Representa un valor faltante o no disponible.
- **NULL**: Indica que un objeto no contiene información.

2.6.2 Comparaciones

```
> 2 == 2
[1] TRUE

> 2 != 2
[1] FALSE
```

```
> 2 <= 3
[1] TRUE
```

- **==**: Verifica igualdad.
- **!=**: Verifica diferencia.
- **<=**: Verifica menor o igual.

2.7 Aprendiendo R: Variables compuestas

R permite trabajar con estructuras de datos más complejas, como los vectores, que son secuencias de elementos del mismo tipo.

2.7.1 Creación y acceso a vectores

```
> texto <- c('ab', 'cd', 'ef')
> texto
[1] "ab" "cd" "ef"

> texto[2]
[1] "cd"

> num <- c(3, 5, 7)
> num
[1] 3 5 7

> num[1]
[1] 3

> num / 2
[1] 1.5 2.5 3.5
```

- **Creación**: Se utiliza la función `c()` para combinar elementos en un vector.
- **Acceso**: Se accede a los elementos mediante corchetes `[]` y especificando la posición.
- **Operaciones**: Las operaciones matemáticas se aplican a cada elemento del vector.

2.7.2 Creación avanzada de vectores

```
> v <- 1:7
> v
[1] 1 2 3 4 5 6 7

> v <- rep(2, 23)
```

```
> v
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

> v <- rep(1:3, times = 2)
> v
[1] 1 2 3 1 2 3

> v <- rep(1:10, each = 2)
> v
[1] 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10

> v <- seq(10, 20, 2)
> v
[1] 10 12 14 16 18 20
```

- **Secuencias**: `1:7` genera una secuencia de 1 a 7.
- **Repeticiones**: `rep()` repite elementos según lo especificado.
- **Secuencias con paso**: `seq()` genera secuencias con un intervalo definido.

2.7.3 Operaciones con Vectores

Los vectores en R permiten realizar operaciones entre elementos de manera eficiente y fácil. Vamos a ver algunos ejemplos prácticos:

2.7.3.1 Sumar y multiplicar vectores

```
> v1 <- 1:5
> v2 <- rep(1, 5)
> v1 + v2 # Suma de dos vectores
[1] 2 3 4 5 6

> v1 + 1 # Sumar 1 a cada elemento del vector
[1] 2 3 4 5 6

> v1 * 2 # Multiplicar cada elemento del vector por 2
[1] 2 4 6 8 10
```

- **Operaciones element-wise**: Cuando realizamos operaciones entre vectores o entre un vector y un valor escalar, R realiza la operación de manera elemento a elemento.
- **Error de tamaños**: Si intentamos operar vectores de diferentes tamaños, como en el siguiente ejemplo, se genera un error. Esto ocurre porque R no sabe cómo emparejar los elementos de los vectores.

```
> v1 + c(1,7)
Error in v1 + c(1, 7) :
```

```
  longer object length is not a multiple of shorter object length
```

2.7.3.2 Operaciones lógicas con vectores

Los vectores también pueden ser usados en operaciones lógicas, que devuelven un vector de valores lógicos (`TRUE` o `FALSE`).

```
> v1 > 2 # Compara cada elemento con 2
[1] FALSE FALSE TRUE TRUE TRUE

> v1 == v2 # Verifica si los elementos de v1 son iguales a los de v2
[1] FALSE TRUE FALSE TRUE FALSE

> v1 != v2 # Verifica si los elementos de v1 son diferentes a los de v2
[1] TRUE FALSE TRUE FALSE TRUE
```

- **Operadores lógicos**:
  - `|`: Operador OR, devuelve un vector de valores lógicos para cada comparación.
  - `&`: Operador AND, devuelve un vector de valores lógicos.
  - `||` **y** `&&`: Versiones de OR y AND, pero devuelven un solo valor lógico en lugar de un vector. Estos operadores se utilizan para comparar solo el primer elemento de cada vector.

```
> (v1 > 2) | (v2 > 0) # Realiza OR lógico entre los vectores
[1] TRUE TRUE TRUE TRUE TRUE

> (v1 > 2) & (v2 > 0) # Realiza AND lógico entre los vectores
[1] FALSE TRUE FALSE TRUE FALSE

> (v1 > 2) || (v2 > 0) # OR lógico entre solo el primer elemento de cada vector
[1] TRUE

> (v1 > 2) && (v2 > 0) # AND lógico entre solo el primer elemento de cada vector
[1] FALSE
```

2.7.3.3 Formas de obtener elementos de un vector

Acceder a elementos específicos de un vector en R es muy sencillo, y existen diferentes formas de hacerlo:

```
> v1[3] # Accede al tercer elemento del vector
[1] 3

> v1[2:4] # Accede a los elementos de las posiciones 2 a 4
[1] 2 3 4
```

```
> v1[c(1,3)] # Accede a los elementos en las posiciones 1 y 3
[1] 1 3

> v1[c(TRUE, TRUE, FALSE, FALSE, FALSE)] # Usamos un vector lógico para acceder a eleme
[1] 1 2
```

- **Acceso directo**: Usamos los índices entre corchetes `[]` para acceder a un elemento o a un subconjunto del vector.
- **Acceso con vectores lógicos**: Se puede crear un vector lógico para seleccionar elementos específicos de un vector.

2.7.3.4 Formas avanzadas de obtener elementos de un vector

R ofrece métodos más avanzados para acceder a elementos en vectores. Aquí te mostramos algunos de ellos:

```
> z <- c(3, 6, 2, 2, 1, 0, 6)
> which(z > 3) # Devuelve las posiciones que cumplen la condición
[1] 2 7

> z[which(z > 3)] # Usamos las posiciones para extraer los elementos
[1] 6 6

> w <- c(2, 3)
> match(w, z) # Encuentra las posiciones de los elementos de w en z
[1] 3 1

> z[match(w, z)] # Extrae los elementos de z que coinciden con w
[1] 2 3
```

- **`which()`**: Devuelve los índices de los elementos que cumplen una condición lógica (por ejemplo, `z > 3`).
- **`match()`**: Devuelve los índices de las posiciones de los elementos de un vector en otro.

2.8 Aplicaciones en Bioinformática

En bioinformática, la manipulación de vectores y matrices es esencial para manejar grandes cantidades de datos, como secuencias genómicas o interacciones proteína-proteína. Por ejemplo, podemos usar vectores para almacenar información sobre expresiones génicas y realizar operaciones de análisis, como comparar los valores de expresión en diferentes condiciones experimentales o identificar genes diferencialmente expresados.

Ejemplo: Comparar la expresión génica en dos condiciones diferentes.

```
# Expresión génica en dos condiciones
expr_condition_1 <- c(5, 8, 3, 10, 6)
expr_condition_2 <- c(4, 7, 2, 9, 5)

# Identificar genes cuya expresión es mayor en la condición 1
expr_condition_1 > expr_condition_2
[1]  TRUE  TRUE  TRUE FALSE  TRUE
```

Aquí, estamos comparando los niveles de expresión génica entre dos condiciones, y el resultado nos indica qué genes tienen una mayor expresión en la primera condición. Esto es solo un ejemplo sencillo de cómo las operaciones con vectores pueden ser aplicadas en el análisis de datos biológicos.

2.8.1 Funciones sobre Vectores

En R, las funciones aplicadas a vectores nos permiten obtener estadísticas y realizar análisis rápidamente sobre los datos. Aquí algunos ejemplos útiles:

```
> num <- c(3, 5, 7, 2, 4, 9)

> length(num) # Devuelve la longitud del vector
[1] 6

> sum(num) # Suma de todos los elementos del vector
[1] 30

> mean(num) # Media de los elementos del vector
[1] 5

> max(num) # Valor máximo en el vector
[1] 9

> sd(num) # Desviación estándar del vector
[1] 2.582
```

- **Funciones estadísticas**: Estas funciones proporcionan información clave sobre la distribución de los datos en el vector.
- **R está orientado al cálculo y la estadística**, por lo que tiene potentes funciones para analizar conjuntos de datos de manera eficiente.

2.9 Evolución del Vector: La Matriz

Una matriz es similar a un vector, pero tiene dos dimensiones: filas y columnas. Es útil para representar datos más complejos, como matrices de interacción entre genes y muestras.

```
> m <- matrix(data = 1, nrow = 5, ncol = 4) # Crear una matriz 5x4 con todos los valore
> m <- matrix(1, 5, 4) # Forma abreviada
> dim(m) # Devuelve las dimensiones de la matriz
[1] 5 4

> m <- matrix(1:10, 10, 10) # Usamos un vector para rellenar la matriz
> t(m) # Transpone la matriz, intercambiando filas por columnas
```

- **Dimensiones de la matriz**: Las matrices tienen dos dimensiones, lo que permite representar diferentes aspectos de los datos (como genes y muestras en un estudio).
- **Operaciones en matrices**: Solo puede contener un tipo de dato (generalmente numérico).

2.9.1 Cómo Acceder a los Datos de una Matriz

Las matrices tienen una estructura bidimensional, lo que permite acceder a sus elementos de varias maneras, ya sea por filas, columnas o índices específicos.

```
> m[2,3] # Accedemos a un solo elemento en la fila 2, columna 3
[1] 2

> m[2,] # Extraemos toda la fila 2
[1] 2 4 6 8 10

> m[,2] # Extraemos toda la columna 2
[1] 2 4 6 8 10

> m[-1,] # Extraemos todas las filas, excepto la primera
[1] 4 6 8 10
```

- **Acceso a elementos**: Se puede acceder a un único elemento usando la notación `m[fila, columna]`, o a filas y columnas completas usando `m[fila,]` y `m[,columna]`.
- **Operaciones sobre matrices**: Al igual que con los vectores, se pueden realizar operaciones sobre las matrices, como comparar elementos o extraer subconjuntos con operaciones lógicas.

```
> m[1:2,4:6] # Extraemos un subgrupo de filas y columnas
[1] 1 2 3 4

> m[1,] == m[,1] # Comparar una fila con una columna
[1] TRUE TRUE TRUE TRUE TRUE

> m > 3 # Genera una matriz de valores TRUE/FALSE
[1] FALSE TRUE TRUE TRUE TRUE
```

```
> m[m > 3] # Seleccionamos los elementos que son mayores a 3
[1] 4 6 8 10
```

2.10 El Cajón de Sastre: La Lista

Las listas en R permiten almacenar objetos heterogéneos, es decir, objetos de diferentes tipos de datos. Se pueden asignar nombres a los elementos de la lista para facilitar el acceso.

```
> a1 <- list(boo = v1, foo = v2, zoo = "Hola") # Creamos una lista con diferentes tipos de objeto
> a1["boo"] # Accedemos al objeto con nombre 'boo' dentro de la lista
$boo
[1] 1 2 3 4 5

> a1[["boo"]] # Extraemos el objeto original sin la lista
[1] 1 2 3 4 5

> a1$boo # Equivalente al anterior
[1] 1 2 3 4 5

> a1[[1]] # Accedemos a la primer entrada de la lista
[1] 1 2 3 4 5

> a1$Something <- "A thing" # Añadimos un nuevo elemento a la lista
> a1[["algo"]] <- rep(2, 10) # Añadimos otro nuevo elemento
```

- **Listas en R**: Las listas permiten almacenar diferentes tipos de datos dentro de un mismo objeto, como vectores, matrices, o incluso otras listas.
- **Acceso a elementos**: Se puede acceder a los elementos de una lista por nombre (`a1$boo`), por índice (`a1[[1]]`), o por nombre de clave dentro de la lista (`a1[["boo"]]`).

2.11 Organizando Datos: DataFrames

Un *data frame* es una estructura muy usada en R para almacenar y organizar datos tabulares, como una hoja de cálculo. Cada columna puede contener diferentes tipos de datos.

```
> nitrato <- c(3, 5, 7, 2, 4, 9)
> peso <- c(5, 7, 9, 4, 6, 11)
> longitud <- c(1, 2, 3, 2, 1, 4)
> data <- data.frame('nitrato' = nitrato, 'peso' = peso, 'longitud' = longitud)

> summary(data) # Resumen estadístico de cada columna
  nitrato         peso         longitud
```

```
 Min.   :2.000   Min.   :4.00   Min.   :1.0
 1st Qu.:3.000   1st Qu.:5.50   1st Qu.:1.5
 Median :4.500   Median :6.00   Median :2.0
 Mean   :5.000   Mean   :7.00   Mean   :2.2
 3rd Qu.:7.000   3rd Qu.:8.00   3rd Qu.:3.0
 Max.   :9.000   Max.   :11.00   Max.   :4.0

> cor(data) # Calcula las correlaciones entre las columnas
           nitrato    peso   longitud
nitrato   1.0000000 0.98104 0.90717
peso      0.9810436 1.00000 0.90535
longitud  0.9071734 0.90535 1.00000
```

- **Dataframes**: Permiten organizar datos de manera estructurada y realizar análisis fácilmente.
- **Funciones útiles**: *summary()* ofrece estadísticas descriptivas y *cor()* calcula las correlaciones entre las variables.

2.12 Consultando Datos de los DataFrames

Al igual que las matrices, podemos acceder a los datos de un *data frame* por nombre de columna, o utilizando índices.

```
> colnames(data) # Devuelve los nombres de las columnas
[1] "nitrato"  "peso"     "longitud"

> rownames(data) # Devuelve los nombres de las filas
[1] "1" "2" "3" "4" "5" "6"

> dim(data) # Devuelve las dimensiones del data frame
[1] 6 3

> nrow(data) # Devuelve solo el número de filas
[1] 6

> ncol(data) # Devuelve solo el número de columnas
[1] 3
```

- **Acceso y manipulación de *data frames***: Se pueden cambiar los nombres de las columnas y filas con *colnames()* y *rownames()*, y obtener información sobre las dimensiones con *dim()*, *nrow()*, y *ncol()*.

2.12.1 Accediendo a Datos de los DataFrames

R proporciona múltiples formas de acceder a las columnas, filas, o subconjuntos de un *data frame*.

```
> data$nitrato # Accedemos a la columna nitrato
[1] 3 5 7 2 4 9

> data[1,] # Accedemos a la primera fila
[1] 3 5 1

> data[,1] # Accedemos a la primera columna
[1] 3 5 7 2 4 9

> data[1:2,2:3] # Extraemos las primeras dos filas y las columnas 2 y 3
[1] 5 7 2 3

> data[c(1,3),] # Extraemos las filas 1 y 3
[1] 3 5 1
```

- **Acceso con índices**: Se puede acceder a filas y columnas de la misma manera que con matrices, utilizando índices numéricos o vectores lógicos.

2.13 Graficando Datos

R ofrece una gran variedad de herramientas para visualizar datos. Aquí te mostramos cómo empezar con gráficos simples:

```
> plot(x = data$nitrato, y = data$peso, main = "Nitrato vs Peso", xlab = "Nitrato", ylab = "Peso"
```

- **Gráficos en R**: El comando *plot()* crea un gráfico básico, donde podemos especificar los ejes x e y, el título del gráfico, las etiquetas y el color de los puntos. Esto es útil para explorar la relación entre variables.

GRAFICANDO DATOS

```
> plot(x=data$nitrato,    # Datos eje X
  y=data$longitud,        # Datos eje Y
  pch=19, cex=5,          # Forma y tamaño
  col="dark red")         # Color
> plot(x=data$nitrato,
  y=data$peso,
  pch=19, cex=5,
  col="dark red")
```

GRAFICANDO DATOS CON **ggplot2**

```
> library(ggplot2)        # Cargamos una librería de graficado
> ggplot(data=data,       # Fuente de datos a graficar
  aes(x=nitrato,          # aes asigna variables a ejes, en este caso nitrato a x
  y=peso)) +             # peso a y, el + indica que hay que añadir un elemento gráfico
  geom_line()             # Definimos tipo de gráfica
```

Comando completo:

```
> ggplot(data=data, aes(x=nitrato, y=peso)) +
  geom_line()
```

Las librerías son módulos que añaden funciones a **R** programadas por otras personas. De esta manera, cualquiera puede expandir las capacidades de la plataforma.

Prueba con `nitrato/longitud` y usar `geom_point()` en vez de `geom_line()`.

En estos casos, `>` indica que **R** se queda esperando a que indiquéis un parámetro adicional a la orden anterior. Os aparecerá como un `+`.

GRAFICANDO DATOS CON FACTORES

```
> condicion <- c('A', 'A', 'A', 'B', 'B', 'B')
> data <- data.frame('nitrato' = nitrato,
  'peso' = peso,
  'longitud' = longitud,
  'condición' = condicion)
> ggplot(data, aes(x=condicion, y=nitrato, fill=condicion)) +
  geom_boxplot()
```

Ahora tenemos un nuevo elemento en **aes**, **fill**, que permite dar color a las series. `geom_boxplot` nos genera un nuevo tipo de gráfico.

2.14 FUNCIONES Y LIBRERÍAS

Las librerías son módulos que añaden funciones a **R** programadas por otras personas. De esta manera, cualquiera puede expandir las capacidades de la plataforma. Más adelante usaremos **igraph** para manipular redes.

```
> res <- cor(data)    # Todos los comandos de R son funciones.
  # Se le da la información para trabajar (argumentos de entrada), en este caso data
  # La función devuelve el resultado (guardado en res)
> centralize_vector <- function(vector){   # function permite crear funciones
  new_vector <- vector - mean(vector)     # Entre {} indicamos los pasos
  return(new_vector)                      # return entrega los resultados que considera
}
> centralize_vector(data$peso)            # Aplicamos la función
```

2.15 LEYENDO ARCHIVOS

```
> net_data <- read.table('net', sep="\t")  # Leemos el archivo net
> net_data
```

```
> names(net_data) <- c('Gene', 'Organism')  # Con estas instrucciones, definimos los nombres de l
> net_data
```

Con este código generamos un dataframe a partir del archivo 'net' que construimos al comienzo de esta sesión con el que vamos a hacer una sencilla práctica de redes.

2.16 REDES BIOLÓGICAS

• Son modelos para detectar relaciones entre elementos.
• Se componen de:
- **Nodos**: Entidades que son objeto de estudio.
- **Enlaces**: Conexiones existentes entre las entidades del modelo.

• Si los nodos son:
- **De un solo tipo**: La red se considera monopartita.
- **Más de un tipo**: N-partita, por ejemplo, bipartita, tripartita, etc.
• Si los enlaces son:
- **Binarios**: Representan únicamente la existencia o ausencia de relaciones.
- **Ponderados**: Reflejan la probabilidad o la magnitud de la conexión.

COMO SE DESCRIBEN LAS REDES NORMALMENTE
PAREJAS
- C – B
- C – A
- B – D
- A – D

MATRICES DE ADYACENCIA

	A	B	C	D
A	0	0	1	1
B	0	0	1	1
C	1	1	0	0
D	1	1	0	0

2.16.1 CREANDO REDES EN R

```
> library(igraph) # Cargamos una librería de manejo de redes
> net1 <- graph( edges=c(1,2, 2,3, 3,1), n=10 )
# Definimos las conexiones (edges) como 1-2, 2-3 y 3-1 para una red de 10 nodos
> plot(net1) # Graficamos la red
> net2 <- graph( c("John", "Jim", "Jim", "Jill", "Jill", "John"))
# En este caso, al definir las conexiones, se sobreentiende la creación de los nodos personas.
> plot(net2)
```

2.16.2 CREANDO NUESTRA RED REAL

```
> real_net <- graph_from_data_frame(d=net_data)
# Transformamos el dataframe "net_data" en un contenedor de información especial de la
> plot(real_net)
```

2.16.3 EXAMINANDO LOS COMPONENTES DE LA RED

```
> V(real_net) # Vemos los nodos
> E(real_net) # Vemos las parejas
> real_net[] # Vemos la matriz de adyacencia
```

2.16.4 FORMAS DE VISUALIZAR UNA RED: LAYOUTS

```
> plot(real_net, layout=layout_randomly)
> plot(real_net, layout=layout_on_sphere(real_net))
> plot(real_net, layout=layout_in_circle(real_net))
> plot(real_net, layout=layout_with_kk(real_net))
```

Los **layouts** son formas de representar las redes. Explorando distintos layouts, la red adquiere distintas estructuras que permiten inferir conclusiones biológicas. En nuestro caso, podemos ver que las hexoquinasas son más frecuentes, cuáles son exclusivas de cierto tipo de organismos, o qué organismos presentan más diversidad de hexoquinasas.

Chapter 3

Secuenciación y genómica

Chapter 4

Redes Biológicas y Teoría de Grafos

Chapter 5

Biología Sintética