
Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

# Análisis de coocurrencia de repeticiones maximales en familias de proteínas utilizando reglas de asociación

Tesis de Licenciatura en Ciencias de la Computación

Jonathan Seijo

Director: Pablo Turjanski

Codirector: Diego U. Ferreiro

Jurado: Cecilia Ruz y Alejandro Nadra

Buenos Aires, 2023

# ANÁLISIS DE COOCURRENCIA DE REPETICIONES MAXIMALES EN FAMILIAS DE PROTEÍNAS UTILIZANDO REGLAS DE ASOCIACIÓN

Existen ciertas repeticiones de caracteres en secuencias, las repeticiones maximales, que han sido estudiadas para buscar patrones recurrentes que puedan caracterizar a diferentes familias de proteínas a partir de sus secuencias de aminoácidos. Por otra parte, el análisis de reglas de asociación es una técnica de Data Mining utilizada para encontrar, de forma automática, relaciones interesantes entre elementos de una base de datos. En un trabajo previo se vincularon las reglas de asociación con las repeticiones maximales existentes en las secuencias de proteínas de la familia Ankyrin.

En el presente trabajo, extendemos dicho análisis para poder involucrar a otras dos familias (TPR y LRR). Encontramos diferencias de órdenes de magnitud entre las cantidades de reglas generadas para estas tres familias de proteínas, que se explican por diferencias observadas en las frecuencias de sus repeticiones maximales. Además, resulta que estas familias no comparten reglas de asociación, debido a que sus conjuntos de repeticiones maximales frecuentes casi no tienen elementos en común.

Presentamos algunas optimizaciones temporales y espaciales para el proceso de generación de reglas de asociación (con y sin pérdida de reglas) e investigamos una forma de obtener los mismos conjuntos de reglas a partir de los k-meros de las secuencias pero sin calcular sus repeticiones maximales. Utilizamos una medida, el lift, para analizar la relevancia de las reglas generadas y observamos que resulta insuficiente cuando se aplica al dominio de nuestro problema. Por último, extendemos una herramienta para la visualización de reglas y proteínas (Protein Rule Visualization Tool) para permitir la exploración de datos provenientes de múltiples familias.

**Palabras clave.** Familias de proteínas. Repeticiones maximales. Reglas de asociación. Algoritmo Apriori.

# AGRADECIMIENTOS

A Pablo y Diego. Gracias por aceptar ser mis directores de tesis y todo lo que eso conlleva. Gracias Pablo por tu paciencia, tu entusiasmo y por tu ayuda de cada semana. Fuiste fundamental para poder hacer este trabajo.

A Lucas y Darío. Gracias por las tardes de competencias, las charlas y las sesiones de estudio. Si no hubiera sido así de divertido, habría sido mucho más difícil.

A Carina y Gustavo. Gracias por haberme dado todo e incentivarme siempre a estudiar y a dar lo mejor de mí. Sin ustedes no sería nada de lo que soy ahora.

A Martina. Gracias por la paciencia todas las veces que estuve ocupado con la facu. No podría haberme tocado una hermana mejor ni aunque hubiera nacido infinitas veces.

A Alejandra. Gracias por ser mi amiga y mi compañera incansable de TPs. Sos la razón por la cual puedo terminar esta carrera y estoy muy feliz de que nos hayamos conocido.

A Lucía. Gracias por todo lo que compartimos durante estos años, y por todo lo que nos queda por compartir. Tu apoyo incondicional es de lo más lindo que tengo y no puedo imaginarme lo distinta que sería mi vida sin vos.

# Índice general

# 1. INTRODUCCIÓN

## 1.1. Proteínas, aminoácidos y repeticiones maximales

Las proteínas son biomoléculas que cumplen funciones esenciales en los organismos: actúan como enzimas, mantienen las estructuras celulares, generan movimientos, transducen señales, entre otras funciones. Estas biomoléculas están formadas por cadenas lineales de aminoácidos. Los aminoácidos son moléculas con una estructura central semejante —que les permite unirse a cualquier otro aminoácido—, y con una cadena lateral que les da su carácter químico distintivo [1].

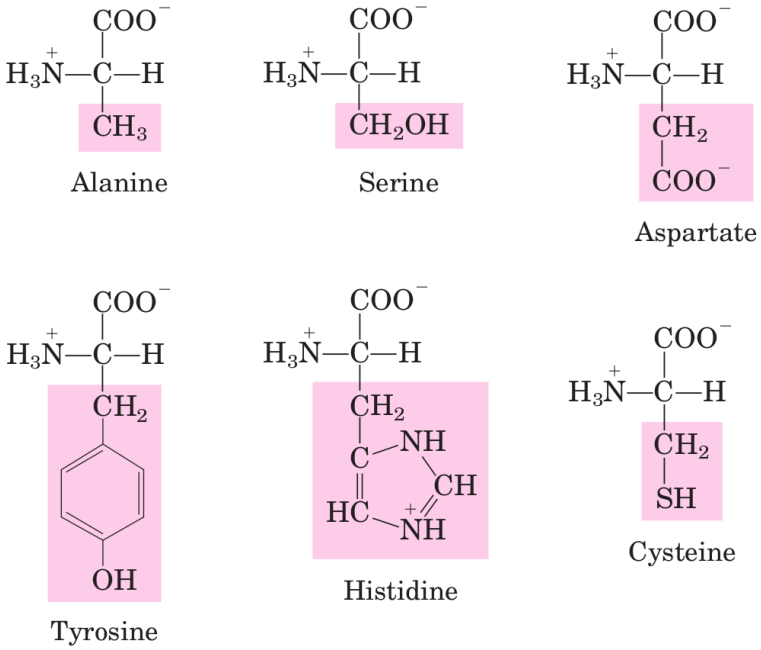



*Fig. 1.1:* Algunos de los aminoácidos presentes en proteínas. Se muestran en rosa las cadenas laterales que los diferencian. Fuente: [2]

En el código genético de los organismos se encuentran codificados alrededor de 20 aminoácidos, los cuales podemos considerar como el alfabeto en el que se escriben las proteínas [2]. Es por esto que podemos representar computacionalmente a cada uno de estos 20 aminoácidos con una letra única y, en consecuencia, pensar a cada proteína (secuencia lineal de aminoácidos) como una cadena de caracteres [3].

Podemos clasificar a las proteínas en grupos, denominados familias, que comparten un origen evolutivo común reflejado en funciones, similitudes en su secuencia o en su estructura [4]. Por ejemplo, las proteínas de la familia *Ankyrin* (Anquirinas) cumplen funciones de estabilidad, estructura y flexibilidad en glóbulos rojos y otras células musculares y cerebrales [5].

Hay trabajos que indican que en una proteína su cadena de aminoácidos es indistinguible de una secuencia aleatoria, pero que, sin embargo, si se sintetizan secuencias aleatorias

éstas no se comportan como proteínas, es decir, no se pliegan en estructuras específicas ni son capaces de funcionar en el contexto celular [6]. Bajo esta observación es natural que surja la siguiente pregunta: ¿existe dentro de una familia de proteínas algún tipo de regla que rija la construcción de sus secuencias proteicas? Para intentar responder esto, Turjanski et al. [7] [8] trabajaron el concepto de repeticiones maximales como bloques constitutivos de las secuencias de proteínas, que usaremos como base en todo este trabajo y definiremos a continuación.

**Definición 1.1.1** (Repetición maximal)**.** Sea $s$ una secuencia finita de caracteres. Una repetición maximal, o Maximal Repeat (MR), es una secuencia que ocurre más de una vez en $s$, y que cada una de sus extensiones (a derecha o izquierda) ocurre menos veces.

Supongamos que tenemos $s =$ `ABCDEABCDFBCDEBCD`. El conjunto de MRs de la secuencia es {`ABCD`, `BCDE`, `BCD`}. `ABCD` y `BCDE` son los MRs más largos, ocurriendo 2 veces. `BCD`, aunque está contenido en los dos anteriores, es también un MR porque ocurre 4 veces en $s$ y cada una de sus extensiones ocurre menos veces. Por el contrario, `BC` no es un MR, porque ocurre 4 veces y su extensión a derecha (`BCD`) ocurre la misma cantidad de veces, no menos.

**Definición 1.1.2** (Categorías de repeticiones maximales)**.** Los MRs pueden ser clasificados en tres categorías disjuntas:

- Super Maximal Repetition (**SMR**): Son MRs donde todas sus extensiones ocurren exactamente una vez (dejando de ser repeticiones).
- Nested Maximal Repetition (**NE**): Todas las ocurrencias de la repetición están contenidas en un MR más grande.
- Non-Nested Maximal Repetition (**NN**): Al menos una de sus ocurrencias no está totalmente contenida dentro de otro MR, y tampoco es un SMR. Es decir, son aquellas que no son NE ni SMR.

Al ser disjuntos, al unir los MRs de las tres categorías obtenemos todos los posibles MRs. En el presente trabajo utilizaremos **ALL** en caso de querer referirnos a las 3 categorías en simultáneo.

En la Figura 1.2 se muestra un ejemplo que permitirá generar una intuición en cuanto a qué es un MR y cuáles son sus posibles clasificaciones. Dada la cadena $s$,

$$s = \texttt{cSMR1dSMR2eMRfSMR2gSMR1h}$$

donde los símbolos en minúscula representan a aquellos caracteres que no se repiten en $s$, su conjunto de MRs es {`MR`, `SMR`, `SMR1`, `SMR2`}.

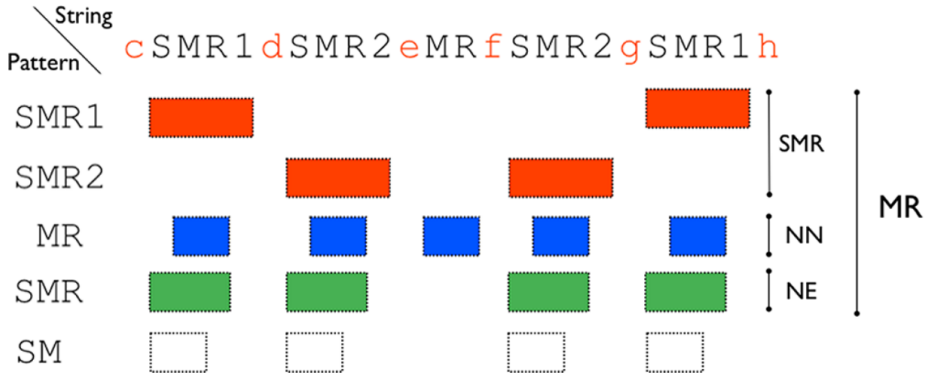



*Fig. 1.2:* MRs para el string que aparece en la parte superior (en rojo y minúsculas, los símbolos que no se repiten). SMR, NN y NE forman conjuntos disjuntos. Fuente: [8]

Llamamos "largo" de un MR a la cantidad de caracteres que contiene. Se puede notar que `SMR1` y `SMR2` son las repeticiones maximales más largas (tienen largo 4, ya que poseen 4 caracteres), ocurriendo dos veces cada una. `SMR` y `MR` son también repeticiones maximales, porque ocurren respectivamente 4 y 5 veces en el string, y todas sus extensiones ocurren menos veces (pero siempre dos o más, por eso no pertenecen a la categoría *Super Maximal Repetition*). `SM` no es una repetición maximal, porque `SMR` (una extensión a derecha) ocurre también 4 veces, rompiendo la definición de que las extensiones debían ocurrir menos veces.

`SMR1` y `SMR2` son *Super Maximal Repetitions* porque sus extensiones (a izquierda o derecha) solo ocurren una vez. `SMR` es una *Nested Maximal Repetition*, porque todas sus ocurrencias están contenidas en `SMR1` y `SMR2`. Finalmente, `MR` se clasifica como *Non-Nested Maximal Repetition* porque, aunque 4 de sus ocurrencias están contenidas en patrones más largos (`SMR1` y `SMR2`), hay una quinta ocurrencia que no está contenida en ninguna repetición mayor.

**Definición 1.1.3** (Repeticiones maximales de un conjunto de secuencias)**.** Sea $S$ un conjunto de $n$ secuencias finitas de caracteres, $S = \{s_1, s_2, ..., s_n\}$. El conjunto de MRs en $S$ es el conjunto de MRs de la secuencia obtenida al concatenar todos los elementos de $S$, intercalados con símbolos distintos $\$_1$, ..., $\$_{n-1}$ que no son parte del alfabeto utilizado. Es decir, los MRs de $S$ son los MRs de la secuencia $s_1\$_1s_2\$_2...\$_{n-1}s_n$.

Como cada $\$_i$ ocurre una única vez y no son parte del alfabeto utilizado, no habrá MR que los contenga. Además, el conjunto de MRs es invariante con respecto al orden en el que concatenemos las secuencias $s_1$, $s_2$, ..., $s_n$. Podremos concatenar cualquier permutación y obtener el mismo conjunto de MRs. Obtener los MRs de esta forma no es equivalente a obtenerlos individualmente en cada $s_i$ y luego agruparlos. Si consideramos $s\$t$, podríamos tener una repetición $r$ que ocurra una vez en $s$ y una vez en $t$. De forma individual, $r$ no es repetición maximal (tiene que aparecer más de una vez) pero sí lo es al estudiar la concatenación, por aparecer 2 veces.

La búsqueda exhaustiva de MRs puede ser implementada con un algoritmo de complejidad computacional $O(n \log n)$, siendo $n$ el tamaño de la secuencia completa de aminoácidos [8] [9].

Utilizar MRs resulta útil para encontrar patrones repetitivos en secuencias. Al pensar a los aminoácidos como caracteres, podemos utilizar esta herramienta para encontrar

patrones en secuencias de proteínas. De esta forma, el analizar diferentes conjuntos de secuencias equivale a analizar familias de proteínas.

## 1.2. Reglas de asociación

El análisis de reglas de asociación es una técnica de Data Mining utilizada para encontrar, de forma automática, relaciones interesantes entre elementos de una base de datos. Esta técnica fue desarrollada para detectar patrones en compras de supermercados, por lo que lleva el nombre de Market Basket Analysis (MBA) en ese contexto. En MBA, a partir de los tickets de compras, se buscan asociaciones entre productos que se llevan juntos de manera frecuente, ya sea para diseñar ofertas o para ubicarlos de forma eficiente en las góndolas.

A continuación, haremos un breve repaso de los diferentes elementos que componen esta técnica y una explicación de su funcionamiento, basándonos en los trabajos de Agrawal et al. [10] y Tan et al. [11].

En la Tabla 1.1 presentamos el conjunto de todos los tickets de un supermercado ficticio, que utilizaremos para ejemplificar las definiciones que detallaremos a continuación. Debemos notar que lo que presentamos como ticket es una abstracción de un ticket real. Un ticket real contiene una lista de productos comprados con sus cantidades, precios individuales, total, vuelto, etc. Los tickets que presentamos contienen únicamente cuáles fueron los productos comprados, que es lo único que interesa en nuestra versión de MBA. El orden de los productos en cada ticket es indistinto.

| ID_TICKET | Productos adquiridos |
|---|---|
| 1 | pan, queso, gaseosa |
| 2 | queso, detergente, pan, arroz |
| 3 | galletitas, gaseosa, queso, pan |
| 4 | arroz, pan, dentrífico |

*Tab. 1.1:* Totalidad de los tickets de un supermercado ficticio.

**Definición 1.2.1** (**Ítem**)**.** Es el elemento básico de nuestro dominio de análisis. En el caso del supermercado un ítem es un producto. En nuestro ejemplo, pan, queso y detergente son algunos de los ítems del supermercado.

**Definición 1.2.2** (**Ítemset**)**.** Es un conjunto de ítems, y es denotado con llaves. Por ejemplo, {queso, detergente} es un ítemset. Notar que al ser un conjunto, no tiene elementos repetidos.

**Definición 1.2.3** (**Transacción**)**.** Es un ítemset que se encuentra en la base de datos a utilizar para la generación de reglas.

Por ejemplo, el ítemset {pan, queso, gaseosa} es una transacción, porque es el ítemset que puede observarse en el primer ticket de la Tabla 1.1. Otro ejemplo, {pan, queso} es un ítemset pero **no** es una transacción, porque no existe ningún ticket que se corresponda con exactamente esos productos.

Cuando hablamos de tickets, estamos hablando del conjunto de productos de un supermercado, en el dominio de MBA. Cuando hablamos de transacciones, hablamos de un conjunto de ítems en el dominio de reglas de asociación. Aunque cada uno de los tickets

de la Tabla 1.1 se corresponde con una transacción, utilizamos las palabras "ticket" y "transacción" de acuerdo al dominio sobre el que estemos hablando.

**Definición 1.2.4** (**Regla**)**.** Es una relación entre dos ítemsets A y B, denotada con una flecha (A → B), que representa una coocurrencia entre el ítemset antecedente y el consecuente. Para ser considerada una regla válida los ítemsets A y B no deben compartir ítems, y además en este trabajo agregaremos la restricción de que el consecuente debe tener un único ítem.

Por ejemplo, una regla podría ser la siguiente: {pan, queso} → {gaseosa}. Esto representa que cada vez que en un ticket aparece que un cliente adquirió pan y queso, es altamente probable que también haya adquirido gaseosa (y no necesariamente al revés).

Se debe tener en cuenta que una regla no tiene por qué tener un resultado esperado. Por ejemplo, también podríamos encontrarnos con la siguiente regla: {detergente} → {arroz}.

En ningún caso una regla representa una implicación: son relaciones de coocurrencia. En una transacción en particular podría potencialmente aparecer el antecedente sin que necesariamente aparezca el consecuente.

Debido a que casi cualquier relación entre ítemsets podría ser una potencial regla, en esta técnica se utilizan dos medidas para poder jerarquizar las reglas: soporte (*support*) y confianza (*confidence*).

**Definición 1.2.5** (Conteo)**.** El conteo (o *support count*) de un ítemset $S$, es la cantidad de transacciones en las cuales todos los elementos del ítemset aparecen juntos. Lo notamos como $\sigma(S)$. En nuestro ejemplo:

$$\sigma(\{\text{pan, queso}\}) = 3$$

porque {pan, queso} aparece en los tickets 1, 2 y 3. En el ticket 4, aparece "pan" pero no "queso", por lo que no suma en el conteo.

**Definición 1.2.6** (Soporte de un ítemset)**.** Sea $T$ la cantidad total de transacciones. El soporte o *support* de un ítemset $S$ es:

$$soporte(S) = \frac{\sigma(S)}{T} \tag{1.1}$$

En nuestro ejemplo,

$$soporte(\{\text{pan, queso}\}) = \frac{\sigma(\{\text{pan, queso}\})}{4} = \frac{3}{4} = 0{,}75$$

En ocasiones el soporte se expresa en porcentaje. Podemos decir que el soporte del ejemplo es del 75 %.

**Definición 1.2.7** (Soporte de una regla)**.** Sea $T$ la cantidad de transacciones. Sean A y B ítemsets. El soporte de una regla $A \to B$ es:

$$soporte(A \to B) = \frac{\sigma(A \cup B)}{T} \tag{1.2}$$

En nuestro ejemplo,

$$soporte(\{\text{pan, queso}\} \to \{\text{gaseosa}\}) = \frac{\sigma(\{\text{pan, queso, gaseosa}\})}{4} = \frac{2}{4} = 0{,}5$$

El soporte representa el porcentaje de apariciones y nos es útil porque en general nos interesará considerar reglas que involucren a ítems con alta frecuencias de aparición en nuestras transacciones.

**Definición 1.2.8** (Confianza)**.** Solo se aplica a las reglas. Si A y B son ítemsets, la confianza, o *confidence*, de una regla $A \rightarrow B$ es:

$$confianza(A \rightarrow B) = \frac{soporte(A \cup B)}{soporte(A)} \tag{1.3}$$

En nuestro ejemplo,

$$confianza(\{\text{pan, queso}\} \rightarrow \{\text{gaseosa}\}) = \frac{soporte(\{\text{pan, queso, gaseosa}\})}{soporte(\{\text{pan, queso}\})} = \frac{0{,}5}{0{,}75} = 0{,}66$$

La confianza representa a la probabilidad condicional $P(B|A)$. Nos indica cuál es la probabilidad de que ocurra el consecuente dado que ocurre el antecedente, y representa qué tan "confiable" es la regla. Con un mismo ítemset pueden formarse varias reglas distintas que, aunque su soporte sea el mismo para todas (por provenir del mismo ítemset), pueden diferenciarse por su confianza dado que este último valor depende de quién participa como antecedente en la regla.

Utilizando estas dos medidas, *support* y *confidence*, los algoritmos de generación de reglas suelen descomponer el problema en dos subtareas, ejecutadas en el siguiente orden:

1. Generación de ítems frecuentes: el usuario fija un valor de soporte mínimo, llamado `min_support`, y se seleccionan todos los ítemsets que tengan un soporte mayor a dicho umbral.

2. Generación de reglas: el usuario fija un valor de confianza mínima, `min_confidence`, y, usando los ítemsets obtenidos en la subtarea 1, se intentan generar reglas que superen dicho valor. Una forma de generar reglas podría ser, por ejemplo, probar todas las combinaciones posibles de antedecendes y consecuentes usando los elementos del ítemset.

La forma de implementar las subtareas descriptas es parte de lo que diferencia a los distintos algoritmos generadores de reglas. Para este trabajo utilizaremos el algoritmo *Apriori* [11] (también usado en el trabajo de Enríquez [12]) y lo definiremos a continuación.

**Definición 1.2.9** (Ítemset frecuente)**.** Es un conjunto de ítems que tiene un soporte mayor a `min_support`. Si el ítemset está compuesto por un único elemento, también hablamos de *ítem frecuente*.

**Definición 1.2.10** (Principio Apriori)**.** Si un conjunto de ítems es frecuente, entonces todos sus subconjuntos también son frecuentes. Por ejemplo, si {pan, queso, gaseosa} es un ítemset frecuente, entonces son también frecuentes: {pan, queso}, {queso, gaseosa}, {pan}, etc.

**Definición 1.2.11** (Corolario del Principio Apriori)**.** Del Principio Apriori puede deducirse que si un ítemset **no** es frecuente, entonces todos los superconjuntos que lo contengan tampoco serán frecuentes. Esto significa que en la búsqueda de ítemsets frecuentes, los ítemsets no-frecuentes pueden ser descartados para reducir el espacio de búsqueda.

**Definición 1.2.12** (Algoritmo Apriori)**.** Es el algoritmo que utilizamos como generador de reglas en este trabajo. El algoritmo *Apriori* comienza con ítemsets frecuentes de un único elemento (llamados también ítems frecuentes) y en cada iteración, teniendo en cuenta sus soportes, realiza la unión de algunos de estos conjuntos de forma eficiente. En estas uniones, se utiliza el *Principio Apriori* y su corolario para obtener ítemsets frecuentes de mayor tamaño cada vez. Una vez obtenidos los ítemsets frecuentes para todos los tamaños, se generan todas las posibles reglas que usan esos ítems y finalmente se las filtra por su confianza.

Una descripción más detallada del funcionamiento interno del algoritmo, incluyendo pseudocódigo y definiciones formales, puede encontrarse en el trabajo de Enríquez [12]. En este trabajo nos limitaremos a usar una implementación preexistente en el paquete *arules* del lenguaje R [13].

## 1.3. Generación de reglas usando repeticiones maximales

Si asumimos que los MRs de una proteína son sus bloques constitutivos, podemos pensar a cada MR como un producto de supermercado o, equivalentemente, como un ítem en el análisis de reglas. Así como un ticket de supermercado está compuesto por distintos productos, y una transacción está compuesta por distintos ítems, la secuencia de aminoácidos que compone a la proteína está compuesta por distintos MRs. De esta manera, podemos pensar en una equivalencia entre los tres dominios (MBA, reglas de asociación y el biológico) que presentamos en la Tabla 1.2.

| **Reglas de asociación** | **MBA** | **Dominio biológico** |
|---|---|---|
| ítem | producto | MR |
| transacción | ticket | proteína |
| base de transacciones | tickets del supermercado | familia de proteínas |

*Tab. 1.2:* Equivalencias entre reglas de asociación, MBA y el dominio biológico.

Visto de esta forma, aplicar el algoritmo *Apriori* para hallar reglas de asociación entre ítems es equivalente a hallar reglas entre los MRs de una familia de proteínas. Nuestra hipótesis de trabajo es que obtener estas reglas nos permitiría entender algunas de las características de la familia y observar si efectivamente existen patrones propios que las diferencien de las secuencias aleatorias.

## 1.4. Trabajo previo

En la tesis de licenciatura de Enríquez [12] (de ahora en más Enríquez) se comenzó a estudiar el problema utilizando la familia de proteínas *Ankyrin*. Para esto, se desarrolló un sistema que genera reglas de asociación usando el algoritmo *Apriori*, partiendo del cálculo de repeticiones maximales para la generación de transacciones.

Uno de los primeros resultados observados en el trabajo de Enríquez fue la aparición de reglas que se cumplen trivialmente, es decir, reglas cuyo consecuente es un substring del antecedente. Por ejemplo, podemos ver en la Figura 1.3 algunas de las reglas triviales generadas en la primer iteración.

$$\{\underline{\textbf{TPLHL}}\text{A}\} \rightarrow \{\text{TPLHL}\}$$
$$\{\text{T}\underline{\textbf{PLHLA}}\} \rightarrow \{\text{PLHLA}\}$$
$$\{\text{T}\underline{\textbf{PLHL}}\text{A}\} \rightarrow \{\text{PLHL}\}$$
$$\{\text{TP}\underline{\textbf{LHLA}}\} \rightarrow \{\text{LHLA}\}$$
$$\{\underline{\textbf{TPLH}}\text{LA}\} \rightarrow \{\text{TPLH}\}$$

*Fig. 1.3:* Algunas de las reglas triviales generadas en la primera iteración del trabajo de Enríquez. En el antecedente de cada regla se subraya el substring que corresponde al consecuente.

Este tipo de reglas donde la asociación se produce trivialmente por ser substrings no está presente en el dominio de MBA, ya que este tipo de inclusión no se encuentra en productos de un supermercado. Para evitar que su generación, se decidió agregar una etapa de **refinamiento** a las transacciones, donde se eliminan de ellas a todos los MRs que, dependiendo el modo elegido, sean substrings o superstrings de algún otro dentro de la misma transacción. Por ejemplo, si utilizamos el modo de refinamiento *substring* con la siguiente transacción:

{TALHYA, TALHY, TALH, LISHGA}

se eliminarían los MRs que contienen a otros como substrings, quedando la transacción de la siguiente forma:

{TALH, LISHGA}

Es posible configurar este refinamiento de dos modos diferentes. El primero es el modo *substring*, que se queda con los MRs más pequeños al eliminar aquellos que contienen a otro MR como substring. El segundo es el modo *superstring*, que se queda con los MRs grandes al eliminar aquellos que son contenidos por otro MR, su superstring. En el trabajo de Enríquez se hicieron pruebas y se decidió utilizar únicamente el modo *substring*, dado que es el único modo que genera reglas con todos los valores de `min_confidence` y `min_support` explorados. Continuaremos utilizando esta forma de generar transacciones y, salvo que aclaremos lo contrario, cuando hablemos de transacciones nos estaremos refiriendo a estas post-refinamiento.

Existe otro parámetro utilizado para la generación de transacciones: `min_len`. Este parámetro define el largo mínimo de los MRs que van a incluirse en las transacciones, antes de la etapa de refinamiento. Su valor por defecto es 4.

El proceso de generación de transacciones y reglas que utilizaremos en este trabajo estará también basado en el software de Enríquez. En el Apéndice (Capítulo 9) puede consultarse un detalle de los scripts que utilizaremos, junto con una descripción de sus parámetros. Se mantienen las tres etapas del proceso:

1. Obtención de MRs a partir de una familia de proteínas.
2. Construcción y refinamiento de transacciones usando los MRs.
3. Generación de reglas de asociación.

Junto a la implementación del software para generar transacciones y reglas de asociación, en el trabajo de Enríquez se analizaron las reglas producidas utilizando la familia

*Ankyrin*. Se intentó clasificarlas según la información que aportan y se calcularon medidas de interés como cobertura de la regla en la proteína, frecuencias y distancias de edición entre los componentes de las reglas. Además, se desarrolló un visualizador web (Protein Rule Visualization Tool) que permite explorar las reglas generadas y algunas de las medidas antes mencionadas.

## 1.5. Contenido del trabajo

En este trabajo extenderemos el análisis de coocurrencias de repeticiones maximales usando reglas de asociación, iniciado en la tesis de Enríquez con la familia *Ankyrin*, a múltiples familias de proteínas. Esperamos de esta forma profundizar nuestro entendimiento de las repeticiones maximales y entender cómo se determinan y cuáles son las reglas de asociación que se generan a partir de diferentes familias.

Primero, estableceremos un criterio que nos permitirá elegir un grupo de familias de proteínas que sean comparables entre sí, y compararemos las reglas de asociación que se generan con cada una de ellas.

A continuación, intentaremos entender en profundidad por qué se producen diferencias entre las reglas de cada familia, y para esto realizaremos un análisis de las repeticiones maximales que se encuentran presentes en las transacciones, junto con sus frecuencias.

Luego, buscaremos optimizaciones para mejorar, tanto en tiempo como en memoria, el proceso de generación de reglas. Para esto diseñaremos un modo distinto de refinamiento, estudiaremos la utilización de subconjuntos de ítems y propondremos una alternativa al uso de repeticiones maximales.

Para finalizar, presentaremos un posible criterio para determinar la relevancia de las reglas generadas y modificaremos el visualizador de reglas de asociación y conjuntos de proteínas (desarrollado por Enríquez en su tesis) para permitir trabajar con múltiples familias.

# 2. FUENTES DE DATOS

## 2.1. Familia de proteínas Ankyrin

Los trabajos de Turjanski et al. 2016 [7] (de ahora en más, Turjanski2016) y de Enríquez utilizaron un conjunto de secuencias al que denominaron `Ank`. Estas secuencias pertenecen a la familia *Ankyrin* [14] y fueron recolectadas en el año 2016 por el grupo de Ferreiro [15]. En el año 2018, para el trabajo Turjanski et al. 2018 [8] (de ahora en más Turjanski2018) se actualizó dicho conjunto y fue denominado `NEWAnk`.

En la Tabla 2.1 se pueden ver las principales diferencias en cuanto a cantidad de proteínas y sus largos (cantidad de aminoácidos que la componen) entre estos dos conjuntos de secuencias. Además, calculamos su intersección para entender cuáles secuencias se encuentran en común.

Resulta que `NEWAnk` posee 5.882 secuencias menos (que representan un 15 % de `Ank`), aunque en promedio éstas son más largas, con alrededor de 100 aminoácidos extra. Al analizar la intersección, podemos ver que cerca de un cuarto de las secuencias presentes en `Ank` se encuentran también en `NEWAnk`.

| | `Ank` | `NEWAnk` | `Ank` ∩ `NEWAnk` |
|---|---|---|---|
| Cantidad de proteínas | 38.051 | 32.169 | 9.463 |
| Largo de proteínas (promedio) | 631 | 717 | 673 |
| Largo de proteínas (mediana) | 493 | 576 | 545 |

*Tab. 2.1:* Diferencias en cantidades y largos de proteínas entre el conjunto `Ank` de Enríquez, `NEWAnk` de Turjanski2018 y su intersección.

A partir de la actualización de las secuencias y de las diferencias vistas, surge la pregunta de si se generan las mismas reglas al seguir la metodología usada en el trabajo de Enríquez, pero cambiando el conjunto de datos de entrada `Ank` por `NEWAnk`. Como queremos partir de los resultados más recientes, nuestro objetivo es poder utilizar `NEWAnk` en este trabajo.

### Metodología

Para responder la pregunta planteada reemplazamos el conjunto de datos `Ank` por `NEWAnk`, y usando el software desarrollado por Enríquez, generamos las reglas de asociación manteniendo los mismos parámetros por defecto que utilizó en su trabajo. Los valores de los parámetros más relevantes se mencionan en la Tabla 2.2. Para más detalle, la lista completa de parámetros se encuentra presente en el apéndice (Sección 9.3).

| Parámetro | Valor por defecto | Descripción |
|---|---|---|
| `min_support` | 0.025 | Soporte mínimo necesario para considerar una regla |
| `min_confidence` | 0.90 | Confianza mínima necesaria para considerar una regla |
| `min_len` | 4 | Longitud mínima de los MRs para ser considerados ítems en las transacciones |

*Tab. 2.2:* Parámetros principales y sus valores por defecto, utilizados para obtener las reglas de asociación mediante el software desarrollado por Enríquez.

Para verificar que el software usado funciona correctamente, también generamos las reglas de asociación utilizando el conjunto de datos original (`Ank`), esperando obtener las mismas reglas que en el trabajo de Enríquez.

### Resultados

Al utilizar el conjunto de datos `Ank` se obtienen 394 reglas, las mismas que en el trabajo de Enríquez. Esto nos permite confiar que el software utilizado y el conjunto de datos coinciden con los de su trabajo.

Al utilizar el conjunto de datos `NEWAnk` se obtienen una cantidad mayor de reglas: 4.888. De estas, 369 (todas menos 25 reglas) son exactamente las mismas a las obtenidas en la corrida donde fue utilizado el conjunto de datos `Ank`. Es decir, el 94 % de reglas de asociación generadas con `Ank` son también generadas con `NEWAnk`.

### Discusión

Al comparar las reglas obtenidas utilizando el nuevo conjunto de datos (`NEWAnk`) y el viejo conjunto (`Ank`), observamos que la cantidad de reglas obtenidas con el primero es un orden de magnitud mayor que las obtenidas con el segundo. También observamos que la intersección es grande (como mencionamos previamente, el 94 % de las reglas obtenidas con `Ank` también fueron obtenidas con `NEWAnk`). Esto es esperable, porque aunque el conjunto de datos no sea exactamente el mismo, éste intenta representar a la misma familia de proteínas.

A modo de prueba exploratoria duplicamos el valor de `min_support` (pasamos de 0.025 a 0.050), es decir, hicimos que el umbral sea más restrictivo, con el objetivo de obtener una cantidad menor de reglas y quedarnos con aquellas de mayor frecuencia. Como resultado observamos que las cantidad de reglas generadas con `NEWAnk` disminuyen de 4.888 a 545 pero, sin embargo el 85 % de las reglas generadas con `Ank` se siguen manteniendo. A partir de esto, asumimos que los resultados obtenidos con el conjunto de datos `NEWAnk` son consistentes con los obtenidos con `Ank` y, por lo tanto, los resultados de nuestro trabajo serán comparables con los obtenidos en Enríquez.

De aquí en adelante, mantendremos los valores por defecto de los parámetros de Enríquez (Tabla 2.2) para poder hacer algunas comparaciones con sus resultados.

## 2.2. Familias comparables

Dado que uno de los objetivos de este trabajo es comparar reglas de asociación generadas por diferentes familias de proteínas, es importante que no existan diferencias sustanciales entre los conjuntos de secuencias a comparar. A modo de ejemplo, si una familia estuviese compuesta por una única secuencia proteica, es de esperar que las reglas de asociación generadas al utilizar dicha familia no sean comparables con las de `NEWAnk`, que cuenta con más de 32.000 secuencias. Es por ello que para nuestro trabajo intentaremos utilizar familias que tengan una cantidad similar de secuencias, y que éstas tengan a su vez similares largos promedios.

### Metodología

Los dominios son unidades funcionales o estructurales dentro de una proteína, y son responsables de funciones o interacciones particulares dentro del rol general de la proteína. Los dominios y las familias están relacionados, ya que pueden utilizarse un dominio para decidir a qué familia pertenece una proteína. Muchas de las proteínas con las cuales trabajamos son multidominio, por lo que la clasificación de una secuencia a una familia específica no es directa. El dataset de familias que utilizamos en este trabajo fue obtenido con la metodología descrita en Turjanski2016, que utiliza las etiquetas dadas por Pfam a las secuencias.

Biológicamente, las familias pueden ser clasificadas en dos tipos: las repetitivas (*repeat proteins*) y las globulares (*globular proteins*). En Turjanski2018 se utilizan 26 familias de tipo repetitivas y 20 de tipo globulares. Utilizamos dichas familias como fuente de datos para el presente estudio y, para seguir en línea con la clasificación mencionada previamente, analizamos la cantidad de secuencias y sus largos en grupos separados de acuerdo a su tipo.

### Resultados

En las Figuras 2.1 y 2.2 se pueden observar la cantidad de secuencias proteicas que contiene cada familia de nuestro conjunto de datos, para familias de proteínas repetitivas y globulares, respectivamente. Las familias se encuentran ordenadas en orden creciente en base a la cantidad de proteínas que contiene cada conjunto, para permitir observar las diferencias en estas cardinalidades.

Dentro de las familias de proteínas repetitivas, la que menos secuencias posee es `Nebulin`, con solo 1.062 secuencias; y entre las que más secuencias posee se encuentra `PPR`, con 38.342 secuencias. Entre las familias con mayor cantidad de secuencias se observan varias familias con cantidades similares, como por ejemplo `NEWAnk` y `LRR1` (con 32.169 y 33.060 secuencias respectivamente).

Dentro de las familias de proteínas globulares, la que menos secuencias posee es `PUD`, con 924 secuencias; y la que más posee es `HelicaseC` con 37.928. Se encuentran también, entre las familias que más secuencias poseen, `ABCtran` y `Pkinase`, con 27.442 y 27.797 secuencias, respectivamente. Éstas tienen cantidades muy similares entre sí, pero con una diferencia de casi 10.000 con respecto a la de máxima cantidad.

Tanto las cantidades mínimas como las máximas son similares entre las familias repetitivas y globulares. Sin embargo, la distribución no es igual. Por ejemplo, en las globulares hay una única familia que supera las 30.000 proteínas (1 entre 20 familias, representando el 5 %) mientras que en las repetitivas hay 8 familias que superan dicho valor (8 entre 26

familias, representando el 30 %).

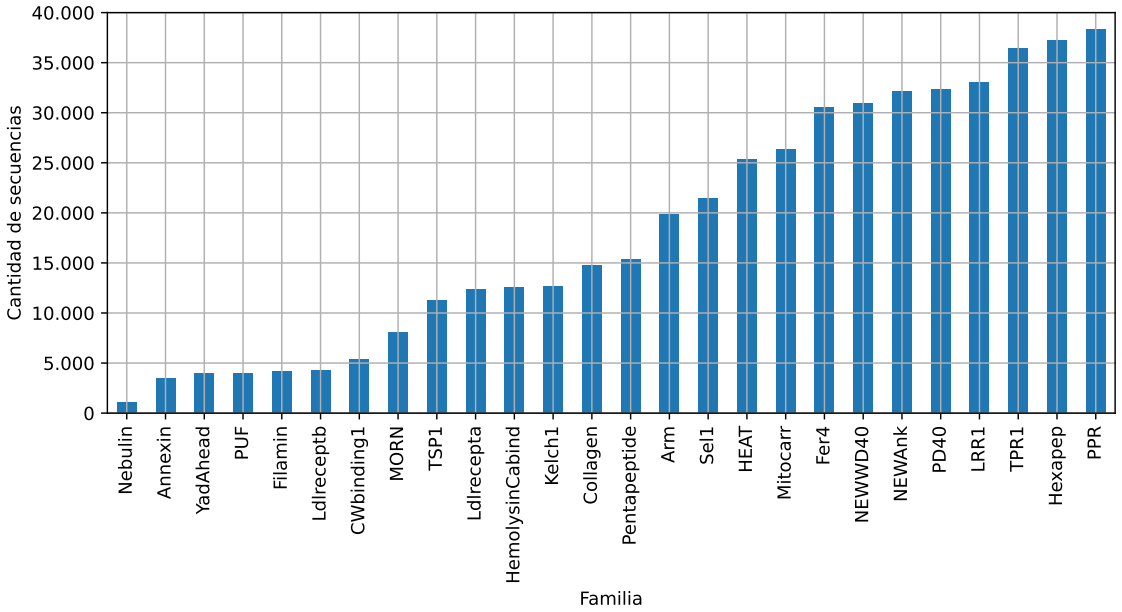



*Fig. 2.1:* Cantidad de secuencias en cada familia de proteínas, para familias de proteínas **repetitivas**. Las familias han sido ordenadas en orden creciente, en base a la cantidad de secuencias que contiene cada familia.

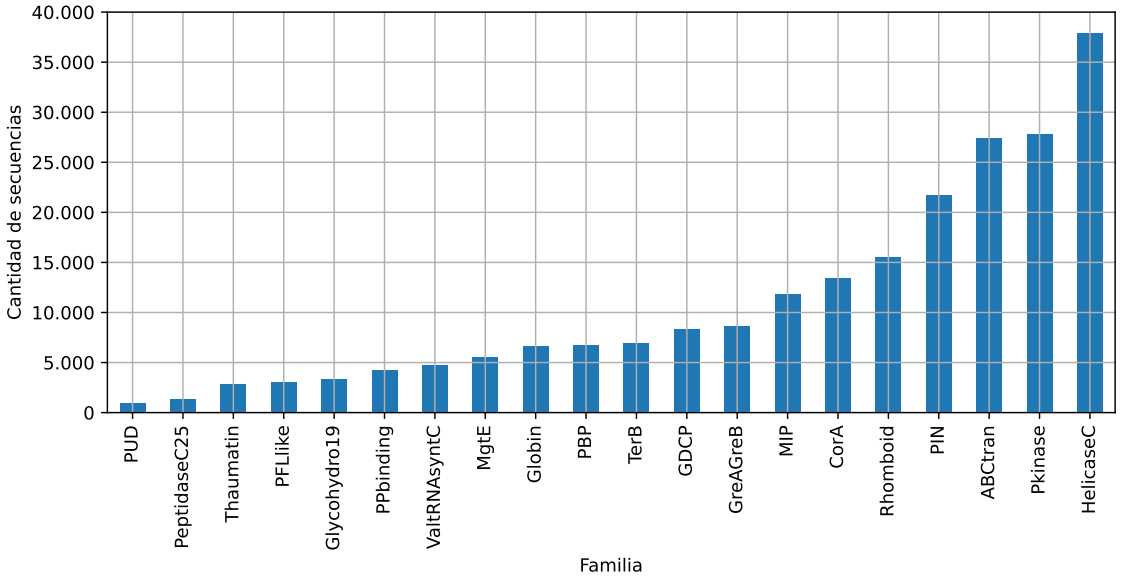



*Fig. 2.2:* Cantidad de secuencias en cada familia de proteínas, para familias de proteínas **globulares**. Las familias han sido ordenadas en orden creciente, en base a la cantidad de secuencias que contiene cada familia.

En las Figuras 2.3 y 2.4 se presentan gráficos de tipo boxplot con los largos de las proteínas de tipo repetitivas y globulares, respectivamente. Las familias se encuentran

ordenadas con respecto a la mediana de su largo.

Dentro de las familias de proteínas repetitivas, la familia con menor mediana en sus largos de proteínas es `Hexapep`, con 217 aminoácidos; y la de mayor mediana es `Ldlreceptb` con 894 aminoácidos. Al seguir el orden de la figura, desde `NEWAnk` hasta `TPR1` inclusive, se encuentra un grupo con medianas muy similares, entre 576 y 585 aminoácidos respectivamente. Esta similitud, con un leve incremento, se mantiene también para el largo de `LRR1`, con una mediana de 669 aminoácidos.

En las familias de proteínas globulares, la familia de menor mediana es `PIN`, con 136 aminoácidos. Podemos apreciar que a diferencia de todas las familias repetitivas, la mayoría de los largos de las secuencias de la familia `PIN` se encuentran muy cercanos a su mediana (hay una menor dispersión). Con la familia `GreAGreB` se produce un hecho similar: hay una dispersión muy baja alrededor de una mediana de 158 aminoácidos. Un grupo de familias parecidas que podemos destacar son aquellas que en la figura van desde `Thaumatin` hasta `Glycohydro19`, con longitudes medianas entre 249 y 268 respectivamente. `PPbinding` tiene la mayor mediana de todas las globulares, con 5.069 aminoácidos, y tiene también mayor largo que todas las proteínas repetitivas analizadas.

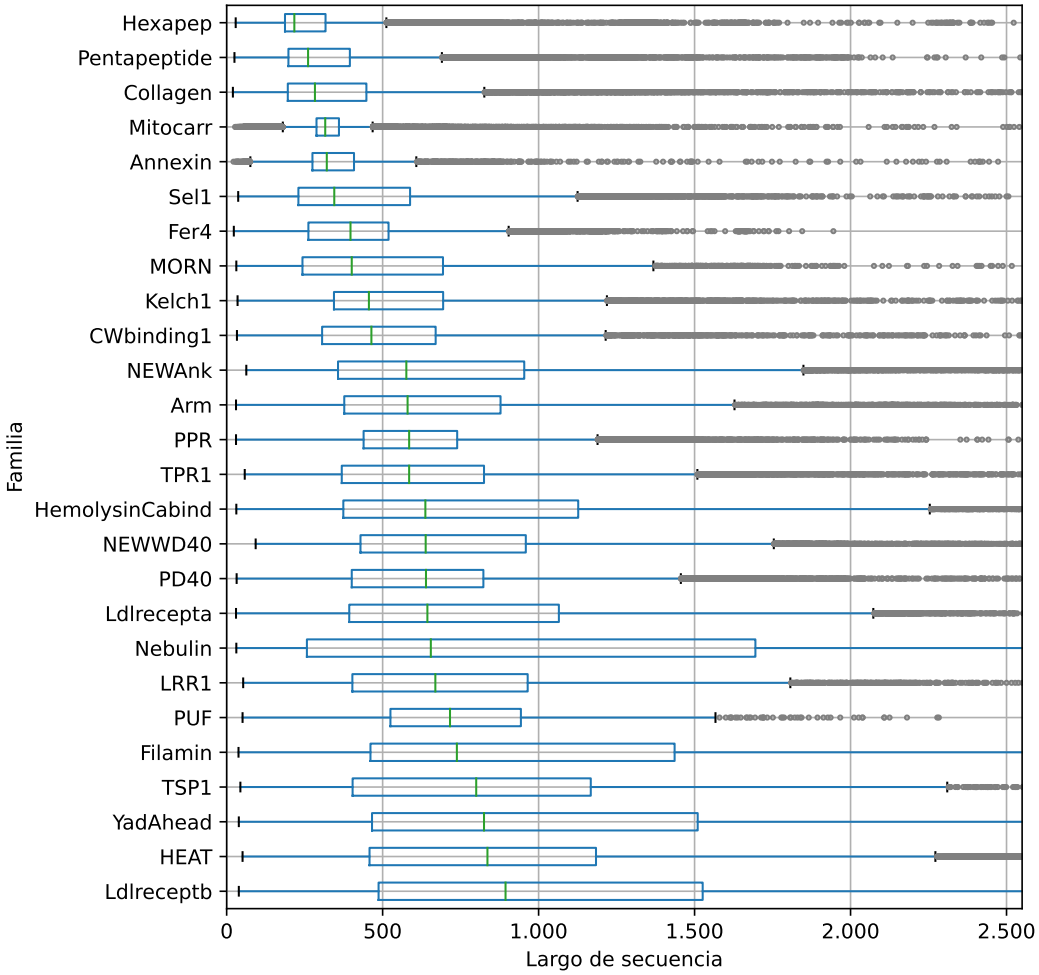



*Fig. 2.3:* Largos de las secuencia en las familias de proteínas de tipo **repetitivas**. Las familias se ordenan en orden creciente con respecto a su mediana. Recortamos el eje X en 2.500 para evitar que los outliers distorsionen el gráfico.

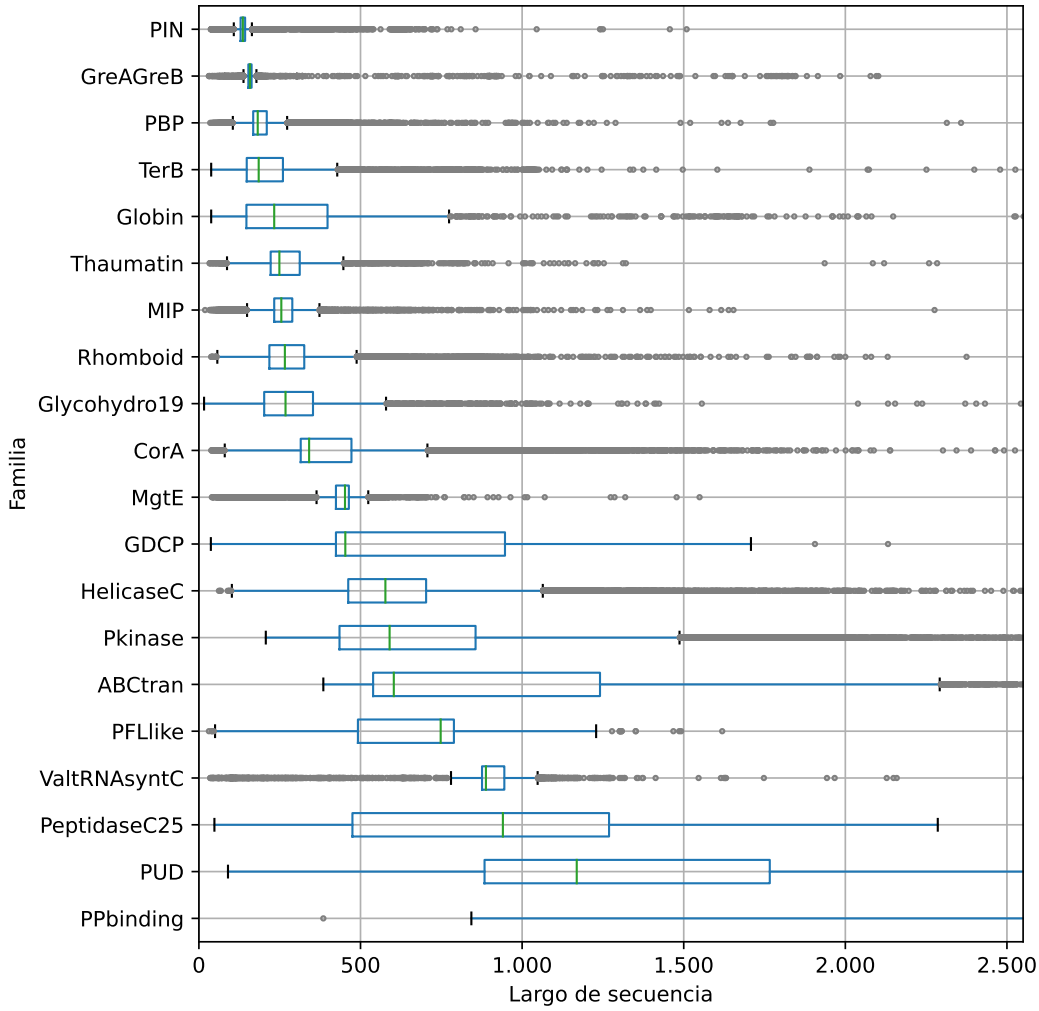



*Fig. 2.4:* Largos de las secuencias en las familias de proteínas de tipo **globulares**. Las familias se ordenan en orden creciente con respecto a su mediana. Recortamos el eje X en 2.500 para evitar que los outliers distorsionen el gráfico.

### Discusión

Según las equivalencias vistas (Tabla 1.2), dada una familia, su cantidad de secuencias equivale a la cantidad de transacciones. Con la misma idea, aunque el largo de las secuencias no será exactamente igual al tamaño de las transacciones (pues el tamaño de la transacción depende de la cantidad de MRs), consideramos que estos dos valores serán proporcionales. Es decir, con secuencias largas esperamos obtener transacciones de gran tamaño. Si bien el largo promedio de las transacciones no es determinante en la generación de las reglas, suponemos que si hay una diferencia muy grande en los tamaños, estaríamos interfiriendo en las propiedades intrínsecas de las familias. Por esas razones, consideramos que si tomamos familias que sean similares en cantidad y largo de sus secuencias, la comparación que se haga entre ellas será en principio más justa, y las diferencias obtenidas corresponderán a cualidades propias de las familias.

Podemos elegir diferentes grupos de familias similares para utilizar. Dentro de las repetitivas podríamos tomar, por ejemplo, `PUF`, `Filamin` y `Ldlreceptb`, pero tienen muy pocas

secuencias. Para que las reglas sean representativas, queremos partir de una gran cantidad de transacciones, por lo que consideraremos familias con más de 30.000 secuencias. Entran en esta clasificación `Fer4`, `NEWAnk`, `PD40`, `LRR1`, `TPR1`, `Hexapep` y `PPR`. Dentro de las globulares solo hay una familia que supera las 30.000 secuencias, así que disminuiremos el valor de nuestro umbral para poder elegir al menos tres familias. Las tres familias con mayor cantidad de secuencias que superan las 25.000 secuencias son `ABCtran`, `Pkinase` y `HelicaseC`.

Al tener en cuenta el largo de las secuencias, los dos grupos posibles mencionados tienen longitudes medianas similares, entre aprox. 500 y 1.000 aminoácidos. Un detalle a observar es que algunas familias que se veían parecidas en cantidades de secuencias no son tan parecidas al analizar el largo medio de ellas. Por ejemplo, en el caso de las repetitivas, `TPR1` se encuentra mucho más cerca a `NEWAnk` que `LRR1` si consideramos largos promedios que si consideramos cantidades.

Entre los grupos de repetitivas y globulares, optamos por quedarnos con las repetitivas. Poder usar a `NEWAnk` es muy importante para nosotros, no solo por tener presentes los resultados de Enríquez, sino porque los directores de este trabajo tienen experiencia previa con esta familia. Además, decidimos quedarnos con un total de tres familias para simplificar el análisis al reducir la cantidad de combinaciones en la experimentación.

Si bien podríamos haber tomado otras, decidimos quedarnos con `NEWAnk`, `LRR1` y `TPR1` (*Ankyrin*, *Leucine Rich Repeat* y *Tetratricopeptide Repeat*, respectivamente). Pueden verse en la Tabla 2.3 los valores correspondientes a las características consideradas. Utilizaremos estas tres familias como fuente de datos primaria para el resto del trabajo. Las denominamos *familias comparables*.

| **Familia** | **Cantidad de secuencias** | **Longitud mediana de secuencias** |
|---|---|---|
| `NEWAnk` | 32.169 | 576 |
| `TPR1` | 36.389 | 585 |
| `LRR1` | 33.060 | 669 |

*Tab. 2.3:* Propiedades de las familias comparables.

Contaremos también con los siguientes grupos de control basados en `NEWAnk`:

- `NEWAnk_UNIFORM`: Cada proteína de `NEWAnk` se reemplaza con la misma cantidad de aminoácidos elegidos al azar con distribución uniforme.
- `NEWAnk_SCRAMBLED`: Cada proteína de `NEWAnk` se reemplaza con una permutación de sus aminoácidos al azar.

Ambos grupos, por construcción, tienen exactamente la misma cantidad de secuencias que `NEWAnk` y el largo de sus secuencias son también iguales al haber únicamente cambiado los valores de los aminoácidos (no la cantidad).

Queda como trabajo futuro extender el conjunto de familias comparables y también analizar qué pasa con familias que no son de tipo repetitivas.

# 3. GENERACIÓN DE REGLAS DE ASOCIACIÓN

## 3.1. Reglas iniciales en familias comparables

En el capítulo anterior tomamos el conjunto de familias de proteínas usadas en Turjanski2018 y observamos que no todas poseen características similares en cuanto a cantidad o largo de las secuencias que las integran. Para evitar que diferencias en dichas características afecten nuestros resultados, decidimos trabajar con un grupo de familias comparables. Así, seleccionamos las familias `NEWAnk`, `TPR1` y `LRR1`.

También vimos que para la familia `NEWAnk`, si utilizamos el software y parámetros del trabajo de Enríquez, se obtienen 4.888 reglas de asociación. Si mantenemos las mismas condiciones experimentales que se usaron al generar reglas con `NEWAnk`, ¿cuántas reglas obtendremos para `TPR1` y `LRR1`?

### Metodología

Utilizamos el software de Enríquez y los parámetros por defecto (mencionados en la Tabla 2.2), para generar las reglas de asociación con las familias `TPR1` y `LRR1`. Adicionalmente, repetimos el mismo proceso para las familias de control `NEWAnk_SCRAMBLED` y `NEWAnk_UNIFORM`. Debido al tamaño de `LRR1`, asignamos un valor de *timeout* mucho mayor al por defecto para que el proceso pudiera terminar.

### Resultados

Podemos ver en la Tabla 3.1 la cantidad de reglas de asociación generadas para cada familia. Existe una diferencia de varias órdenes de magnitud entre la cantidad de reglas obtenidas utilizando las familias comparables. `LRR1` es la familia que más reglas generó, con 1.657.444 reglas. Le siguen `NEWAnk` con 4.888 reglas y `TPR1` con 4 reglas. Las familias de control no generaron ninguna regla.

| **Familia** | **#Reglas** |
|---|---|
| `NEWAnk` | 4.888 |
| `TPR1` | 4 |
| `LRR1` | 1.657.444 |
| `NEWAnk_SCRAMBLED` | 0 |
| `NEWAnk_UNIFORM` | 0 |

*Tab. 3.1:* Cantidad de reglas de asociación por familia, generadas utilizando el software de Enriquez con los parámetros mostrados en la Tabla 2.2.

### Discusión

Si bien suponíamos que no íbamos a obtener exactamente la misma cantidad de reglas de asociación, esperábamos que para familias similares (en cuanto cantidad y largo de secuencias) íbamos a obtener al menos la misma magnitud en cantidad de reglas. Esto no ocurrió. Como se puede ver en la Tabla 3.1, existen órdenes de magnitud de diferencia entre `NEWAnk`, `LRR1` y `TPR1`.

Como estamos usando familias comparables, esto nos hace pensar que la causa de dichas diferencias se encuentran en los patrones presentes en las secuencias y no en las propiedades de las familias (cantidad y largo de las secuencias).

En referencia a las familias control, al utilizar las mismas como fuente de datos no se obtienen reglas de asociación. Esto coincide con lo que esperábamos ya que las repeticiones maximales se reparten más uniformemente rompiendo patrones de asociación. Esto último es interesante, porque soporta la idea de que las reglas de asociación están capturando algún tipo de patrón biológico no presente en cadenas azarosas.

## 3.2. Frecuencias individuales

De la sección anterior nos surge la pregunta si se puede explicar la diferencia de cantidad de reglas generadas en base a la cantidad de ocurrencias de los ítems (MRs) en las transacciones.

Recordemos que el soporte de un ítem es la cantidad de veces que aparece el ítem en las transacciones, dividido por la cantidad de transacciones en la familia para poder normalizar. Además, recordemos que en el algoritmo *Apriori* se utiliza el parámetro `min_suport`, que determina cuáles ítems usar de acuerdo a la cantidad de veces que aparecen (es decir, de acuerdo a su soporte). Por ejemplo, si se define el `min_support` en 0.025, se usarán ítems que aparezcan en al menos el 2,5 % de las transacciones (es decir, ítems cuyo soporte sea al menos de 0.025). En el primer paso del algoritmo *Apriori* se eliminan todos aquellos ítems que no superen dicho umbral.

### Metodología

Tomamos las transacciones generadas por las familias comparables y calculamos el soporte para cada ítem. Luego, analizamos la distribución de los soportes con un histograma. En otras palabras, vimos cuántos ítems hay con un cierto soporte, agrupando los soportes en buckets para poder visualizar la distribución. Si se quisiera saber cuántos son los ítems que superan el `min_support` (por ejemplo, 0.025 según las secciones anteriores) basta con sumar la cantidad de ítems que aparecen a la derecha de dicho valor.

Consideramos para este experimento los soportes entre 0 y 0.10, y dividimos el histograma en 10 buckets iguales. Pueden existir ítems con soporte mayor a 0.10 pero no fueron tenidos en cuenta en las figuras. Además, repetimos el experimento con las familias de control.

### Resultados

En la Figura 3.1 podemos observar un histograma con la distribución de los soportes de los ítems, en 10 buckets, para nuestras familias comparables. El gráfico se encuentra en escala logarítmica en el eje Y. Se observa un claro descenso en las cantidades a medida que los soportes aumentan, lo que resulta esperable. En todos los buckets con soportes mayores al `min_support` (0.025), `LRR1` supera en cantidad de ítems a las demás familias. El bucket comprendido entre 0.01 y 0.02 es el único en donde esta superioridad no se mantiene y `LRR1` es la familia con menor cantidad de ítems. `NEWAnk` se mantiene segunda en cantidad de ítems en casi todos los buckets excepto en los soportes entre 0.03 y 0.05, donde es superada levemente por `TPR1`.

En la Figura 3.2 podemos observar un experimento idéntico pero utilizando `NEWAnk` y las familias de control. El gráfico se muestra en escala logarítmica en el eje Y. La

variante `UNIFORM` solo presenta ítems en los soportes más bajos, entre 0 y 0.01. La variante `SCRAMBLED` presenta ítems en soportes mayores, pero su cantidad parace descender más rápido que los de `NEWAnk`. La variante `SCRAMBLED` posee menos de 10 ítems en los buckets con soportes superiores a 0.06, y para soportes superiores a 0.09 ya no presenta ítems.

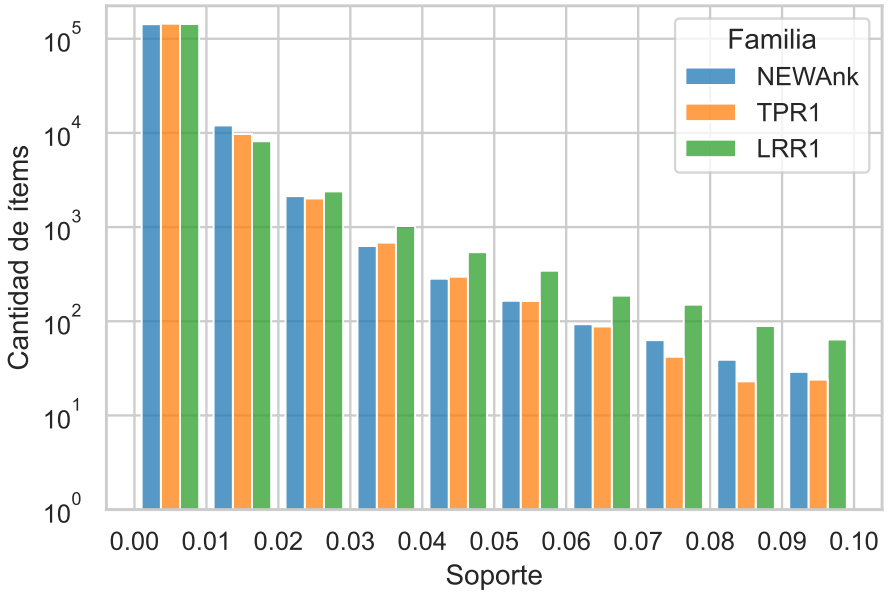



*Fig. 3.1:* Histograma con la distribución de los soportes de los ítems, entre 0 y 0.1, en 10 buckets, para las familias comparables. Se muestra la cantidad de ítems presentes para cada rango de soportes. El gráfico se encuentra en escala logarítmica en el eje Y.

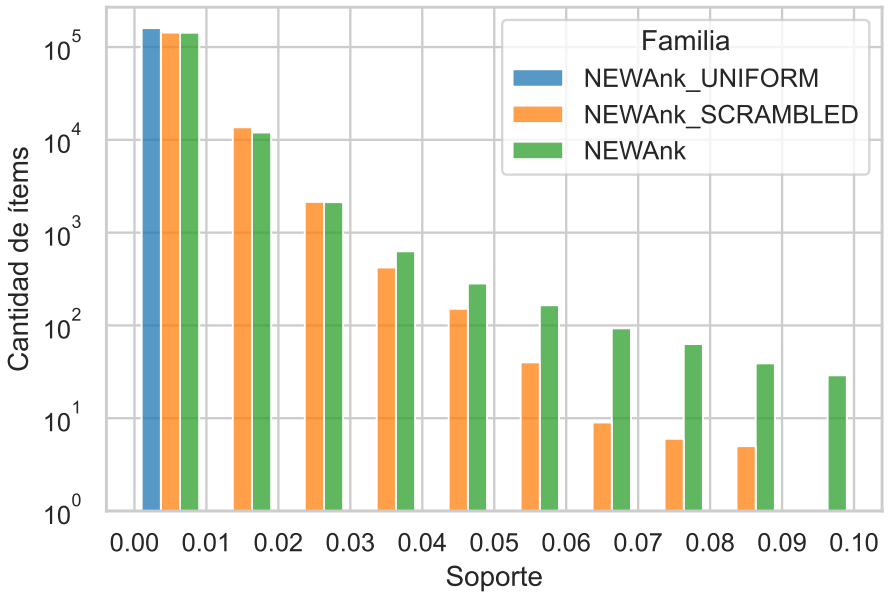



*Fig. 3.2:* Histograma con la distribución de los soportes de los ítems, entre 0 y 0.1, en 10 buckets, para las familias de control. Se muestra la cantidad de ítems presentes para cada rango de soportes. El gráfico se encuentra en escala logarítmica en el eje Y.

### Discusión

La cantidad de ítems en las transacciones decrece a medida que se consideran soportes cada vez más grandes. Esencialmente, son pocos los MRs que aparecen muchas veces.

Si comparamos los resultados entre las familias, podemos observar que la cantidad de ítems en las transacciones de `LRR1` son mayores a las otras familias, acentuándose la diferencia al superar el `min_support` que fijamos anteriormente al generar las reglas (0.025). Resulta esperable que `LRR1` tenga mayor cantidad de ítems que superen el umbral, ya que es la familia que generaba más reglas, con órdenes de magnitud de diferencia. Lo que no es tan claro es por qué las curvas que se forman tienen una caída tan marcada. Estos decrecimientos podrían tener una caída exponencial, pero queda como trabajo futuro analizarlos con mayor detalle y ver si explican algún otro fenómeno.

Las diferencias de frecuencias en ítems entre las familias no se explican ni por la cantidad de proteínas ni por la longitud mediana de ellas (valores en Tabla 2.3). `TPR1` tiene valores mayores que `NEWAnk` en ambas características, pero sin embargo frecuencias de ítems menores. Esta falta de relación puede verse con claridad al analizar las familias de control (Figura 3.2). Ellas tienen características idénticas a `NEWAnk` por la forma en la que fueron generadas, y sin embargo las diferencias en cantidades de ítems son muy apreciables. La razón por la cuál las características de las familias no influyen es porque los ítems son los MRs, y estos dependen de las secuencias de aminoácidos que componen a sus familias y no estrictamente del largo o la cantidad de proteínas.

Un punto que nos llama la atención es la relación entre `NEWAnk` y `TPR1`. Nos sorprende que por momentos parece que `TPR1` tiene más ítems en buckets mayores al umbral de soporte mínimo (por ejemplo, en el bucket que considera soportes entre 0.03 y 0.04). Hipotetizábamos que existiría una diferencia marcada entre ambas familias (esperábamos que las barras de `NEWAnk` siempre se sitúen por encima de `TPR1`), como sucede con `LRR1`. Esto, a causa de que con `TPR1` sólo se generaron 4 reglas y con `NEWAnk` 4.888 (ver Tabla 3.1). Sin embargo la evidencia muestra otro escenario.

En el algoritmo *Apriori*, se utiliza el parámetro `min_support` para filtrar en la primer iteración aquellos ítems individuales cuyo soporte no supere ese umbral. En la segunda iteración del algoritmo, se intenta expandir estos ítems a ítemsets de tamaño 2. Es decir, se toman todos los **pares** de ítems que superan el `min_support`. Son recién estos conjuntos de dos elementos los primeros considerados para constituir reglas, porque toda regla se compone como mínimo de dos ítems (antecedente y consecuente). Por lo tanto, la frecuencia individual de los ítems no puede ser la única relevante en la generación de reglas, sino que es necesario superar también el umbral en las frecuencias de los ítems tomados de a pares.

Finalmente, luego de este experimento concluimos que la frecuencia individual de ítems no es suficiente para explicar la cantidad de reglas generadas y por lo tanto nos propusimos analizar frecuencias de pares de ítems.

## 3.3. Frecuencias de pares

Intentamos explicar por qué nos encontramos con diferencias de varios órdenes de magnitud en la cantidad de reglas de asociación generadas, a pesar de estar usando familias elegidas por ser similares entre sí. En la sección anterior, concluimos que analizar únicamente las frecuencias de los ítems de forma individual no es suficiente, y decidimos realizar el análisis pero considerando pares de ítems.

### Metodología

Un posible algoritmo que se podría implementar para calcular la frecuencia de todos los ítemsets de tamaño 2 es el que se propone en la Figura 3.3. En el mismo se puede observar que para cada par posible de ítems y para cada transacción, sumar 1 a un contador si el par está presente en dicha transacción. Si llamamos $M$ a la cantidad de ítems distintos y $T$ la cantidad de transacciones, la complejidad computacional es $O(M^2 * T)$.

Para el caso de `NEWAnk`, con transacciones generadas con `min_len` 4, tenemos que $M = 158.107$ y $T = 32.169$. Con estos valores, el algoritmo en la práctica no termina de ejecutarse (incluso introduciendo algunas optimizaciones simples, como evitar iterar por los pares simétricos).

```
def calcular_frecuencias_pares_naive():
    for mr_a in mrs:
        for mr_b in mrs:
            for tx in transactions:
                if {mr_a, mr_b} in tx:
                    frequency[{mr_a,mr_b}] += 1

```

*Fig. 3.3:* Propuesta de algoritmo para calcular frecuencias de pares. En la práctica, debido a la gran cantidad de ítems distintos y transacciones que poseen las familias de proteínas a analizar, nunca termina de ejecutar.

Recordemos que decimos que un ítem es frecuente cuando su soporte es mayor al `min_support` (Definición 1.2.9). Sabemos que por el *Principio Apriori* (Definición 1.2.10), para que un par de ítems sea frecuente, ambos deben ser frecuentes individualmente. Por lo tanto, no tiene sentido considerar en los cálculos a los ítems no-frecuentes si queremos entender cómo es la distribución de los pares que superen el `min_support`. Es por esto que para optimizar el cálculo, iteramos únicamente por aquellos pares formados por ítems frecuentes.

Con esta optimización, al fijar el valor de `min_support` en 0.025, se tiene que para la familia `NEWAnk` la cantidad de ítems distintos en las transacciones, generadas con `min_len` 4, se reduce de 158.107 a 2.130 (es decir, ahora $M$ =2.130). Aunque es una buena optimización, la complejidad computacional del algoritmo sigue siendo la misma y el tiempo de ejecución en la práctica sigue resultando excesivo.

Para calcular las frecuencias de los ítemsets de tamaño 2, nos desviamos de la idea anterior y nos concentramos en analizar las transacciones. Pueden verse en la Tabla 3.2 algunas estadísticas para los largos de las transacciones en `NEWAnk` si nos quedamos únicamente con los ítems frecuentes.

| | |
|---|---|
| Cantidad de transacciones | 32.169 |
| Promedio de cantidad de ítems por transacción | 97,6 |
| Desvío estándar de la cantidad de ítems por transacción | 46,47 |

*Tab. 3.2:* Estadísticas para las transacciones de `NEWAnk` generadas con `min_len` 4, luego de eliminar ítems no-frecuentes al considerar un `min_support` de 0.025.

Como puede verse en la Tabla 3.2, seguimos teniendo 32.169 transacciones, pues solo modificamos la cantidad de ítems dentro de ellas, pero lo interesante es que ahora, en promedio, cada transacción contiene alrededor de 100 ítems.

Dado que en las transacciones no todos los ítems están presentes, tomar un cierto ítem y buscarlo iterando entre todas las transacciones no es muy eficiente. Los ítems que modifican las frecuencias son únicamente aquellos que sí están presentes en las transacciones.

En base a esta observación, proponemos un nuevo algoritmo, que puede verse en la Figura 3.4. Para cada transacción se generan todos sus pares de ítems presentes, sumando 1 a la frecuencia total de cada par. Hacer esto resulta mucho más eficiente que antes, porque la iteración es sobre pares que sí o sí están presentes, pues los estamos tomando de una misma transacción.

```
def calcular_frecuencias_pares(transacciones_de_frecuentes):
    for tx in transacciones_de_frecuentes:
        for mr_a in tx:
            for mr_b in tx:
        # (En la implementacion real, considero solo a la derecha de mr_a)
                frecuencia[{mr_a,mr_b}] += 1

```

*Fig. 3.4:* Algoritmo para calcular frecuencias de pares. Para cada transacción (formada por ítems frecuentes) se generan todos sus pares de ítems presentes, sumando 1 a la frecuencia total de cada par.

Si llamamos $P$ al largo promedio de las transacciones (es decir, la cantidad promedio de ítems en ellas) la complejidad en el caso promedio es de $O(P^2 * T)$. Para el caso de `NEWAnk`, tenemos que $T = 32.169$ y $P \approx 100$, lo que finalmente posibilita al programa finalizar en un tiempo razonable.

Tomamos las transacciones de las familias comparables, eliminamos todos los ítems no-frecuentes al considerar un `min_support` de 0.025 y, utilizando el algoritmo descripto en Figura 3.4, calculamos el soporte para cada par de ítems. Luego, realizamos un histograma para observar la distribución de los soportes de pares de ítems, entre los valores de 0 y 0.10, utilizando 10 buckets iguales. En este histograma puede observarse, para cada rango de soporte, cuántos pares de ítems tiene su soporte en dicho rango. No mostramos el gráfico de las familias de control, ya que al no poseer ítems frecuentes, no aparece ningún par de ítems en el histograma.

### Resultados

Se muestra en la Figura 3.5 un histograma con la distribución de los soportes en pares de ítems, usando ítems frecuentes, en nuestras familias comparables.

Al igual que al analizar frecuencias de ítems individuales, las cantidades de pares muestran una clara curva decreciente a medida que consideramos soportes cada vez más altos. Las curvas con las que decrece cada familia podrían responder a caídas exponenciales con diferentes coeficientes, pero queda como trabajo futuro profundizar ese análisis.

En soportes superiores a 0.02, se aprecia una diferencia en orden de magnitud entre las familias. El orden de cantidades entre las familias se mantienen consistentes con el orden en en cantidad de reglas que cada una generó: `LRR1` es la que más pares de ítems acumula,

seguida de `NEWAnk` y luego por `TPR1` (Tabla 3.1).

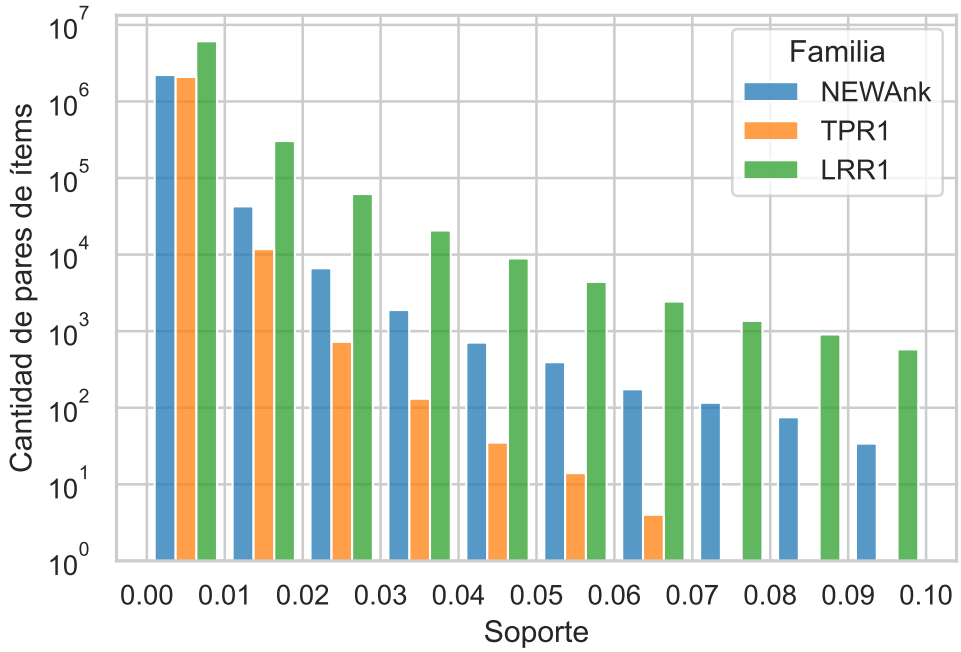



*Fig. 3.5:* Histograma con la distribución de los soportes en pares de ítems, usando ítems frecuentes, en nuestras familias comparables. Se muestra la cantidad de pares de ítems presentes para cada rango de soportes. Escala logarítmica en el eje Y.

### Discusión

Consideramos que el gráfico presentado en la Figura 3.5 es esencial para poder entender las diferencias en la generación de reglas. Puede verse con claridad la distinción, en órdenes de magnitud, de las cantidades de pares de ítems frecuentes en las diferentes familias. A diferencia del gráfico analizado en la sección anterior (Figura 3.1), aquí puede apreciarse una diferencia real entre `NEWAnk` y `TPR1`. Las diferencias de magnitudes entre las tres familias son consistentes con la cantidad de reglas que se generaron a partir de cada una de ellas (Tabla 3.1).

Si nos centramos en `TPR1`, a partir de la cual se habían generado sólo 4 reglas, vemos en el gráfico que posee una cantidad de pares que superan el `min_support` del orden de cientos. ¿A qué se debe que no se generaron cientos de reglas? Es importante entender que el soporte es un primer filtro, pero no el único: en una etapa posterior del algoritmo *Apriori* se eliminarán eliminan los pares que no superan el umbral de confianza fijado (`min_confidence`).

A partir del experimento propuesto concluimos que, para las familias que estamos estudiando, las frecuencias de ítemsets de tamaño 2 nos dan una explicación razonable a las diferencias en las cantidades de reglas generadas en familias comparables.

## 3.4. Importancia de las frecuencias de pares

En la Sección 3.3 vimos cómo con las frecuencias de pares de ítems pudimos encontrar una posible explicación a las diferencias en la cantidad de reglas generada por familia

(Tabla 3.1). Ahora nos preguntamos, ¿las frecuencias de pares de ítems son suficientes para poder determinar (o predecir) la cantidad de reglas generadas?

Sabemos que si fijamos el `min_support` en 0.025 y `min_len` en 4, se generan 4.888 reglas con la familia `NEWAnk`. Nos propusimos determinar la cantidad de reglas generadas para `TPR1`. Si suponemos que las frecuencias de pares son determinantes en la generación de reglas y si encontramos un nuevo valor de `min_support` adecuado para que en `TPR1` haya la misma cantidad de pares frecuentes que había en `NEWAnk`, entonces con ese nuevo valor de `min_support` debería generarse la misma cantidad de reglas habíamos formado al usar `NEWAnk`.

### Metodología

Para explicar la metodología de esta sección, nos apoyamos en el gráfico de la Figura 3.6, donde mostramos un histograma con la cantidad de pares de ítems que hay con un cierto soporte, para las familias `NEWAnk` y `TPR1`. Este histograma está dividido en 10 buckets y muestra, para los soportes entre 0 y 0.05 cuántos pares de ítems pueden encontrarse en cada rango de soportes. Recordemos que el valor de `min_support` que estábamos usando era de 0.025, y es por esto que marcamos este valor con una línea vertical roja. La cantidad de pares ítems que superan al `min_support` equivale a sumar las cantidad de ítems que se encuentran a la derecha de la línea roja. Aunque el límite derecho que mostramos en el gráfico es 0.05, existen pares de ítems con soportes mayores, que no son mostrados para evitar distorsionar el gráfico, pero son tenidos en cuenta en los cálculos que presentamos a continuación.

Definimos $S[familia, minsup]$ como la cantidad de pares de ítems en las transacciones de $familia$ cuyo soporte es mayor a $minsup$. Para el caso de `NEWAnk`, si contamos cuántos pares de ítems tienen soportes mayores al `min_support` (0.025) resulta que $S$[`NEWAnk`, 0.025]=7.157.467. Podemos pensar a este valor como la cantidad de pares de ítems frecuentes.

Queremos encontrar para la familia `TPR1`, el valor de $minsup$ para que $S$[`TPR1`, $minsup$] sea igual a $S$[`NEWAnk`, 0.025]. Dicho de una forma más gráfica, queremos ver en qué lugar deberíamos colocar la línea roja de la Figura 3.6 para que los elementos a su derecha en `TPR1` ($S$[`TPR1`, $minsup$]) también sumen $S$[`NEWAnk`, 0.025]. Una vez encontrado este soporte $minsup$ para `TPR1`, ejecutamos el algoritmo *Apriori* fijando `min_support` en ese valor, y vemos si efectivamente se generaron las mismas reglas que tenía `NEWAnk` originalmente.

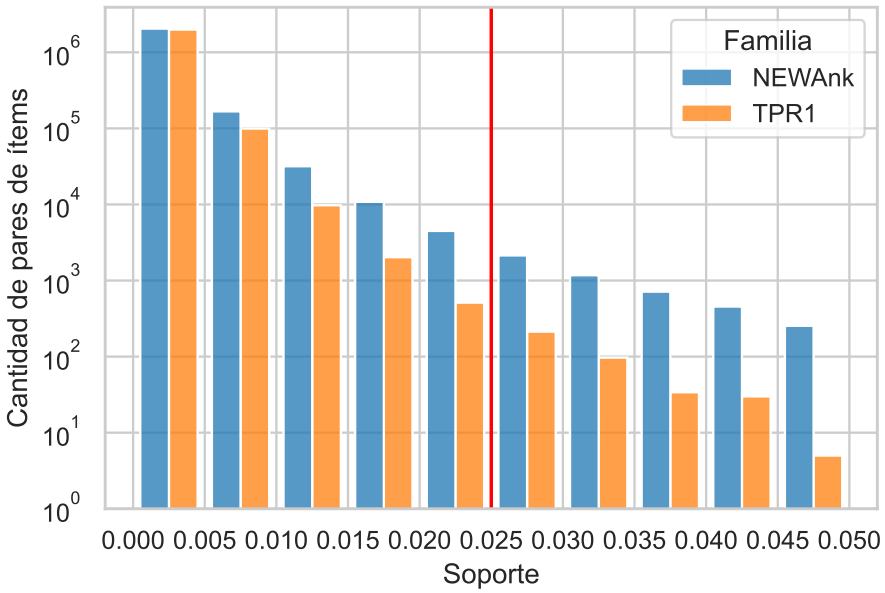



*Fig. 3.6:* Histograma con la distribución de los soportes en pares de ítems, usando ítems frecuentes, en las familias `NEWAnk` y `TPR1`. Escala logarítmica en el eje Y. La línea vertical roja es el punto de corte para `min_support`=0.025.

Para calcular el *minsup* de $S$[`TPR1`, *minsup*] podemos asumir que las frecuencias de pares son decrecientes con respecto al soporte y usar un algoritmo de búsqueda binaria. Esta asunción es razonable y útil en la práctica con las familias con las cuáles experimentamos, pero podría no ser cierta en algún conjunto particular de secuencias. Por ejemplo, si tuviésemos un conjunto de secuencias donde muchos ítems aparecen en casi todas las transacciones (teniendo entonces un soporte alto) y donde fueran pocos los ítems de pocas apariciones.

Presentamos un pseudocódigo en la Figura 3.7, cuyos parámetros son los siguientes:

- `S_objetivo`: La suma objetivo que queremos encontrar ($S$[`NEWAnk`, 0.025]).
- `left_support`: Límite inferior de soporte para la búsqueda. Soportes muy bajos (experimentalmente menores a 0.005) provocan que el algoritmo no termine al contar la cantidad de pares.
- `right_support`: Límite superior de soporte para la búsqueda. Como `TPR1` genera menos reglas que `NEWAnk`, sabemos que debemos buscar un `min_support` menor a 0.025 para poder generar más reglas, así que lo fijamos en ese valor.
- `iteraciones`: Cantidad de iteraciones para la búsqueda binaria. Decidimos utilizar una cantidad de iteraciones fijas en vez de detener la búsqueda al hallar una suma exacta porque en todas las variantes probadas la suma exacta no fue posible de generar. Observamos experimentalmente que con 10 iteraciones era suficiente para converger.
- `familia`: La familia sobre la cuál queremos hacer la búsqueda. En nuestro caso, `TPR1`.

```
def calcular_min_support_para_suma(S_objetivo, left_support, right_support,
    iteraciones, familia):

    # Obtiene las transacciones de 'familia'
    # formadas por items con soporte mayor a 'left_support'
    transacciones = transacciones_filtradas_por_frecuencia(
        familia, left_support)

    # Calcula frecuencias de pares usando items frecuentes
    freq_pares_data = calcular_frecuencia_pares(transacciones)

    for iteracion in range(iteraciones):
        query_support = (left_support + right_support) / 2

        # Calcula la suma de las frecuencias
        Sres = sumar_pares_mayores(query_support, freq_pares_data)

        if Sres < S_objetivo:
            # Support buscado esta mas a la izquierda
            right_support = query_support
        else:
            # Busco supports mas grandes.
            left_support = query_support

    return query_support
```

*Fig. 3.7:* Pseudocódigo de la búsqueda binaria usada en hallar el soporte que genera `S_objetivo` pares de ítems frecuentes en `familia`. El soporte se buscará entre `left_support` y `right_support`

En el primer paso del algoritmo, antes de realizar la búsqueda binaria, se calculan las frecuencias de pares de la misma forma que en la Sección 3.3. Recordemos que en el primer paso del método se filtran los ítems individuales que no superan un cierto soporte mínimo. Como buscamos pares cuyo soporte sea mayor a `left_support`, entonces por el Principio *Apriori* los ítems individuales también deben superar `left_support` y es por eso que lo pasamos como parámetro en la función `transacciones_filtradas_por_frecuencia`.

Calculamos $S$[`NEWAnk`, 0.025] y utilizamos el algoritmo descripto para obtener un soporte *minsup* tal que $S$[`TPR1`, *minsup*] sea igual a $S$[`NEWAnk`, 0.025]. Luego, generamos reglas de asociación para `TPR1` fijando en `min_support` el resultado obtenido y comparamos estas reglas con las generadas por `NEWAnk`.

### Resultados

Podemos ver los resultados de correr el algoritmo para `NEWAnk` y `TPR1` con `min_len` 4 en la Tabla 3.3. El `min_support` necesario para que `TPR1` tenga aprox. la misma cantidad de pares frecuentes que tenía `NEWAnk` es 0.0096. Utilizando ese `min_support`, se generaron 11.152 reglas. La cantidad de reglas generadas con `TPR1` es un poco más del doble que las generadas por `NEWAnk`.

| Familia | min_support | #Pares frecuentes | #Reglas |
|---|---|---|---|
| NEWAnk | 0,025 | 7.157.467 | 4.888 |
| TPR1 | 0,0096 | 7.150.342 | 11.152 |

*Tab. 3.3:* Cantidad de reglas obtenidas en NEWAnk y TPR1 usando en cada una el min_support que genera una cantidad similar de pares frecuentes para ambas familias. min_len=4

Repetimos el experimento pero esta vez utilizando min_len 5 y 6, cuyos resultados pueden verse en las Tablas 3.4 y 3.5, respectivamente. Podemos observar que los min_support necesarios en TPR1 para obtener la misma cantidad de pares que en NEWAnk es cada vez menor al aumentar el min_len. La diferencia en cantidad de reglas generadas entre las dos familias aumenta ahora en órdenes de magnitud, siendo de 174 contra 26.986.304 en min_len 5, y 13 contra 151.283.346 en min_len 6.

| Familia | min_support | #Pares frecuentes | #Reglas |
|---|---|---|---|
| NEWAnk | 0,025 | 484.060 | 174 |
| TPR1 | 0,0076 | 477.326 | 26.986.304 |

*Tab. 3.4:* Cantidad de reglas obtenidas en NEWAnk y TPR1 usando en cada una el min_support que genera una cantidad similar de pares frecuentes para ambas familias. min_len=5

| Familia | min_support | #Pares frecuentes | #Reglas |
|---|---|---|---|
| NEWAnk | 0,025 | 50.696 | 13 |
| TPR1 | 0,0043 | 36.486 | 151.283.346 |

*Tab. 3.5:* Cantidad de reglas obtenidas en NEWAnk y TPR1 usando en cada una el min_support que genera una cantidad similar de pares frecuentes para ambas familias. min_len=6.

### Discusión

En estos experimentos mostramos que aunque la cantidad de pares que supere el min_support sea similar entre las dos familias, la cantidad de reglas generadas difiere en varias órdenes de magnitud. Esta diferencia se incrementa a medida que aumentamos el valor de min_len.

Hay varios motivos por los cuales podemos explicar estos resultados. Si bien supusimos que la cantidad de pares frecuentes podían ser determinantes en la generación de reglas, lo que sucede es que al tomar soportes muy bajos (para lograr igualar la suma a derecha) hay muchos ítems individuales (ítemsets de tamaño 1) que ahora superan el umbral y que antes no lo hacían. Al haber más ítems que superan este umbral, aparecen también muchos grupos frecuentes de 3, 4 y 5 ítems. Considerar grupos de varios elementos provoca que la cantidad de combinaciones sea mucho mayor a tomar sólo pares y por lo tanto existe una mayor cantidad de posibles reglas a generar. Recordemos que en el experimento solo tuvimos en cuenta las frecuencias de pares y no las frecuencias de triplas o tuplas mayores, por lo que el método utilizado resulta incompleto.

Otra razón que podría influir en la discrepancia en la cantidad de reglas es que, aunque las cantidades de pares frecuentes sean similares, no se está teniendo en cuenta otro

parámetro importante del algoritmo: `min_confidence`. De todos modos, en todos nuestros experimentos esto se encuentra fijo por lo que podría no afectar si no está en función del `min_support`.

### Conclusión

Los resultados obtenidos evidencian que las frecuencias de pares no son lo determinante a la hora de entender la cantidad de reglas generadas, sino que también hay que tener en cuenta las frecuencias de ítemsets de tamaños mayores. Hacer un análisis de grupos mayores genera tantas combinaciones de ítems que no encontramos un método efectivo para analizar estas frecuencias. Además, al enfocarnos únicamente en `min_support`, estamos dejando de lado `min_confidence`, que podría ser estudiada en un trabajo futuro.

Si se quisiera generar una cantidad de reglas específica, al no poder "predecir" el resultado usando los soportes, el método más confiable que encontramos fue correr el *Apriori* y decidir sus parámetros a partir de la cantidad de reglas obtenidas en la corrida anterior, haciendo una búsqueda binaria en el `min_support` (teniendo cuidado de no bajar demasiado el soporte ya que si no, no terminará). Queda como trabajo futuro pensar y explorar otros métodos para poder seleccionar parámetros automáticamente (o de forma parcialmente automatizada) sin tener que repetir el proceso cada vez.

# 4. COMPARACIÓN DE REGLAS DE ASOCIACIÓN ENTRE FAMILIAS

## 4.1. Reglas de asociación en común

En el capítulo anterior analizamos las diferencias vistas entre las cantidades de reglas de asociación generadas por las distintas familias comparables. Para esto, estudiamos las frecuencias de los ítemsets de 1 y de 2 elementos. Ahora, nos proponemos analizar las reglas en sí, ¿existen entre las familias reglas en común?

### Metodología

Debemos tener en cuenta a la hora de comparar reglas de asociación que, como los ítemsets son conjuntos, dos reglas pueden estar escritas de modo diferente pero representar la misma regla. Por ejemplo, la regla

$$\{\text{GADPN, LEAGA}\} \rightarrow \{\text{LLEAG}\}$$

es equivalente a

$$\{\text{LEAGA, GADPN}\} \rightarrow \{\text{LLEAG}\}$$

Generamos las reglas de asociación con las familias comparables. Utilizamos en el primer experimento los valores por defecto (Tabla 2.2) y luego repetimos las pruebas variando el valor de `min_len` entre 4 y 7, y variamos `min_support` entre 0.010 y 0.050. Teniendo cuidado con las equivalencias entre reglas, analizamos las intersecciones que se produjeron entre los conjuntos generados.

### Resultados

Los resultados muestran que no se generan reglas de asociación equivalentes entre ninguna de las familias comparables, para ninguna de las combinaciones de parámetros utilizados.

### Discusión

Los resultados son interesantes. En base a los trabajos Turjanski2016 y Turjanski2018 sospechábamos que las familias se diferenciarían, pero no esperábamos la ausencia total de reglas en común. Aunque nuestro primer pensamiento fue relacionarlo a los parámetros por defecto que utilizamos, vimos que esto no era determinante porque el fenómeno se repitió con todas las combinaciones de parámetros utilizados. En las siguientes secciones intentaremos entender por qué es que esto sucede.

## 4.2. Repeticiones maximales en común

Nos proponemos entender por qué no existen reglas en común entre familias comparables. Los MRs son el lugar natural para comenzar el análisis, dado que son ellos los elementos que conforman las reglas. Nos preguntamos si las familias comparten MRs, dado

que si no comparten MRs entonces sería lógico que tampoco compartan reglas. También nos preguntamos, ¿tiene cada familia un conjunto de MRs propio? ¿O acaso comparten MRs pero varían su frecuencia?

### Metodología

Tomamos todos los ítems de las transacciones (es decir, los MRs de las secuencias) y vimos cuántos de ellos son comunes entre las familias. Como queremos entender cómo se asemejan las familias, contamos solo los MRs (es decir, la cantidad de MRs distintos) y no la cantidad de instancias que ellos poseen.

### Resultados y discusión

Presentamos en la Tabla 4.1 la cantidad de MRs que comparten las familias indicadas. Los MRs fueron generados tomando un umbral de `min_len` de 4. Observamos que las familias poseen entre 156.549 (`LRR1`) y 158.107 (`NEWAnk`) MRs. Las tres familias comparten 154.157 MRs, que representa el 97 % de los MRs presentes en `NEWAnk`.

| `NEWAnk` | `TPR1` | `LRR1` | **Cantidad de MRs compartidos entre las familias seleccionadas** |
|---|---|---|---|
| x | | | 158.107 |
| | x | | 157.521 |
| | | x | 156.549 |
| x | x | | 156.340 |
| x | | x | 155.456 |
| | x | x | 154.910 |
| x | x | x | 154.157 |

*Tab. 4.1:* Cantidad de MRs en las transacciones de las familias comparables, usando `min_len` 4 y refinamiento *substring*. No se tiene en cuenta la cantidad de instancias. En cada fila se marca con una x aquellas familias que se tomaron para calcular la intersección.

La cantidad de MRs en cada familia es similar y casi todos ellos se encuentran presentes en las intersecciones, tanto en las intersecciones tomadas de a dos, como en la intersección de las tres familias. Antes de proponer una explicación, haremos una observación. Aunque el `min_len` representa el tamaño mínimo de los MRs incluidos en las transacciones, resulta en la práctica que, a causa del refinamiento, prácticamente todos los MRs tienen un largo que coincide con el `min_len`. Presentamos un análisis más detallado en la Sección 5.1.

Para explicar por qué casi todos los MRs se encuentran en la intersección, debemos tener en cuenta es que existen 20 tipos de aminoácidos presentes en las proteínas. Si consideramos una longitud de 4 en los MRs, existen $20^4$ (=160.000) ítems posibles. Cuando dos familias distintas tienen alrededor de 150.000 MRs cada una, pensando en el principio del palomar[1], tiene que suceder que la mayor parte de ellos se encuentre en la intersección.

Es importante notar que si bien casi todos los MRs se encuentran en la intersección de las familias, este resultado no nos habla ni de su frecuencia (cuántas instancias tienen) ni de su distribución (cuántos son los MRs que tienen muchas instancias). Ya habíamos visto

[1] `https://es.wikipedia.org/wiki/Principio_del_palomar`

en la Sección 3.3 que las frecuencias (de ítems y de pares de ítems) son importantes para entender la cantidad de reglas generadas y éstas tienen mucha variación entre las familias.

La pregunta que nos surge luego de ver el anterior resultado es qué pasa si generamos MRs tomando un `min_len` mayor. Esto aumentaría la cantidad potencial de MRs (por lo explicado de que la mayoría de sus largos coinciden con este parámetro) y esperamos que al haber más posibles MRs, haya también suficiente lugar como para que las familias puedan diferenciarse un poco más.

En la Tabla 4.2 mostramos los resultados de repetir el experimento pero fijando `min_len` en 5. Aquí podemos ver cómo `NEWAnk` y `TPR1` tienen una cantidad similar de MRs, alrededor de 1.670.000. La familia `LRR1` es la que menos tiene, con casi 1.300.000 MRs. Al subir el `min_len` de 4 a 5, la cantidad de MRs de `LRR1` no aumentó en la misma proporción que las demás familias, lo que podría indicar que en `LRR1` los MRs se concentran en largos menores, pero es algo que debemos estudiar mejor.

La intersección entre `NEWAnk` y `TPR1` es la que más MRs tiene, con aprox. 1.250.000. El resto de las intersecciones tomadas de a dos tienen cerca de 1.000.000 de MRs en común, mientras que la intersección de las tres familias tiene aprox. 860.000 MRs.

| `NEWAnk` | `TPR1` | `LRR1` | **Cantidad de MRs compartidos entre las familias seleccionadas** |
|---|---|---|---|
| x | | | 1.670.107 |
| | x | | 1.674.199 |
| | | x | 1.299.696 |
| x | x | | 1.252.358 |
| x | | x | 1.008.665 |
| | x | x | 996.533 |
| x | x | x | 861.715 |

*Tab. 4.2:* Cantidad de MRs en las transacciones de las familias comparables, usando `min_len` 5 y refinamiento *substring*. No se tiene en cuenta la cantidad de instancias. En cada fila se marca con una x aquellas familias que se tomaron para calcular la intersección.

Debemos tener en cuenta es que tiene sentido haya mayor cantidad MRs con `min_len` 5 que con `min_len` 4, cuando parecería que debe ser al revés. ¿Los MRs de longitud mínima 4 no incluyen acaso a los MRs de longitud 5? La respuesta es que en la práctica no. La explicación es que debemos recordar que estos MRs son tomados de transacciones que pasaron por una etapa de refinamiento, según lo explicado en la Sección 1.4. En particular, se utilizó el refinamiento en modo *substring*, que elimina todos los MRs que tienen como substring a algún otro. Al generar transacciones con `min_len` 4, resulta que casi todos los MRs de largo 4 aparecen como substrings de MRs de largos mayores, por lo que aplicar el refinamiento nos quedamos casi solo con MRs de largo 4, explicando la diferencia de cantidades. El mismo fenómeno se repite cuando usamos `min_len` 5. Analizaremos las distribuciones de largos con respecto al `min_len` en un capítulo posterior.

Para el caso de largo 5, la cantidad de MRs posible es $20^5 = 3.200.000$. Comparado a usar largo 4, se tiene mayor libertad para generar ítems (porque el espacio de secuencias posibles es más grande). De todos modos, deberíamos tener en cuenta que en la práctica los 20 aminoácidos no aparecen con la misma frecuencia, sino que existen algunos que aparecen muy pocas veces (por ejemplo la Y o la W) y sería menos probable encontrar

proteínas que lo incluyan frecuentemente como una repetición maximal. Es decir, en la realidad, podría ser que estas familias estén cercanas a generar la máxima cantidad de MRs posibles. No es sorprendente que compartan al menos la mitad de sus repeticiones maximales.

### Conclusión

Casi todos los MRs se encuentran en común si fijamos el valor de `min_len` en 4. Para largos mayores, la cantidad de ítems en común disminuye, pero siempre se mantiene una gran cantidad de patrones en común, así que no se explica en base a estas cantidades el por qué no se tienen reglas iguales a pesar de compartir MRs. Como sabemos que las frecuencias son muy importantes para la generación de reglas, decidimos explorar qué sucede cuando las tenemos en cuenta al momento de comparar ítems.

## 4.3. Repeticiones maximales frecuentes en común

Los MRs en común son muchos, pero una gran parte de ellos son potencialmente descartados en la primera iteración del algoritmo *Apriori* al comparar su frecuencia con el `min_support`. ¿Cuántos de estos MRs se mantendrán en común entre las familias si solo consideramos los frecuentes?

### Metodología

Para definir cuándo un ítem tiene alta frecuencia, consideramos la cantidad de transacciones de `NEWAnk` (32.169) y el `min_support` por defecto (0,025). Decimos que un ítem tiene alta frecuencia cuando aparece más de $804 (\approx 0{,}025 * 32.169)$ veces en las transacciones. Es decir, consideraremos aquellos MRs que tienen más de 804 instancias. Consideramos que la comparación usando el mismo umbral para todas las familias no es injusta, ya que la cantidad de transacciones y la cantidad de MRs son muy similares en las tres familias. Además, cuando vimos que la no existían reglas en común usando las tres familias, se habían usado los mismos valores de `min_support` para todas.

Tomamos todos los ítems con alta frecuencia de las transacciones de las familias comparables y analizamos cuántos se tienen en común. Al igual que en el experimento de la sección 4.2, contamos solo los MRs y no la cantidad de instancias que ellos poseen.

### Resultados y discusión

En la Tabla 4.3 se muestran las intersecciones de MRs de alta frecuencia, con `min_len` fijo en 4, entre las familias comparables. Si analizamos las familias de forma individual, la cantidad de MRs disminuye en dos órdenes de magnitud comparado a no filtrar los frecuentes (por ejemplo, los de `NEWAnk` disminuyen desde aprox. 155.000 hasta 2.130).

Los MRs compartidos entre las tres familias (316 MRs) ya no representan el 97 % de `NEWAnk` como en el experimento anterior (Tabla 4.1), sino que ahora representan el 15 % de `NEWAnk` y el 8 % de `LRR1`.

| NEWAnk | TPR1 | LRR1 | **Cantidad de MRs compartidos entre las familias seleccionadas** |
|---|---|---|---|
| x | | | 2.130 |
| | x | | 2.677 |
| | | x | 3.759 |
| x | x | | 708 |
| x | | x | 485 |
| | x | x | 628 |
| x | x | x | 316 |

*Tab. 4.3:* Cantidad de MRs, con más de 804 instancias, en las transacciones de las familias comparables, usando `min_len` 4 y refinamiento *substring*. En cada fila se marca con una x aquellas familias que se tomaron para calcular la intersección.

El hecho de que la cantidad de ítems de alta frecuencia haya disminuido dos órdenes de magnitud con respecto a considerar todos los presentes nos indica que son realmente pocos (en comparación) los ítems que potencialmente participan en la generación de reglas, dado que solo ítems cuyo soporte supere el `min_support` serán los considerados. Los porcentajes bajos de MRs en las intersecciones nos indican que la mayoría de los MRs relevantes para las reglas son propios de cada familia, lo cual es consistente con el hecho de que las familias no comparten reglas.

Se muestran en la Tabla 4.4 los resultados de tomar intersecciones de MRs de alta frecuencia con `min_len` 5 para las familias comparables. Las cantidades de MRs para `NEWAnk`, `TPR1` y `LRR1` son, respectivamente, 398, 133 y 1.136. En cuanto a las intersecciones aparece un único MR entre `NEWAnk` y `TPR1`, y también solo uno entre `NEWAnk` y `LRR1`. Estos son "AAAAA" y "SSSSS", respectivamente. La intersección es vacía entre `TPR1` y `LRR1`, y por lo tanto también lo es al considerar las tres familias a la vez.

| NEWAnk | TPR1 | LRR1 | **Cantidad de MRs compartidos entre las familias seleccionadas** |
|---|---|---|---|
| x | | | 398 |
| | x | | 133 |
| | | x | 1.136 |
| x | x | | 1 |
| x | | x | 1 |
| | x | x | 0 |
| x | x | x | 0 |

*Tab. 4.4:* Cantidad de MRs, con más de 804 instancias, en las transacciones de las familias comparables, usando `min_len` 5 y refinamiento *substring*. En cada fila se marca con una x aquellas familias que se tomaron para calcular la intersección.

Al haber aumentado el `min_len` de 4 a 5, observamos que el fenómeno se agudiza. Con la excepción de 2 MRs, todos los MRs de alta frecuencia son propios de alguna familia. Esto muestra con claridad por qué las reglas generadas son diferentes: las familias no compartían MRs frecuentes. Podemos notar además cómo las cantidades de los MRs de

alta frecuencia, viéndolos individualmente, se corresponden con las cantidades de reglas: el de menos MRs y menos reglas es `TPR1`, siguiendo con `NEWAnk` y por último `LRR1` (Tabla 3.1).

### Conclusión

Al tomar de las transacciones de nuestras familias comparables los MRs de alta frecuencia, y al considerar largos mínimos cada vez mayores, se pone en evidencia cómo disminuye la intersección entre diferentes familias. Es decir, cada familia tiene un núcleo propio de MRs. Esto explica por qué las reglas que se generan son distintas: no puede haber reglas en común si no hay MRs frecuentes en común. Estos resultados aportan evidencia a lo analizado en Turjanski2016: los MRs ayudan a la caracterización de las familias de proteínas.

Queda para un trabajo futuro analizar qué pasa si tomamos otros conjuntos de familias. ¿Se generarán MRs únicos en todas las familias? ¿Familias muy cercanas biológicamente generarán conjuntos de MRs similares? También podría analizarse qué pasaría si comparamos MRs "parecidos" (es decir, no exactamente iguales) al estudiar las intersecciones entre familias. Podríamos tomar alguna medida de similitud como distancia de edición o de Levenshtein. Podría estar sucediendo que los MRs sean distintos pero que se diferencien en un único aminoácido (por ejemplo, HYAC vs. HYAD) que haga que querramos considerarlos similares. Esto involucraría también pensar en nociones de similitud de reglas y consideramos que exceden el alcance de este trabajo.

# 5. OPTIMIZACIONES EN EL CÓMPUTO DE REGLAS DE ASOCIACIÓN

## 5.1. Distribución de largos de repeticiones maximales

El proceso completo de generación de reglas de asociación incluye la búsqueda de repeticiones maximales, la generación de transacciones utilizando un refinamiento y finalmente la ejecución del algoritmo *Apriori*. Todo este proceso implica un costo computacional muy elevado que, dependiendo de cuáles parámetros se deseen modificar, es necesario repetir reiteradas veces para poder realizar un análisis completo en múltiples familias de proteínas. De aquí surge la importancia de optimizar el proceso para que sea más eficiente.

En esta primera sección nos enfocaremos en la etapa de refinamiento de transacciones, presentada en la Sección 1.4. A lo largo de todo este trabajo estuvimos utilizando transacciones generadas utilizando el modo *substring*, dado que fue la opción elegida en el trabajo de Enríquez. Nos proponemos buscar una alternativa que genere los mismos (o similares) resultados pero que sea computacionalmente más eficiente. Para eso, comenzaremos esta sección analizando la distribución de los largos de los MRs presentes en las transacciones, así podemos entender cómo el refinamiento impacta en los resultados finales.

Recordemos que uno de los parámetros para la generación de transacciones es `min_len`. Este parámetro determina el largo mínimo que debe tener un MR para ser considerado en una transacción. Sin embargo, no alcanza con que un MR satisfaga la condición del largo para que quede en la transacción final, porque este MR podría ser eliminado en el refinamiento. La existencia del refinamiento (en nuestro caso, de modo *substring*) implica que, por ejemplo, un MR de longitud 5 no necesariamente esté presente cuándo el parámetro de longitud mínima sea 4. Veamos un ejemplo de cómo un conjunto de MRs se ve afectado por el refinamiento al generar la transacción final:

$$\{\text{TPLH, TPLHA, ATPLH}\} \xrightarrow{\text{refinamiento substring}} \{\text{TPLH}\}$$

El refinamiento *substring* elimina todos los MRs que tienen a algún otro MR como substring. En el ejemplo, todos los MRs eliminados contenían a TPLH como substring.

### Metodología

Generamos transacciones utilizando el refinamiento en modo *substring* para nuestras familias comparables. Variamos el parámetro `min_len` de 4 a 8 para analizar las distribuciones de los largos de los MRs en las transacciones. Para estos experimentos, contamos solo los MRs y no la cantidad de instancias que ellos poseen.

### Resultados y discusión

Analizamos la cantidad de MRs en transacciones en función de su largo, para la familia `NEWAnk`. En la Tabla 5.1 se encuentran los valores obtenidos fijando `min_len` en 4. Se observa que casi la totalidad de los MRs (158.105) tiene largo 4 mientras que únicamente 2 tienen largo 5. No hay MRs de otros largos.

| Largo | Cantidad |
|---|---|
| 4 | 158.105 |
| 5 | 2 |

*Tab. 5.1:* Cantidad de MRs en función de su largo, en transacciones refinadas con modo *substring*, usando `min_len` 4 en la familia `NEWAnk`

El hecho de que casi todos los MRs sean de largo 4 puede explicarse por la gran cantidad de MRs de largo 4 que existen y por cómo funciona el modo *substring*. Como hay 20 aminoácidos presentes en la familia, la cantidad máxima teórica de MRs de largo 4 es 160.000 ($= 20^4$). Si tenemos 158.105 MRs de largo 4, significa que se generaron casi todos los MRs posibles. Cualquier MR con largo mayor que se hubiese generado, tendría casi con certeza como substring a uno de largo 4 y, por lo tanto, será eliminado por el filtro de refinamiento, provocando que no haya MRs de largos mayores en las transacciones de la familia.

Pueden observarse en las Figuras 5.1 y 5.2 la distribución de las cantidades de MRs en función de su largo usando `min_len` 6 y 8, respectivamente. No mostramos los gráficos de `min_len` 5 y 7 por ser similares a los presentados. En todos los casos puede apreciarse cómo los MRs cuyo largo coincide con el `min_len` elegido superan en cantidad a los MRs de otros largos por al menos una orden de magnitud. Vemos cómo largos mayores cobran un poco más de relevancia en valores de `min_len` a partir de 8, pero en la práctica vimos que tomar valores tan altos no tiene sentido porque las reglas generadas son muy pocas, dado que MRs largos son poco frecuentes en las familias.

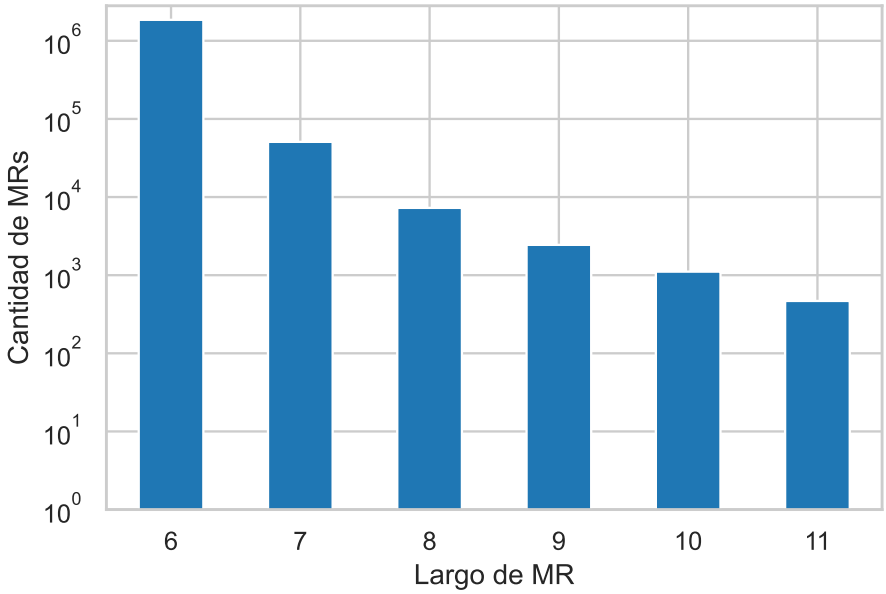



*Fig. 5.1:* Cantidad de MRs en transacciones refinadas con modo *substring*, usando `min_len` 6 en la familia `NEWAnk`. Escala logarítmica en Y.

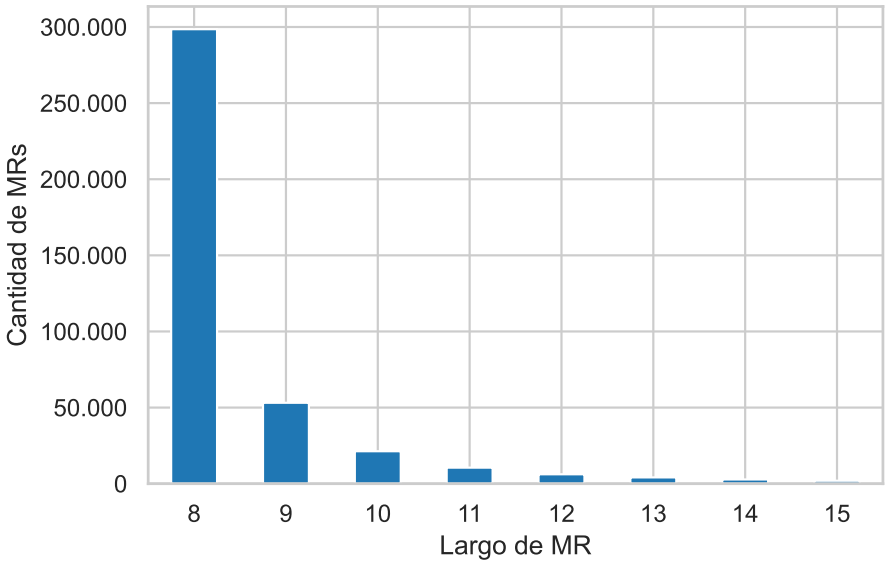



*Fig. 5.2:* Cantidad de MRs en transacciones refinadas con modo *substring*, usando `min_len` 8 en la familia `NEWAnk`.

Repetimos el experimento con las dos familias restantes (`TPR1` y `LRR1`) y obtuvimos resultados equivalentes a los vistos con `NEWAnk` (resultados no mostrados).

La mayoría de los largos de MRs presentes en las transacciones, luego del refinamiento, coinciden con el `min_len` elegido para el experimento. ¿Tiene sentido que en las transacciones consideremos MRs de largos mayores al `min_len`?

## 5.2. Refinamiento mínimo en repeticiones maximales

Como vimos, la gran mayoría de los ítems en las transacciones refinadas con modo *substring* tienen largos que coinciden con el parámetro de `min_len` que se fijó para generarlas. Estos ítems de tamaños mayores ¿aportan algo a las reglas generadas? Si los largos mayores al `min_len` no aportan nada, podríamos ahorrarnos el proceso costoso de analizar substrings en el refinamiento de transacciones, porque bastaría con ver el largo de un MR para decidir si mantenerlo o no.

### Metodología

Generamos reglas de asociación en las familias comparables manteniendo fijos los valores por defecto (Tabla 2.2) pero variando `min_len` entre 4 y 7 inclusive al generar las transacciones. Analizamos cuántos de los ítems que forman las reglas coinciden con el `min_len` elegido.

### Resultados

En la Tabla 5.2 se muestra, para cada combinación de familia y `min_len` posibles, cuál es el porcentaje de ítems presentes en las reglas que coincide con el `min_len` que se utilizó para generar las transacciones asociadas. `TPR1` no generó reglas utilizando `min_len` mayores a 4. Para el resto de las familias y con todos los valores analizados, el 100 % de los ítems de las reglas tienen el mismo largo que el especificado en `min_len`.

| Familia \ min_len | 4 | 5 | 6 | 7 |
|---|---|---|---|---|
| NEWAnk | 100 % | 100 % | 100 % | 100 % |
| LRR1 | 100 % | 100 % | 100 % | 100 % |
| TPR1 | 100 % | - | - | - |

*Tab. 5.2:* Cantidad de MRs presentes en las reglas que coinciden con el `min_len` que las generó. Las combinaciones con **-** no generaron reglas.

### Discusión

Como ningún ítem de tamaño mayor al `min_len` aparece en las reglas, si quitásemos estos ítems de las transacciones no habría cambio en las reglas generadas, porque las frecuencias de los que sí aparecen no se modifican de ninguna manera. De esta forma, el refinamiento de transacciones en modo *substring* puede reemplazarse con tomar los MRs de tamaño mínimo en cada transacción y obtendríamos los mismos resultados.

El refinamiento en modo *substring* es un proceso que puede demorar aprox. unos 30 minutos, dependiendo del tamaño y la cantidad de transacciones. Es posible que pueda implementarse este filtro de substrings de forma más eficiente a la que existe actualmente, quizá utilizando estructuras de datos más complejas (y más apropiadas) para trabajar con substrings. En base a estos resultados, implementamos un nuevo modo de refinamiento de transacciones: el modo *minimum*. En este modo nos quedamos únicamente con los ítems que tienen el mínimo tamaño en cada transacción.

Utilizamos este nuevo modo para generar las reglas y comprobamos que se generan las mismas reglas que antes. El tiempo de refinamiento pasó a ser de unos pocos segundos. Esta reducción de tiempo, de varios órdenes de magnitud, es de gran importancia para las etapas exploratorias, dónde refinamos reiteradas veces al experimentar con diferentes combinaciones de parámetros y familias. Salvo que mencionemos lo contrario, de aquí en más refinaremos las transacciones utilizando el modo *minimum*.

## 5.3. Reducción de repeticiones maximales

En la Sección 4.3 vimos que los MRs con alta frecuencia forman un núcleo en cada familia y determinan cuáles son las reglas que se forman. Hasta ahora venimos utilizando como base para generar las transacciones el conjunto de todos los MRs (ALL), pero no tuvimos en cuenta que podemos clasificar a los MRs en categorías disjuntas (NE, NN y SMR, ver Definición 1.1.2). Nos preguntamos cómo es la distribución de estas categorías de MRs en nuestras transacciones.

### Metodología

Tomamos la familia `NEWAnk` y utilizamos las distintas categorías de MRs (NN, NE y SMR) para generar las transacciones a partir de cada una de ellas (con refinamiento en el modo *minimum*, ver Sección 5.2). Vimos para cada categoría cómo varía la cantidad resultante de ítems generados en las transacciones. Experimentamos variando `min_len` entre 4 y 8 inclusive.

### Resultados

En la Figura 5.3 presentamos la cantidad de MRs para cada una de las categorías, sin tener en cuenta la cantidad de instancias. Podemos ver que la cantidad de MRs tiene un incremento a medida que nos movemos de `min_len` bajos hacia `min_len` medios, y un decremento a medida que nos movemos de los valores de `min_len` medios a los altos. Para la categoría NN, este incremento se da hasta `min_len` 5, mientras que para el resto (ALL, NE y SMR) se produce hasta `min_len` 6.

Observamos que las transacciones generadas con NE tienen pocos MRs distintos comparados a las formadas por el resto de las categorías. Vemos también que en `min_len` 4, las transacciones generadas con MRs de categoría SMR tiene más MRs que aquellas generadas con ALL. Analizaremos este último punto en la subsección Discusión.

Las cantidades crecientes de MRs son consistentes con nuestras observaciones en la Sección 4.2, donde vimos que al elegir valores de `min_len` más grandes conseguimos un espacio de búsqueda mayor y, por lo tanto, se permite la generación de más MRs. En esta figura podemos observar que este fenómeno no es una tendencia que se mantiene, sino que rápidamente (a partir de `min_len` 6) la cantidad de MRs disminuye de forma abrupta. Esto podría estar relacionado a que se da naturalmente que existen pocos MRs con valores altos de `min_len`.

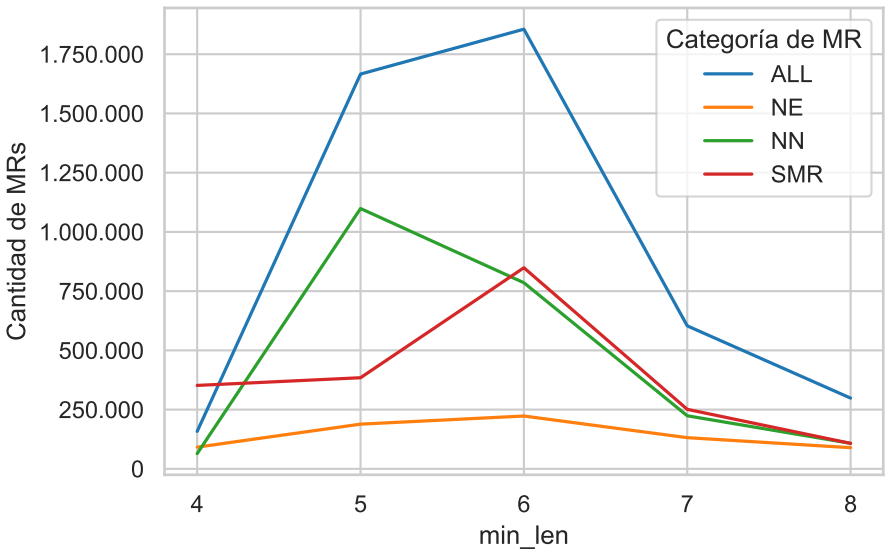



*Fig. 5.3:* Cantidad de MRs en transacciones refinadas con modo *minimum* en función de su `min_len`, para distintas categorías de MRs. No se tiene en cuenta la cantidad de instancias. Familia `NEWAnk`.

Como deseamos analizar cómo se reducirían los tamaños de las transacciones al usar otras categorías de MRs en su generación, repetimos el experimento pero esta vez teniendo en cuenta la cantidad de instancias. Pueden verse estos resultados en la Figura 5.4, donde mostramos la cantidad de instancias de MRs por categoría para la familia `NEWAnk`.

Vemos una reducción general en la cantidad de MRs a medida que aumenta el `min_len` (con la excepción de NN pasando `min_len` de 4 a 5, y en SMR con `min_len` menores a 6) y todas las categorías terminan por debajo de 2.500.000. La cantidad de ítems en SMR se mantiene siempre por debajo de las demás.

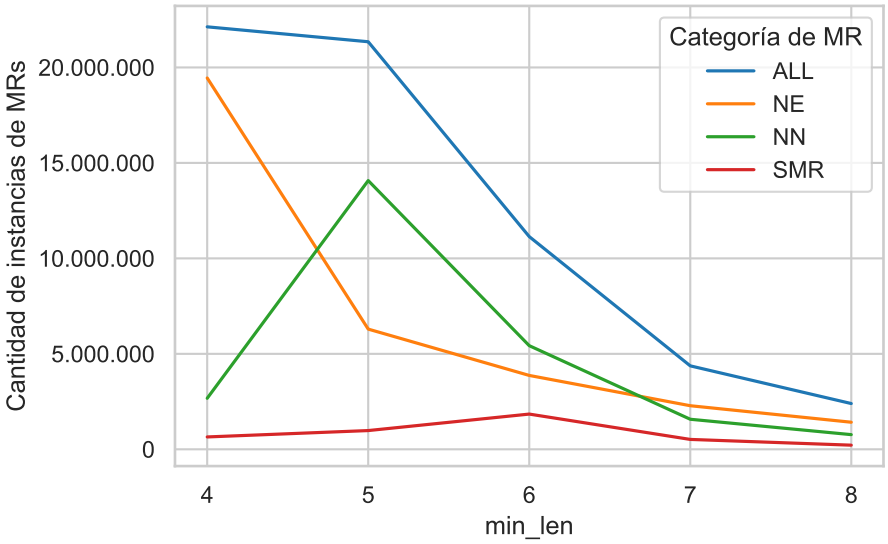



*Fig. 5.4:* Cantidad de instancias de MRs en transacciones refinadas con modo *minimum* en función de su `min_len`, para distintas categorías de MRs. Familia `NEWAnk`.

Existe un pico en la cantidad de instancias para `min_len` 5 al usar la categoría NN. Realizamos estos experimentos con otras familias y también nos encontramos con un pico en ese mismo punto para dicha categoría (resultados no mostrados).

Para entender mejor las posibles reducciones en los tamaños de las transacciones, presentamos en la Figura 5.5 qué proporción de la cantidad de instancias en las transacciones representan las categorías NE, NN y SMR con respecto a ALL, en función de `min_len`. Es decir, tomamos la cantidad de instancias totales en cada categoría y lo dividimos por la cantidad de instancias en ALL.

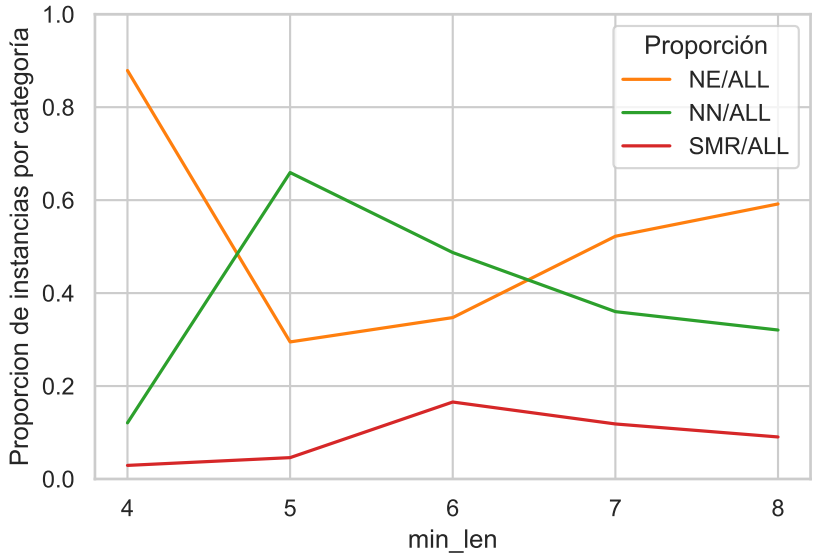



*Fig. 5.5:* Proporciones de la cantidad de instancias en transacciones de diferentes categorías (NE, NN y SMR) con respecto a la cantidad de instancias en transacciones que usan ALL, en función de su `min_len`. Transacciones refinadas con modo *minimum*. Familia `NEWAnk`.

La proporción de cantidades de instancias de MRs en transacciones que usan SMR con respecto a ALL es menor al resto de las categorías en todos los `min_len`, con un pico en 16 % para `min_len` 6 y un posterior descenso. Para el caso de la categoría NN, la proporción es del 12 % al usar `min_len` 4, y al aumentar el parámetro a 5 hace un salto a 65 %, correspondiente con el pico visto en el experimento anterior. Pasado ese punto, para valores de `min_len` a partir de 6, la proporción desciende hasta estabilizarse por debajo del 50 %. Por último, en la categoría NE se produce el efecto opuesto a las demás categorías: comienza con una proporción muy alta (87 %) en `min_len` 4, baja hasta 29 % en `min_len` 5, y luego continúa subiendo hasta llegar a 60 % en `min_len` 8.

### Discusión

Observamos en la Figura 5.3 un crecimiento y decrecimiento de la cantidad de MRs a medida que aumenta el `min_len`. Creemos que al aumentar `min_len` hay más combinaciones posibles de aminoácidos en los MRs, por lo que tiene sentido que haya mayor cantidad de MRs distintos. Sin embargo, esto tiene un límite, ya que sabemos que a medida que el largo aumenta, decrece la cantidad de MRs existentes. Esto también es evidenciado en la Figura 5.4, dónde se aprecia cómo en `min_len` 6 la cantidad de instancias de MRs disminuye fuertemente. Estos resultados sobre cantidades de instancias son esperables. Como para tamaños mayores hay más combinaciones de aminoácidos, es más difícil que haya las suficientes repeticiones para que los patrones se clasifiquen como MRs.

Algo importante a notar es que en las transacciones ya no es cierto que ALL sea la unión del resto de las categorías. Esto puede verse con claridad en la Figura 5.3: con `min_len` 4, SMR tiene más MRs que ALL. La explicación tiene que ver con el refinamiento de transacciones, que en nuestro caso es el modo *minimum*. Supongamos que tenemos los siguientes conjuntos de MRs pre-refinamiento en una transacción:

- `ALL = { SMR01, SMR02, SMR03, MR }`
- `SMR = { SMR01, SMR02, SMR03 }`

Todos los MRs son de tamaño 5 con excepción de `MR` que tiene tamaño 2. A continuación aplicamos refinamiento en modo *minimum*. Recordemos que este refinamiento toma el largo mínimo entre los ítems de la transacción y elimina todos los ítems mayores a este mínimo. De esta manera, post-refinamiento, obtenemos:

- `ALL = { MR }`
- `SMR = { SMR01, SMR02, SMR03 }`

La transacción queda con más ítems en SMR que en ALL. Aunque este tipo de casos pueden no ser usuales, queda pendiente para un trabajo futuro analizar la frecuencia con que se dan estos fenómenos. Vemos en la práctica que la situación se estabiliza pasando `min_len` 4 y ALL se convierte en el conjunto con más elementos que las demás categorías.

Por último, como nos muestra la Figura 5.5, es posible reducir el tamaño de las transacciones si usamos las diferentes categorías de MRs en vez de ALL. Exploraremos en la siguiente sección si hacer esta reducción tiene sentido teniendo en cuenta las reglas que se generan.

## 5.4. Reglas por categorías de repeticiones maximales

Dado que si agrupamos los MRs según su categoría (SMR, NN, NE) obtenemos conjuntos disjuntos, nos preguntamos qué porcentaje de cada uno de ellos participa en la totalidad de MRs que componen las reglas generadas. Más específicamente, ¿los MRs que conforman las reglas generadas, pertenecen a una única categoría de MR? Por ejemplo, ¿son todos NE?

### Metodología

Generamos reglas de asociación para las familias comparables, usando como entrada en cada experimento transacciones formadas por cada una de las distintas categorías de MRs (ALL, NE, NN, SMR) y refinadas en modo *minimum*, variamos `min_len` entre 4 y 8, variando `min_support` entre 0.015 y 0.030 con saltos de 0.005, y dejamos fijo `min_confidence` en 0.90.

### Resultados y discusión

Presentamos en la Tabla 5.3 la cantidad de reglas generadas para `NEWAnk`, partiendo de transacciones formadas por las diferentes categorías de MRs (ALL, NE, NN, SMR), con `min_support` fijo en 0.025 y variando `min_len` entre 4 y 8. Para `min_len` 4, ya sabíamos por experimentos anteriores que ALL generaba 4.888 reglas; observamos ahora que NE genera casi la totalidad de ellas: 4.887 reglas. Para tamaños de `min_len` mayores se observa un fenómeno similar. En los experimentos donde utilizamos MRs únicamente de las categorías NN y SMR, el método no generó ninguna regla.

| Categoría / min_len | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 4 | 4.888 | 4.887 | 0 | 0 |
| 5 | 174 | 172 | 0 | 0 |
| 6 | 13 | 11 | 0 | 0 |
| 7 | 2 | 2 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |

*Tab. 5.3:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_len` entre 4 y 8. Familia `NEWAnk`. `min_support` 0.025. `min_confidence` 0.90

Nos resultó sorprendente que la única categoría que genera reglas es NE, y más sorprendente que genere casi todas las posibles. Habíamos esperado alguna distribución más equilibrada entre las diferentes categorías.

La causa de esta diferencia en la cantidad de reglas entre NE y el resto de las categorías no se basa su explicación en que el conjunto de instancias de MRs en la categoría NE tiene un tamaño mayor al resto (ver tamaños de los conjuntos en Figura 5.4), ya que para `min_len` 5 y 6, el conjunto de MRs de tipo NN es mayor y sin embargo NE sigue concentrando casi la totalidad de las reglas.

Una posible explicación que surge viendo la Figura 5.3, es que NE tiene menos MR distintos, por lo que cada uno de ellos aparecería con mayor frecuencia que MRs en otras categorías. Como el valor de frecuencia de cada MR es muy importante a la hora de

generar reglas, tiene sentido que la categoría que tiene más MRs que superen el umbral de frecuencia sea la que genere las reglas. Como los conjuntos de cada categoría de MRs son disjuntos, que no se hayan generado reglas con NN y SMR significa que prácticamente no aportan MRs cuyas frecuencias superan el `min_support`. Solo podrían aportar si tuviesen algún MR en común, pero por definición no lo tienen.

No se generan reglas con la categoría NN porque las bajas frecuencias de sus MRs, pero no encontramos una razón teórica detrás de esto. Sin embargo, es interesante notar que existe una cota superior teórica para las instancias de la categoría SMR, que es consistente con lo visto en la Figura 5.4, donde puede verse que esta categoría tiene menos instancias que las demás para todos los valores de `min_len`. Esta cota teórica en la cantidad de instancias explica por qué ningún SMR supera la barrera del `min_support` y por lo tanto no se generan reglas con ellos. Tomamos la explicación del trabajo de Taillefer y Miller [16].

Aplicado a nuestro caso, como nuestro alfabeto de aminoácidos tiene 20 caracteres posibles, aseguramos que cada SMR no puede tener más de 20 instancias. Si un MR es un SMR, significa que éste es una repetición maximal donde todas sus extensiones ocurren exactamente una vez. Como solo hay 20 símbolos posibles para extender la repetición a izquierda o derecha, si la repetición tuviese más de 20 ocurrencias, necesariamente al menos dos de ellas son extensibles, y por lo tanto no serían SMRs.

Habiendo encontrado una posible explicación de por qué los MRs de la categoría NE son los que generan casi la totalidad de las reglas, queda la pregunta de por qué no genera la totalidad de ellas. Tomemos como ejemplo el caso de la siguiente regla, generada con ALL y faltante en NE, con `min_len` 4 (ver Tabla 5.3):

$$\{\text{HYAC}\} \rightarrow \{\text{LHYA}\}$$

Al buscar estos ítems en nuestra base de MRs, descubrimos que el consecuente {LHYA} es un MR de la categoría NE mientras que el antecedente {HYAC} es un MR de la categoría NN. Como los conjuntos de MRs son disjuntos, no es posible que usando sólo uno de ellos generemos esta regla. En consecuencia, si decidimos utilizar MRs de las categorías NE, NN o SMR en vez de utilizar ALL, es posible que estemos perdiendo alguna regla. Sin embargo, la cantidad de reglas perdidas en el caso de MRs de tipo NE no parece ser grande. Para el caso de los restantes experimentos (`min_len` 5 y 6, ver Tabla 5.3) las reglas que no han sido generadas cuando se utilizó la categoría NE son presentadas en la Tabla 5.4. Todas ellas no aparecen al usar la categoría NE por la misma razón: tienen una componente NE y otra componente NN.

| `min_len` | **Regla** |
|---|---|
| 4 | {HYAC} → {<u>LHYA</u>} |
| 5 | {<u>TALHM</u>} → {ALHMA} |
| 5 | {<u>LKNGA</u>} → {LLKNG} |
| 6 | {LEKGAD} → {<u>LLEKGA</u>} |
| 6 | {RTALHL} → {<u>TALHLA</u>} |

*Tab. 5.4:* Reglas generadas por transacciones que utilizan MRs de categoría ALL que se pierden al utilizar la categoría NE, para valores de `min_len` entre 4 y 6, correspondientes a la Tabla 5.3. Los MRs subrayados pertenecen a la categoría NE.

Repetimos el mismo experimento, pero esta vez fijando otros valores de `min_support`, algunos menores y otros mayores. Mostramos en la Tabla 5.5 los resultados utilizando `min_support` 0.020 y en la Tabla 5.6 utilizando `min_support` 0.030. En todos los casos notamos el mismo comportamiento observado que con `min_support` 0.025. Es decir, con transacciones que utilizan MRs de la categoría NE se generan casi la totalidad de las reglas, mientras que con el resto de las categorías no se genera ninguna.

| Categoría / min_len | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 4 | 10.111 | 10.107 | 0 | 0 |
| 5 | 311 | 308 | 0 | 0 |
| 6 | 23 | 21 | 0 | 0 |
| 7 | 3 | 3 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |

*Tab. 5.5:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_len` entre 4 y 8. Familia `NEWAnk`. `min_support` 0.020. `min_confidence` 0.90

| Categoría / min_len | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 4 | 2.718 | 2.718 | 0 | 0 |
| 5 | 112 | 112 | 0 | 0 |
| 6 | 8 | 8 | 0 | 0 |
| 7 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |

*Tab. 5.6:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_len` entre 4 y 8. Familia `NEWAnk`. `min_support` 0.030. `min_confidence` 0.90

Soportes menores a 0.025 (Tabla 5.5) siguieron sin generar reglas para MRs de las categorías NN y SMR. El `min_support` más bajo que probamos fue 0.015 (resultado no mostrado). Sospechamos que existe un valor mucho menor para el cual las reglas podrían empezar a aparecer, pero si el soporte es demasiado bajo, además de que las potenciales reglas no aportarían información que se pueda generalizar, existe el problema adicional de que el algoritmo se vuelve impracticable ejecutar (devolvería demasiadas reglas y/o el tiempo de ejecución sería muy grande).

Resultados obtenidos con un valor mayor en el `min_support` (Tabla 5.6, donde fijamos `min_support` en 0.030) muestran que se parecen descartar todos los ítems que quizá antes estaban cercanos al umbral y termina dándonos como resultado que al usar solo los NE obtenemos exactamente las mismas reglas que juntando todas las categorías (ALL). Es decir, valores mayores en el `min_support` terminan siendo más restrictivos.

Concluimos que, para el caso de `NEWAnk`, tomar el conjunto de NE alcanza para generar prácticamente la totalidad de las reglas. Nos preguntamos si podremos concluir lo mismo

para el resto de las familias comparables. Para responder esto, generamos las reglas de la misma forma que en el experimento anterior pero con la familia `TPR1` (Tabla 5.7) y la familia `LRR1` (Tabla 5.8). Para que sea más notorio el hecho de que el valor de `min_support` tiene una influencia menor en los resultados, variamos dicho valor entre 0.015 y 0.030, con incrementos de 0.005, y dejamos fijo `min_len` en 4.

| Categoría / min_support | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 0,015 | 112 | 107 | 0 | 0 |
| 0,020 | 22 | 22 | 0 | 0 |
| 0,025 | 4 | 4 | 0 | 0 |
| 0,030 | 0 | 0 | 0 | 0 |

*Tab. 5.7:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_support` entre 0.015 y 0.030, con incrementos de 0.005. Familia `TPR1`. `min_len` 4. `min_confidence` 0.90

| Categoría / min_support | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 0,015 | 2.279.705 | 2.198.265 | 4 | 0 |
| 0,020 | 3.776.858 | 3.697.938 | 3 | 0 |
| 0,025 | 1.657.444 | 1.634.553 | 2 | 0 |
| 0,030 | 1.445.207 | 1.433.774 | 2 | 0 |

*Tab. 5.8:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_support` entre 0.015 y 0.030, con incrementos de 0.005. Familia `LRR1`. `min_len` 4. `min_confidence` 0.90

Observamos con ambas familias fenómenos similares a los vistos con `NEWAnk`. Podemos apreciar una diferencia en `LRR1`, donde es la primera vez que se generan reglas con la categoría NN. Sospechamos que esto está fuertemente relacionado con la gran cantidad de MRs que se generaron en esta familia; no nos sorprende que existan algunos NN muy frecuentes que generen reglas.

En la Tabla 5.9 presentamos qué sucede si subimos el `min_len` a 6 para la familia `LRR1`. Sigue siendo cierto que para el caso donde se utilizan MRs de la categoría NE, éstos son los únicos que generan una cantidad importante de reglas, pero ya no es cierto que genera casi todas (aunque sigue siendo una parte mayoritaria). Por ejemplo, para el caso de `min_support`=0.015, entre utilizar MRs de categoría NE en vez de ALL, hay una diferencia de 487 reglas (1.639 reglas utilizando ALL y 1.152 reglas utilizando NE). La razón es la misma que aquellas en la Tabla 5.4: los MRs que se encuentran en las reglas perdidas no son en su totalidad de la categoría NE. Basta con que haya un único MR de otra categoría para que la regla no pueda generarse, dado que las categorías son disjuntas.

| Categoría / min_support | ALL | NE | NN | SMR |
|---|---|---|---|---|
| 0,015 | 1.639 | 1.152 | 1 | 0 |
| 0,020 | 871 | 506 | 1 | 0 |
| 0,025 | 290 | 174 | 1 | 0 |
| 0,030 | 65 | 49 | 0 | 0 |

*Tab. 5.9:* Cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de cada categoría (ALL, NE, NN, SMR). Variamos `min_support` entre 0.015 y 0.030, con incrementos de 0.005. Familia `LRR1`. `min_len` 6. `min_confidence` 0.90

Aunque la cantidad de reglas al usar MRs de las categorías ALL y NE bajaron al aumentar `min_len` de 4 a 6, no lo hicieron en la misma proporción. Vemos en la Tabla 5.10 cómo varía la proporción de reglas para la familia `LRR1` a medida que aumenta `min_len`.

| min_len | NE | ALL | NE / ALL |
|---|---|---|---|
| 4 | 1.634.553 | 1.657.444 | 0,986 |
| 5 | 3.090 | 3.704 | 0,834 |
| 6 | 174 | 290 | 0,600 |

*Tab. 5.10:* Proporción de la cantidad de reglas generadas para transacciones refinadas en modo *minimum*, formadas por MRs de las categorías NE y ALL, para valores de `min_len` entre 4 y 6. Familia `LRR1`. `min_support` 0.025.

Este cambio en la proporción no es observable en las demás familias (`NEWAnk` y `TPR1`), sospechamos que por la baja cantidad de reglas que se generan en comparación a `LRR1`. Las reglas generadas utilizando MRs de la categoría ALL que no se generan utilizando MRs de la categoría NE son, al igual que en el ejemplo inicial, reglas que tienen algún MR de la categoría NN (ya sea en el antecedente o en el consecuente). Nos preguntamos por qué con valores mayores de `min_len` aparecen más de estas reglas "combinadas". Creemos que tiene que ver con un fenómeno estudiado en el trabajo de Turjanski2018. Resulta que si tenemos en cuenta MRs de largos mayores a 4, la cantidad de los MRs que son NE disminuye a una velocidad mucho mayor que los de la categoría NN, provocando que al juntar los conjuntos de MRs (ALL) veamos pares combinados con frecuencias altas. De todas maneras, queda como trabajo futuro profundizar los estudios en esta línea.

### Conclusión

Las transacciones formadas por MRs de las categorías NN y SMR no generan una cantidad considerable de reglas por sí solas en ninguna de las tres familias. La única categoría que vale la pena considerar como posible para reemplazar al conjunto ALL es NE.

En `NEWAnk`, al usar transacciones formadas con MRs de la categoría NE, con `min_len` 4, se mantienen presentes casi la totalidad las reglas si lo comparamos con usar el conjunto de todos los MRs (ALL). Si observamos en la Figura 5.5 la proporción de MRs de la categoría NE con respecto a ALL, podemos ver que para `min_len` 4, NE representa más del 95 % de los MRs, por lo que utilizar una categoría diferente a ALL no presenta ninguna ventaja

con este `min_len`.

Sin embargo, en esa misma Figura se aprecia una reducción de tamaño considerable (de más del 50 %) al utilizar la categoría NE si tomamos `min_len` más altos, por ejemplo `min_len` 6. Con ese `min_len`, en `NEWAnk` se mantienen exactamente las mismas reglas si reemplazamos el conjunto ALL por NE. Por lo tanto, para estos casos, no utilizar ALL tiene sentido si se busca una reducción en el tamaño de las transacciones.

Si también tomamos `min_len` 6 pero en cambio analizamos `LRR1`, la cantidad de reglas que se mantienen al cambiar ALL por NE deja de ser el 100 % como en `NEWAnk` y pasa a ser del 60 % (Tabla 5.10). Esto nos muestra que no podemos decidir de forma general si conviene usar una categoría específica en vez de tomar todos los MRs, porque la cantidad de reglas perdidas (al menos en los casos estudiados) es muy variable y se debe decidir si aceptar o no la pérdida en cada caso particular. Queda como trabajo futuro estudiar por qué en una familia ocurre esto (`LRR1`) y en otra no (`NEWAnk`).

Intentaremos formular en un capítulo posterior (Capítulo 6) una idea de relevancia de las reglas, que podría ayudarnos a entender si las reglas perdidas pueden considerarse relevantes. Como trabajo futuro queda la pregunta de si es posible reducir aún más el núcleo de MRs que genera la mayoría de las reglas. Este núcleo tendrá fuerte relación con los MRs frecuentes, pero podría ser complicado determinarlos de forma genérica.

## 5.5. Una alternativa a las repeticiones maximales

Incluso si tenemos en cuenta las optimizaciones de las secciones anteriores, la generación de reglas de asociación a partir de secuencias de proteínas sigue siendo un proceso costoso computacionalmente. Gran parte del costo está dado por el primer paso del proceso: el cálculo de las repeticiones maximales. Es por esto que nos preguntamos si podremos generar reglas de asociación sin computar los MRs.

Uno de los criterios por el cual seleccionamos a nuestras familias comparables fue la similitud en la longitud de sus secuencias proteicas. Creemos que existe una relación entre el largo de una proteína y el largo de la transacción asociada, pero la relación no es uno a uno. Es decir, si por ejemplo tenemos una proteína formada por 1.000 aminoácidos, no significa que su transacción asociada tenga también 1.000 ítems.

Por lo visto en experimentos anteriores, sabemos que al generar transacciones con un cierto `min_len` de MRs, resulta que luego del refinamiento la gran mayoría de los MRs que quedan en las transacciones tienen un largo igual al `min_len` especificado (Sección 5.1). Por lo tanto, es esperable que la cantidad de MRs en una transacción (es decir, el largo de una transacción) sea menor que la cantidad de caracteres en la secuencia asociada.

Observamos en la Figura 5.6 la distribución de los largos de las transacciones generadas con MRs de `min_len` 4 en nuestras familias comparables. Las medianas de los largos son 637 ítems por transacción para `LRR1`, 557 ítems para `NEWAnk` y 566 ítems para `TPR1`. Los percentiles 25 y 75 son similares en las tres familias, con largos aprox. de 350 y 900, respectivamente.

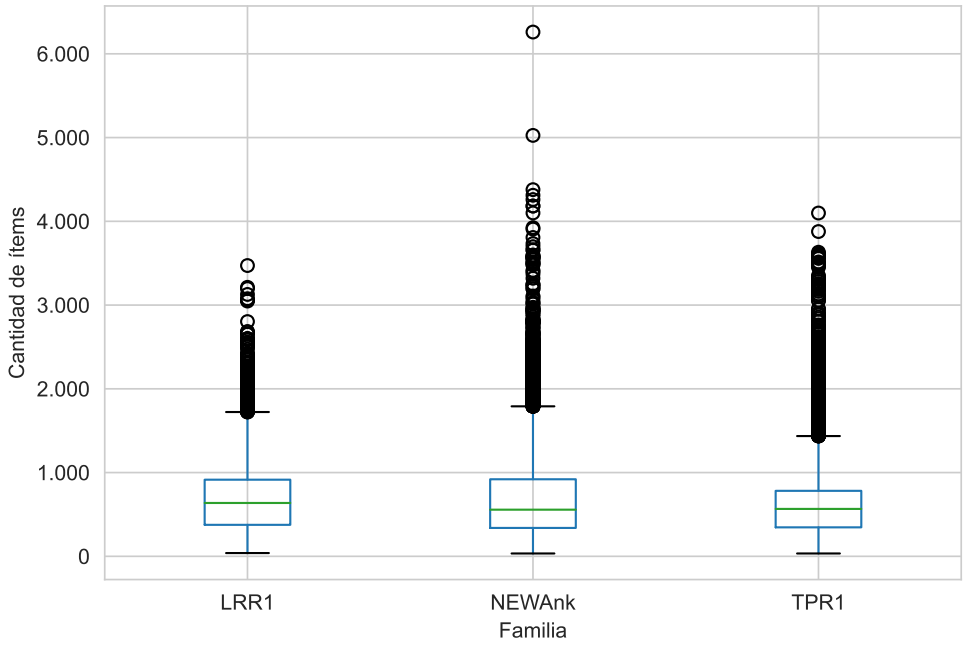



*Fig. 5.6:* Distribución de los largos (cantidad de ítems) de transacciones de MRs, refinadas en modo *minimum*, para las familias `LRR1`, `NEWAnk` y `TPR1`. `min_len`=4.

Era esperable que los largos de estas transacciones sean similares entre sí, ya que en estas familias las cantidades de ítems están relacionadas a los largos de las secuencias de proteínas y esta similitud fue parte del criterio para la elección de las familias comparables.

Aunque estos valores están relacionados, intuitivamente pensamos que una transacción debería contener menos elementos que una secuencia de proteínas, porque si post-refinamiento casi todos los MRs restantes tienen el mismo tamaño, no esperamos que casi todas las subsecuencias de ese tamaño sean MRs. Sin embargo, como podemos ver en la Tabla 5.11, los valores son demasiado similares. Las transacciones parecen tener casi los mismos largos que las secuencias. Esto significa que hay aproximadamente tantos ítems en las transacciones como k-meros de longitud `min_len`, que a su vez es aprox. la cantidad de aminoácidos en las secuencias de proteínas. Dicho de otro modo, podríamos pensar que una transacción está formada por tantos substrings distintos como caracteres hay en las secuencias.

| Familia | Mediana proteínas | Mediana transacciones |
|---|---|---|
| `LRR1` | 669 | 637 |
| `NEWAnk` | 576 | 557 |
| `TPR1` | 585 | 566 |

*Tab. 5.11:* Medianas de largos de secuencias de proteínas y transacciones de MRs refinadas con modo *minimum*, para las familias comparables, con `min_len` fijo en 4.

En base a estos resultados nos preguntamos, ¿obtendremos las mismas reglas si en vez de formar las transacciones usando MRs de `min_len`=`k` armásemos las transacciones usando todos los substrings de tamaño `k`? Esto equivale, en términos biológicos, a tomar

todos los k-meros de la secuencia proteica, con `k` fijo.

**Definición 5.5.1** (Transacción KMERA)**.** Llamamos KMERA a una transacción generada al fijar un valor `k` y utilizar todos los `k`-meros distintos de la secuencia sin realizar el cómputo de los MRs.

Por ejemplo, veamos cómo son los pasos para la generación de una transacción KMERA con `k`=4 a partir de una cierta secuencia inicial.

1. Secuencia: ABCDEABCDE
2. 4-meros: {ABCD, BCDE, CDEA, DEAB, EABC, ABCD, BCDE}
3. Sin repetidos: {ABCD, BCDE, CDEA, DEAB, EABC}

**Definición 5.5.2** (Transacciones KMERAS)**.** Llamamos "transacciones KMERAS", o simplemente "KMERAS", al conjunto de transacciones que se obtienen al aplicar este proceso a cada una de las secuencias de una familia de proteínas. Es decir, a la unión de toda transacción KMERA de una cierta familia.

Es claro que para distintos valores de `k` obtendremos distintas transacciones, de la misma forma que con las transacciones de MRs se obtienen transacciones distintas al variar el parámetro `min_len`.

Las transacciones de MRs no están contenidas en las transacciones KMERAS. En las transacciones de MRs se especifica un tamaño mínimo de MR (`min_len`) por lo que existirían ítems de tamaños mayores a ese `min_len` que forman parte de la transacción. Si generamos transacciones KMERAS fijando `k`=`min_len`, por definición no encontraremos en estas transacciones MRs de largos mayores. De esta forma, pueden existir ítems en las transacciones de MRs que no estén presentes en las KMERAS. Sin embargo, como la mayoría de los MRs no superan el `min_len`, casi todos ellos sí estarán presentes en las KMERAS, por lo que nuestra hipótesis es que prácticamente no perderemos información al utilizarlas.

En la práctica, la contención en el sentido inverso tampoco sucede. Es decir, en la práctica las KMERAS no están totalmente contenidas en las transacciones de MRs. Aunque teóricamente podría ser posible, para que sea cierto absolutamente todos los ítems presentes en las KMERAS deberían ser MRs, pero vimos experimentalmente que esto no sucede con los valores de `k` que utilizamos.

De todos modos, consideraremos que si fijamos el mismo valor de `min_len` en las transacciones MRs y de `k` en las KMERAS, estos dos tipos de transacciones son comparables por lo expuesto en la Sección 5.1, donde vimos que en las transacciones de MRs existen muy pocos ítems con largos que superen el `min_len`.

Nos proponemos analizar cómo cambian las reglas de asociación generadas si utilizamos este nuevo tipo de transacciones en vez de utilizar aquellas generadas en base a MRs.

### Metodología

Generamos reglas de asociación utilizando transacciones de MRs de todas las categorías (ALL) refinadas en modo *minimum* y transacciones KMERAS con las familias comparables. Por lo explicado anteriormente, las comparamos equiparando los valores de `min_len` en las transacciones de MRs y de `k` en las KMERAS, y variamos estos valores entre 4 y 6. Fijamos `min_support` en 0.025 y `min_confidence` en 0.90.

Para la familia `NEWAnk`, realizamos un análisis más exhaustivo, variando el valor de `min_support` entre 0.010 y 0.030, con incrementos de 0.005.

### Resultados y discusión

Para nuestro primer experimento realizamos la comparación de reglas generadas a partir de transacciones de MRs y KMERAS, en las tres familias (`NEWAnk`, `TPR1` y `LRR1`), utilizando todas las combinaciones de parámetros mencionadas en la subsección Metodología. El resultado fue que en todos los casos obtuvimos las mismas reglas.

Consideramos razonable el hecho de que las KMERAS generen al menos las misma cantidad de reglas que las transacciones de MRs, porque en la mayoría de los MRs el `min_len` y `k` coinciden, y todos los MRs con ese mismo largo están presentes en las KMERAS. En otras palabras, nos parece razonable que con las KMERAS no se pierda información.

Nos llama la atención que las KMERAS no generen reglas adicionales, ya que pensamos que para un `k` fijo, existen más k-meros que MRs de ese largo, y esto podría dar lugar a nuevos ítems frecuentes. Más adelante nos encargaremos de analizar el por qué no se generan más reglas utilizando KMERAS que utilizando MRs.

Para intentar entender mejor la diferencia entre los dos tipos de transacciones, analizamos cómo varía la cantidad de ítems (en promedio) que poseen las transacciones de la familia `NEWAnk` en función del largo del ítem utilizado (Figura 5.7). Por simplicidad, cuando hablemos de largo de los ítems nos referiremos al `min_len` para las transacciones de MRs, y al `k` para las KMERAS.

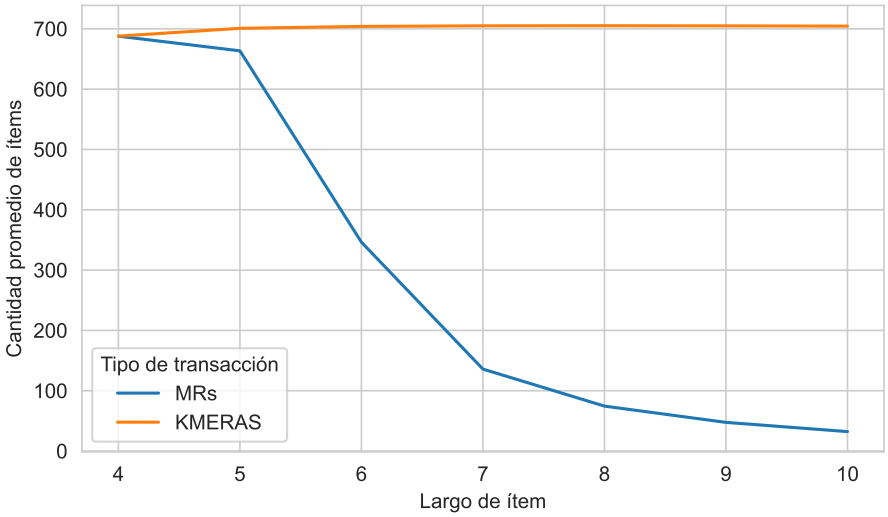



*Fig. 5.7:* Cantidad de ítems promedio en transacciones de MRs refinadas con modo *minimum* y en transacciones KMERAS, variando el largo de los ítems que las generan. El largo del ítem se refiere a `min_len` en transacciones de MRs, y a `k` en KMERAS. Familia `NEWAnk`.

Cuando el largo de los ítems es 4, el tamaño promedio de ambas transacciones coincide en 687 ítems. Esto es esperable si tenemos en cuenta la Tabla 5.11, donde vemos que al usar `min_len` 4, la cantidad de MRs prácticamente coincide con la cantidad de k-meros que poseen las secuencias. Cuando el largo de los ítems es 5 los tamaños también son similares, con 663 ítems en transacciones de MRs y 700 ítems en las KMERAS. Como sigue sin haber tantos MRs distintos posibles, siguen apareciendo la mayoría de los k-

meros. A partir del largo 6, los tamaños promedio de ambos tipos de transacciones dejan de ser similares, y las transacciones de MRs tienen en promedio aprox. la mitad de los elementos que las KMERAS (con 346 ítems en transacciones de MRs y 703 ítems en las KMERAS). En largos superiores, la diferencia se hace cada vez más grande, hasta llegar a largo 10, donde en promedio las transacciones de MRs tienen 32 ítems y las KMERAS, 704 ítems. Es decir, las transacciones KMERAS tienen aprox. 20 veces más ítems que las transacciones de MRs en ese largo.

El tamaño promedio de las transacciones KMERAS se mantiene casi constante en aprox. 700 ítems, con un crecimiento muy leve a medida que aumentamos el valor de `k`. Si a la secuencia de aminoácidos que compone a la proteína la asumimos como aleatoria, entonces es razonable pensar a la mayoría de sus k-meros como elementos distintos. Para ejemplificar, si pensamos en un caso extremo y tuviésemos una proteína formada por el mismo aminoácido repetido todas las veces, todos sus k-meros serían iguales y por lo tanto al eliminar repetidos la transacción KMERA tendría un único elemento. En nuestro caso esto no sucede. Nuestras proteínas se comportan más parecido a como si fueran secuencias aleatorias de aminoácidos

El pequeño incremento que se observa al pasar de largo 4 a 5 se explica porque al tomar ítems de largo mayor, son más los k-meros posibles que pueden aparecer en la proteína, y por lo tanto disminuye la probabilidad de encontrarse con dos k-meros repetidos. Otro punto interesante se observa con las transacciones de MRs, donde a medida que aumentamos el largo de los ítems, los tamaños de transacciones se reducen abruptamente, hasta tener un tamaño 20 veces menor que las KMERAS. También influye que al aumentar el valor de `min_len`, es más difícil que en el caso de los MRs estas secuencias posean más de una instancia.

Es esperable que al bajar la cantidad promedio de ítems en las transacciones, disminuya también la cantidad de ítems totales (pensándolo en cantidad de instancias totales). Podemos ver este resultado en la Figura 5.8, donde la forma de las curvas que se observan son similares a las vistas en la Figura 5.7. La diferencia se encuentra en la magnitud de la cantidades (eje Y), al analizar en esta ocasión la cantidad de ítems totales. Para largos de ítems 4 y 5 las cantidades totales de ítems en ambas transacciones son muy similares, y a partir de largo 6 las diferencias son siempre crecientes.

Lo más interesante de ver y que quizá no es tan obvio, es qué sucede con la cantidad de ítems distintos. Es decir, queremos ver qué pasa con los ítems de las transacciones una vez que eliminamos todas las repeticiones. Podemos observar el resultado en la Figura 5.9.

Otra vez, para largos de ítems 4 y 5, las cantidades de ítems en ambas transacciones es muy similar, y se produce un salto abrupto que las diferencia a partir del largo 6. Con transacciones KMERAS la cantidad de ítems distintos no deja de crecer, lo cual tiene sentido porque al tomar k-meros cada vez mayores, hay mayor lugar a que se formen combinaciones distintas. En el caso de usar MRs, al tomar largos cada vez mayores, la cantidad total de MRs disminuye (pues las secuencias que potencialmente pueden conformar un MR, a mayor tamaño tienen menor probabilidad de repetirse) y por el mismo motivo la cantidad de MRs únicos también disminuye, manteniéndose por debajo del millón considerando MRs a partir de largo 7.

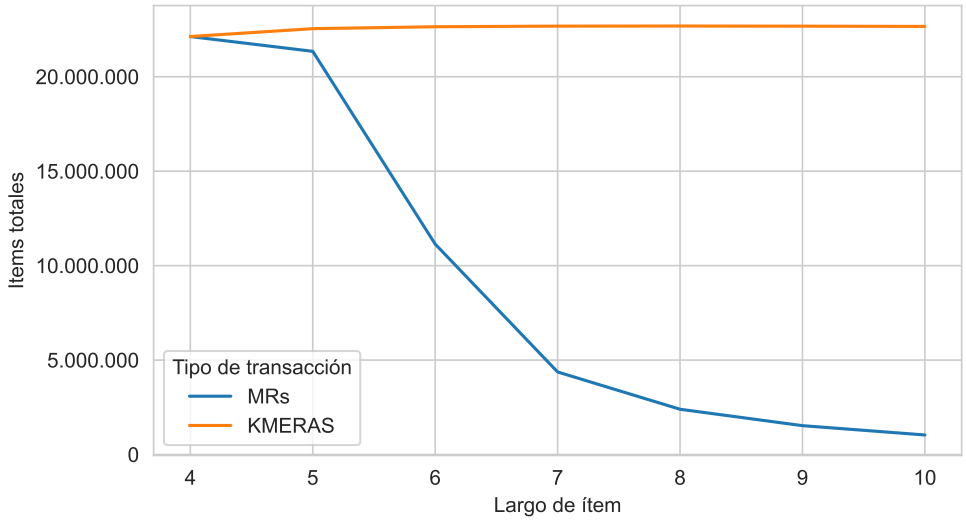



*Fig. 5.8:* Cantidad de ítems totales (instancias) en transacciones de MRs refinadas con modo *minimum* y en transacciones KMERAS, variando el largo de los ítems que las generan. El largo del ítem se refiere a `min_len` en transacciones de MRs, y a `k` en KMERAS. Familia `NEWAnk`.

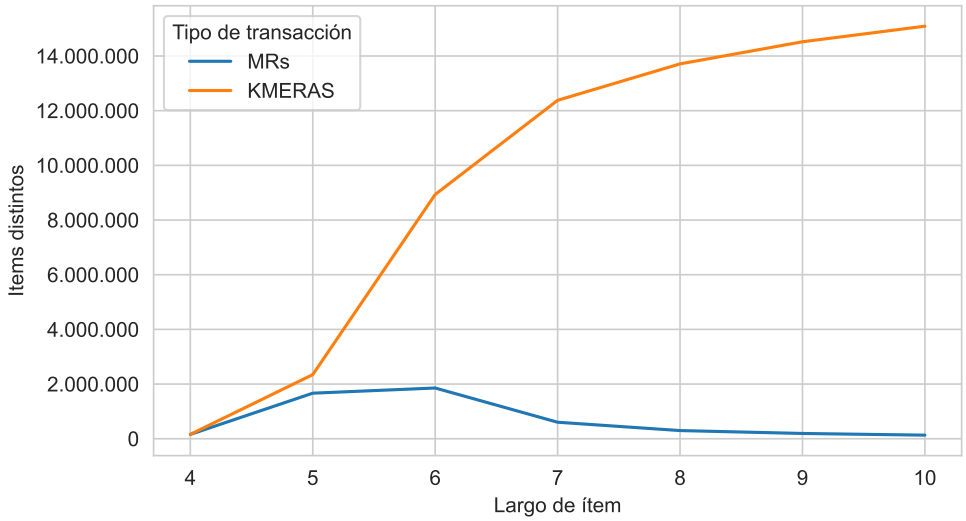



*Fig. 5.9:* Cantidad de ítems distintos en transacciones de MRs refinadas con modo *minimum* y en transacciones KMERAS, variando el largo de los ítems que las generan. El largo del ítem se refiere a `min_len` en transacciones de MRs, y a `k` en KMERAS. Familia `NEWAnk`.

Finalmente, nos queda pendiente intentar encontrar una explicación al hecho de que las transacciones KMERAS generan exactamente las mismas reglas que los MRs (y no más). Acabamos de ver que la cantidad de ítems en las KMERAS es hasta 10 veces mayor para largos altos, por lo que tendría sentido esperar que con esta cantidad de ítems se generen nuevas reglas.

Pero, ¿por qué no se generan más reglas que antes? Pensemos cuál es una de las condiciones necesaria (aunque no suficiente) para que una regla se genere. Como vimos

en la Sección 3.3, las frecuencias de los pares de ítems son importantes para entender la generación de reglas. Lo que sospechamos es los ítems frecuentes que determinan la aparición de una regla en las transacciones KMERAS son siempre MRs. Para intentar determinarlo realizamos el experimento descripto en la Sección 3.3, donde se comparan las distribuciones de las frecuencias de pares de ítems (recordemos que un par de ítems es la mínima cantidad de elementos necesarios para generar una regla). Variamos el largo de los ítems (`min_len` y `k`) entre 4 y 8. Como se explica en la subsección Metodología de la sección mencionada, los pares analizados están formados por ítems individuales que superan el umbral del `min_support`.

Resultó que no hay ninguna diferencia en las frecuencias de pares de ítems en ambos tipos de transacciones. Esto nos indica que aunque las transacciones KMERAS tienen muchos más ítems que las transacciones de MRs, ninguno de estos ítems adicionales supera al `min_support` y por lo tanto son descartados en el primer paso del algoritmo *Apriori*. Luego, al quedarnos con los ítems frecuentes estamos obteniendo exactamente los mismos pares de ítems en ambas transacciones y esto explica por qué obtenemos las mismas reglas en los dos casos.

### Conclusión

Al considerar las transacciones generadas por ambos métodos y los ítems de largos 4 y 5, prácticamente no hay diferencia en las cantidades de ítems presentes en ellas. Sin embargo, la generación que utiliza MRs, aunque usamos un algoritmo eficiente, en la práctica sigue siendo computacionalmente costosa. Por el contrario, obtener todos los k-meros para un cierto `k` demora unos pocos segundos, pues podemos obtener los substrings de una secuencia mientras la iteramos. Para eliminar repetidos, basta con colocarlos en un *set*, llevando el costo computacional a $O(\mathrm{n} * \log \mathrm{n})$, siendo n el tamaño de la secuencia. Por estos motivos, usar transacciones KMERAS puede ser conveniente para largos pequeños (`k`=4 y 5).

Para largos de ítems mayores, de 6 o más, si bien el tiempo de procesamiento puede resultar un problema, los tamaños de las transacciones de MRs terminan siendo mucho menores que las KMERAS (Figura 5.8). La primera consecuencia de esto es que los tamaños de los archivos de transacciones KMERAS terminan siendo mucho mayores comparado con los de MRs. Por ejemplo, para la familia `NEWAnk` con ítems de largo 8, las transacciones KMERAS ocupan 204MB mientras que las transacciones de MRs, 21MB, es decir casi 10 veces menos. Esto resulta importante al trabajar con una gran cantidad de combinaciones de familias, longitudes y categorías de MR, como lo fue en el desarrollo de este trabajo. Debemos notar que si el tamaño que ocupan las transacciones es un factor limitante, vimos que es posible reducir el conjunto de MRs para utilizar únicamente una categoría (NE), aunque se presenta la desventaja de la pérdida de reglas. Esta pérdida podría o no ser aceptable, dependiendo la familia y los parámetros utilizados.

Otra consecuencia de utilizar transacciones basadas en MRs es la eficiencia de la generación de reglas: transacciones con grandes cantidad de ítems provocan un mayor costo en tiempo y, sobre todo, memoria en el proceso de generación, pudiendo resultar bastante problemático si a las combinaciones anteriores le agregamos más combinaciones para los parámetros de `min_support` y `min_confidence`. Por último, tener transacciones con pocos ítems implica que la mayoría de los ítems irrelevantes son eliminados antes de comenzar, simplificando el proceso de experimentación y análisis en todos los casos.

En cuanto a reglas de asociación, los resultados indican que generar reglas en las

familias comparables utilizando transacciones basadas en MRs con un cierto `min_len` no aporta más información que utilizar transacciones formadas por todas las subsecuencias de largo `k=min_len`. Por lo tanto, como generar las KMERAS resulta en la práctica mucho más eficiente, no tiene sentido computar MRs si lo único que se busca es la obtención rápida de reglas de asociación.

Vimos también que como ambos tipos de transacciones generan las mismas reglas, significa necesariamente que los ítems que determinan la generación de reglas están contenidos en los MRs (pues las transacciones de MRs las generan), por lo que los MRs siguen proporcionando un gran valor teórico en el análisis de secuencias de proteínas.

Para explorar en un trabajo futuro, otro posible tipo de transacciones podría ser un punto intermedio: de todos los k-meros, quedarse sólo con aquellos más frecuentes (los que no lo son, serían descartados por el algoritmo *Apriori* en la primer iteración de todas formas). Estimamos que generará las mismas reglas y disminuirá el tamaño de las transacciones, presentando las ventajas de ambos métodos estudiados. Sin embargo, agrega el problema adicional de definir el parámetro para la frecuencia de corte. Éste podría ser igual al `min_support`, aunque una vez generadas las transacciones, si se quiere experimentar con soportes menores en la generación de reglas, será necesario regenerar las transacciones pues los ítems con menores frecuencias ya fueron eliminado, dificultando la exploración.

# 6. ANÁLISIS DE RELEVANCIA EN REGLAS DE ASOCIACIÓN

## 6.1. Repeticiones maximales frecuentes y lift

Existen varias maneras para decidir si una regla es relevante o de interés. La forma más sencilla es ver el *soporte* de la regla (Definición 1.2.7), pero una de las desventajas que presenta es que mide la frecuencia de los ítemsets que componen a la regla pero no tiene en cuenta la correlación entre ellos. Si usamos solo esta medida, ítemsets con mucha frecuencia podrían parecer "mejores" que ítemsets que quizá aparecían menos pero con mayor correlación.

Otro método podría ser usar la *confianza* de la regla, pero esto también puede ser problemático, aunque de forma más sutil, como veremos a continuación. Recordemos la definición de la confianza para una regla (Definición 1.2.8). Si A y B son ítemsets, la *confianza* de la regla $A \to B$ se define como:

$$confianza(A \to B) = \frac{soporte(A \cup B)}{soporte(A)} \tag{6.1}$$

Para entender cuál es el problema con tomar la confianza para analizar la relevancia, pensemos en un caso extremo. Si miramos la ecuación 6.1 y suponemos que un ítem B aparece en todas las transacciones, resulta que la confianza de una regla que tenga a B como consecuente será cercana al 100 % (porque en ese caso, $soporte(A \cup B) = soporte(A)$). Esta confianza de 100 % resultaría engañosa, porque si B apareciera en todas las transacciones, las reglas que incluyan a B como consecuente no serían interesantes. Una confianza alta, en este caso, no resulta buen criterio para decidir si una regla es relevante o no.

Una medida que podemos tomar para resolver estos problemas es el *lift* [11]:

$$lift(A \to B) = \frac{confianza(A \to B)}{soporte(B)} = \frac{soporte(A \cup B)}{soporte(A) * soporte(B)} \tag{6.2}$$

El *lift* compara la frecuencia de un patrón con respecto a la frecuencia que se obtiene si asumiríamos una distribución independiente de sus componentes. Podemos distinguir tres casos:

- *lift* $= 1$ si A y B son independientes estadísticamente. Es decir, si la probabilidad de aparición de A es independiente de la probabilidad de aparición de B.
- *lift* $> 1$ si A y B están positivamente correlacionadas.
- *lift* $< 1$ si A y B están negativamente correlacionadas.

Esta forma de medir tampoco es perfecta, pero consideramos que para nuestros propósitos es superadora de las otras vistas (*soporte* y *confianza*). Queda como trabajo futuro explorar otras posibles medidas.

Usaremos el *lift* para poder ordenar las reglas con algún criterio y entender cuál es su relevancia. Para esto, intentaremos encontrar relaciones con los MRs más frecuentes. ¿Aparecerán los MRs más frecuentes en las reglas de mayor *lift*? ¿Y en las de menor *lift*?

### Metodología

Generaremos reglas de asociación para las familias comparables, utilizando transacciones refinadas en modo *minimum*, compuestas por el conjunto total del MRs (ALL). Variamos los valores de `min_len` entre 4 y 6, y fijamos `min_support` en 0.025 y `min_confidence` en 0.90. Luego, calculamos el *lift* de las reglas generadas.

Tomamos las 15 reglas de mayor y menor *lift* y comparamos sus ítems con los MRs de mayor frecuencia para poder encontrar similitudes.

### Resultados y discusión

Presentamos en la Tabla 6.1 un listado de los 15 MRs con mayor soporte tomados de las transacciones de la familia `NEWAnk`, y un listado con las 15 reglas de mayor *lift*, para esa misma familia, en la Tabla 6.2. Ambos resultados fueron generados de la forma explicada en la subsección de Metodología.

| MR | Soporte |
|---|---|
| TPLH | 0.595 |
| TALH | 0.535 |
| LHLA | 0.445 |
| GADV | 0.340 |
| HLAA | 0.298 |
| PLHL | 0.296 |
| ADVN | 0.287 |
| LLLE | 0.279 |
| RTPL | 0.261 |
| KLLL | 0.257 |
| GADP | 0.250 |
| VKLL | 0.242 |
| RLLL | 0.238 |
| LHYA | 0.236 |
| ELLL | 0.234 |

*Tab. 6.1:* Listado de los 15 MRs con mayor soporte en la familia `NEWAnk`, tomados de transacciones refinadas en modo *minimum*, utilizando MRs de la categoría ALL y `min_len` fijo en 4.

| Regla | Lift |
|---|---|
| {LFLA, TPLF} → {PLFL} | 25.9981 |
| {KHGA, LLKH} → {LKHG} | 22.6905 |
| {KNGA, LLKN} → {LKNG} | 22.5792 |
| {DNGA, LLDN} → {LDNG} | 22.5484 |
| {LLSH, SHGA} → {LSHG} | 21.7620 |
| {DKGA, LLDK} → {LDKG} | 21.5482 |
| {LLSR, SRGA} → {LSRG} | 20.9019 |
| {GHLE, LEVV} → {HLEV} | 20.8868 |
| {LLSK, SKGA} → {LSKG} | 20.1765 |
| {LLQH, QHGA} → {LQHG} | 19.3067 |
| {LLQA, QAGA} → {LQAG} | 19.0432 |
| {DALD, LDEC} → {ALDE} | 18.4383 |
| {AARN, RNGH} → {ARNG} | 18.3530 |
| {DRGA, LLDR} → {LDRG} | 17.7346 |
| {PLFL, TPLF} → {LFLA} | 17.1549 |

*Tab. 6.2:* Listado de las 15 reglas con mayor *lift* en la familia `NEWAnk`. Utilizamos transacciones refinadas en modo *minimum*, con MRs de la categoría ALL, con `min_len` 4, `min_support` 0.025 y `min_confidence` 0.90.

Ninguno de los MRs más frecuentes aparece en las reglas con mayor *lift*. Lo mismo sucede al aumentar el valor de `min_len` a 5 y 6 (resultados no mostrados). Pareciera que utilizar el *lift* como criterio nos ayuda a elegir reglas que son interesantes por su correlación y no tanto por la cantidad de apariciones de sus ítems.

Repetimos el experimento con las demás familias comparables. Recordemos que con `TPR1` se generaron solo 4 reglas al fijar `min_support` en 0.025, así que también probamos con soportes menores a ese para poder generar más reglas. Los resultados fueron equivalentes a los observados con `NEWAnk`. Con `LRR1` también obtuvimos el mismo resultado, aunque con una excepción: el MR más frecuente (LDLS) apareció en 4 de las 15 reglas de mayor *lift*. La explicación que encontramos es que el soporte de este ítem es sumamente alto (0.813), por lo que resulta razonable que aparezca en algunas de las reglas. Es interesante notar que el *lift* cumplió parcialmente su "función" al aparecer ese ítem en 4 reglas y no en todas ellas. LDLS es un MR que aparece en casi todas las transacciones, por lo que no todas las reglas que lo contienen parecen ser tan interesantes.

Nos preguntamos si al tomar las 15 reglas de **menor** *lift* y los MRs de mayor frecuencia se produce el efecto contrario. Es decir, queremos saber si las frecuencias altas influencian de forma excesiva a las reglas con menor *lift*. Esto nos permitiría analizar reglas cuyo *lift* sea menor a 1, que también podrían ser interesantes por indicar una correlación negativa. Los resultados obtenidos se pueden ver en la Tabla 6.3.

| Regla | Lift |
|---|---|
| {AGAD, HLAA, PLHL, TALH} → {TPLH} | 1.5129 |
| {HCAA, LHLA, TALH} → {TPLH} | 1.5130 |
| {HLAA, HVAA, LHLA, LHVA, TALH} → {TPLH} | 1.5130 |
| {ALHY, PLHL} → {TPLH} | 1.5131 |
| {LHIA, LHLA, PLHI} → {TPLH} | 1.5136 |
| {ELLL, LHIA, PLHI} → {TPLH} | 1.5137 |
| {AGAD, HLAA, LHLA, PLHL} → {TPLH} | 1.5139 |
| {GANV, PLHI} → {TPLH} | 1.5140 |
| {HYAA, RTPL, TALH} → {TPLH} | 1.5142 |
| {GADP, PLHL, TALH} → {TPLH} | 1.5142 |
| {GADP, LHLA, PLHL, TALH} → {TPLH} | 1.5143 |
| {ADPN, GADP, PLHL} → {TPLH} | 1.5144 |
| {LHLA, LLEA, PLHL, TALH} → {TPLH} | 1.5144 |
| {ADVN, DVNA, GADV, PLHL} → {TPLH} | 1.5144 |
| {LHAA, PLHA} → {TPLH} | 1.5145 |

*Tab. 6.3:* Listado de las 15 reglas con menor *lift* en la familia `NEWAnk`, utilizando transacciones refinadas en modo *minimum*, con MRs de la categoría ALL, con `min_len` 4, `min_support` 0.025 y `min_confidence` 0.90.

Al comparar estas reglas con los MRs de mayor frecuencia (Tabla 6.1) observamos que las reglas están casi exclusivamente compuestas por los ítems más frecuentes. Puede verse en dicha tabla que TPLH es el MR más frecuente (con un soporte de 0.595) y que en las reglas de menor *lift* aparece siempre como el consecuente. Esto nos indica que utilizar el *lift* tiene sentido, ya al usarlo podemos descartar varias reglas fácilmente.

Algo interesante a notar es que el *lift* de todas las reglas generadas, incluyendo las de menor *lift*, es de un valor superior a 1,5. Siguiendo la definición, esto significa que los antecedentes y consecuentes están positivamente correlacionados en todos los casos. Como nuestros ítems no provienen de una base de datos "natural", sino que vienen de ser los MRs y luego de un refinamiento de transacciones, nos hace sospechar que quizá el *lift* no sea tan buena medida para nuestro conjunto. Continuaremos explorándolo en la siguiente sección.

## 6.2. Relevancia de reglas perdidas

Vimos en la Sección 5.4 que se pierden reglas al usar los MRs de la categoría NE en reemplazo de ALL. Analicemos la importancia de las reglas perdidas. Nuestra hipótesis de es que dichas reglas deberían ser de baja relevancia bajo la medida de *lift*.

### Metodología

Tomamos las reglas presentadas en la Tabla 5.4, donde se muestran reglas que se pierden al reemplazar los MRs de la categoría ALL por los de la categoría NE en las transacciones de la familia `NEWAnk`, y calculamos el *lift* de cada una de ellas. Ver la tabla referenciada para más detalles de la generación.

### Resultados y discusión

Podemos ver en la Tabla 6.4 los valores de *lift* de las reglas de la familia `NEWAnk` que se perdieron al usar MRs de la categoría NE. La única regla perdida usando `min_len` 4 tiene un *lift* aprox. de 4, las reglas de `min_len` 5 tienen *lift* cercano a 30, y las reglas de `min_len` 6 tienen dos valores muy diferentes, una aprox. 7 y otra cercano a 20.

| `min_len` | **Regla** | **Lift** |
|---|---|---|
| 4 | {HYAC} → {LHYA} | 3.9534 |
| 5 | {TALHM} → {ALHMA} | 30.4918 |
| 5 | {LKNGA} → {LLKNG} | 30.9775 |
| 6 | {LEKGAD} → {LLEKGA} | 19.5201 |
| 6 | {RTALHL} → {TALHLA} | 6.9792 |

*Tab. 6.4:* Reglas generadas por transacciones que utilizan MRs de categoría ALL que se pierden al utilizar la categoría NE, para valores de `min_len` entre 4 y 6, correspondientes a la Tabla 5.3. Para cada una de ellas, mostramos su *lift* asociado.

Vemos que en las cinco reglas los valores de *lift* son muy altos. Para dar una idea de la magnitud, el percentil 75 de las reglas con `min_len` 4 es 2.66. En las de `min_len` 5, tenemos *lifts* cercanos a 30: un valor superior a todos los valores de reglas de largo 4 de esa misma familia (ver listado de las 15 mayores en la Tabla 6.2).

Miremos con más detalle estas reglas. Puede observarse que entre antecedentes y consecuentes aparece un solapamiento: ambas partes de la regla son casi iguales salvo un caracter, como si hubiese un desplazamiento a izquierda o derecha.

Por ejemplo, la regla

$$\{\underline{\mathbf{LKNG}}\mathrm{A}\} \rightarrow \{\mathrm{L}\underline{\mathbf{LKNG}}\}$$

parece provenir del string

$$\mathrm{L}\underline{\mathbf{LKNG}}\mathrm{A}$$

Entonces si bien la regla no es completamente irrelevante, en la práctica no nos es interesante ni nos aporta información nueva. No es información nueva que "LLKNGA" es una cadena que aparece mucho en las proteínas, porque resulta que es un MR. Como tiene muchas apariciones, también es lógico esperar que, en los casos donde éste aparece, sus substrings (que resultaron ser MRs) también aparezcan mucho.

Este caso nos sirve para advertir que el *lift* nos ofrece un panorama incompleto y, en nuestro contexto de MRs, no puede ser determinante para decidir el interés de una regla.

### Conclusión

El *lift* suele ser una medida útil en el contexto general de reglas de asociación y posee muchas ventajas con respecto a usar únicamente el soporte y la confianza, como poder descartar reglas que no son muy interesantes por no aportar mucha información. Sin embargo, vimos también que en nuestro contexto tener un *lift* alto no implica que la regla nos esté aportando información nueva.

A diferencia de dominios dónde las reglas de asociación encajan de forma más directa (por ejemplo, productos en supermercados), estamos trabajando ahora en un dominio donde los ítems son substrings, con intersecciones entre sí, de una familia de strings más grandes provenientes de un dominio biológico. Este hecho complejiza muchísimo el desarrollo de una medida única para elegir reglas relevantes, y queda como trabajo futuro realizar una investigación más extensa de otras posibles medidas.

Otra posibilidad a considerar es el uso de otro tipo de reglas de asociación, como lo son las reglas de asociación generalizadas (o jerárquicas), donde pueden definirse inclusiones (jerarquías) entre ítems [17].

## 6.3. Visualizador de reglas y proteínas

Una forma de ayudar al análisis de reglas es usar la herramienta visualizadora de reglas, Protein Rule Visualization Tool— una aplicación web desarrollada por Enríquez, que permite ver algunas características básicas e información relevante utilizada en su trabajo, como distancia entre ocurrencias o clasificaciones de reglas. Para utilizar esta herramienta, es necesario generar una base de datos a partir de las reglas y secuencias de proteínas de la forma descripta en el Apéndice.

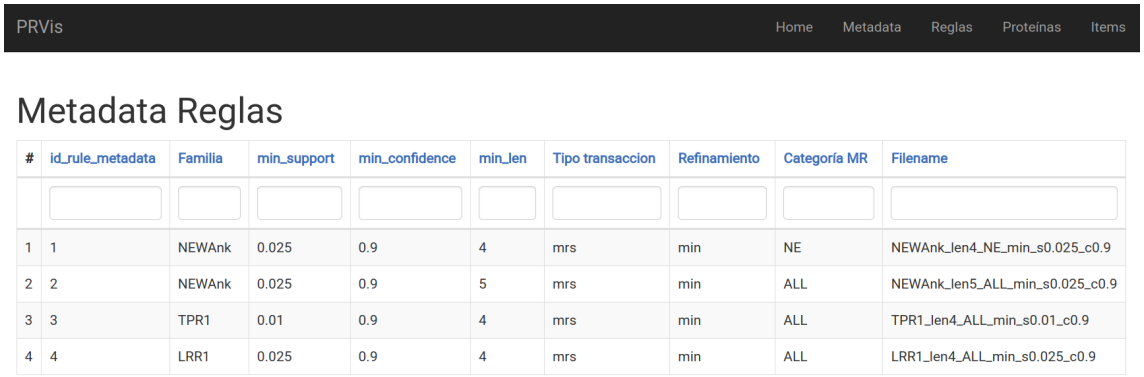
Extendimos la herramienta de visualización para permitir la comparación de familias múltiples y agregamos nueva información sobre las reglas, como su soporte, confianza y lift. El cambio más importante que introdujimos es que ahora cada regla tiene asociada una *metadata*. Esta *metadata* indica sobre cómo fue generada la regla e incluye, por ejemplo, el tipo de transacciones utilizadas, el `min_support` y la familia que las origina. Gracias a la *metadata* es posible diferenciar reglas provenientes de diferentes familias, y permite incluso poder analizar reglas de una misma familia pero que fueron generadas con parámetros distintos.

En la Figura 6.1 mostramos una captura de pantalla de la sección *Metadata*, donde puede verse el listado de la información de cómo fueron generadas las reglas. Para poder tener trazabilidad de la generación y poder desambiguar reglas repetidas, cada regla presente en el sistema muestra su *metadata* asociada, como puede observarse en la Figura 6.2. En esta pantalla también pueden verse, asociadas a cada regla, las medidas mencionadas a lo largo de este trabajo (soporte, confianza y lift) y la familia a la cuál pertenece. Es posible escribir en los recuadros para filtrar las reglas que nos interesan (por ejemplo, las que contienen un cierto ítem, o las que pertenecen a determinada familia). Los recuadros de soporte, confianza y lift sirven para quedarse únicamente con reglas cuyos valores sean mayores a los indicados. La columna *Tipo* hace referencia a una posible clasificación desarrollada en el trabajo de Enríquez.

Si accedemos a la vista sobre una proteína particular, como en la Figura 6.3, podemos observar un listado con las reglas que aplican sobre la secuencia. Al hacer clic en alguna regla, se marca con color la ubicación del ítem dentro de dicha secuencia. Como podrían existir reglas repetidas dentro del sistema (por haber sido generadas con parámetros distintos), indicamos junto a cada regla el identificador de su *metadata* para poder diferenciarlas.

Además de los cambios visuales y la incorporación de la *metadata*, ahora pueden agregarse proteínas y reglas nuevas a una base de datos preexistente sin necesidad de generarla desde cero. También, como la generación de la base de datos es un proceso muy costoso computacionalmente (por calcular algunas estadísticas presentes en el trabajo de Enríquez,

como la distancia promedio entre repeticiones o información sobre los cubrimientos), modificamos el script generador de la base para que pueda ejecutarse en modo *multithread*. Queda como trabajo futuro enfocarse en la optimización de los cálculos, debido a que ahora es problemático ingresar conjuntos de proteínas y reglas muy grandes (como los generados por `LRR1`).




| # | id_rule_metadata | Familia | min_support | min_confidence | min_len | Tipo transaccion | Refinamiento | Categoría MR | Filename |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | NEWAnk | 0.025 | 0.9 | 4 | mrs | min | NE | NEWAnk_len4_NE_min_s0.025_c0.9 |
| 2 | 2 | NEWAnk | 0.025 | 0.9 | 5 | mrs | min | ALL | NEWAnk_len5_ALL_min_s0.025_c0.9 |
| 3 | 3 | TPR1 | 0.01 | 0.9 | 4 | mrs | min | ALL | TPR1_len4_ALL_min_s0.01_c0.9 |
| 4 | 4 | LRR1 | 0.025 | 0.9 | 4 | mrs | min | ALL | LRR1_len4_ALL_min_s0.025_c0.9 |

*Fig. 6.1: Metadata* de las reglas. Cada regla tiene asociado una *metadata*, que indica la familia de proteínas usada, el tipo de transacciones, su refinamiento y demás parámetros que la generaron.


## Reglas

| id_regla | id_rule_metadata | Familia | Regla | Soporte | Confianza | Lift | Tipo |
|---|---|---|---|---|---|---|---|
| | | NEWAnk | | | | | |
| 1 | 1 | NEWAnk | {GRTP} => {RTPL} | 0.51137 | 0.96197 | 1.56204 | Agrega +L |
| 2 | 1 | NEWAnk | {HLAA} => {LHLA} | 0.51053 | 0.94246 | 1.40012 | Agrega +L |
| 3 | 1 | NEWAnk | {GRTP,TPLH} => {RTPL} | 0.40438 | 0.95618 | 1.55264 | Overlapping |
| 4 | 1 | NEWAnk | {HLAA,TALH} => {LHLA} | 0.40859 | 0.96806 | 1.43816 | Overlapping |
| 5 | 1 | NEWAnk | {HLAA,TPLH} => {LHLA} | 0.4524 | 0.96237 | 1.4297 | Overlapping |
| 6 | 1 | NEWAnk | {LHLA,RTPL} => {TPLH} | 0.41028 | 0.92762 | 1.31551 | Overlapping |
| 7 | 2 | NEWAnk | {ALHL} => {LHLA} | 0.38332 | 0.97015 | 1.44126 | N/A |
| 8 | 2 | NEWAnk | {GRTP} => {RTPL} | 0.51137 | 0.96197 | 1.56204 | Agrega +L |
| 9 | 2 | NEWAnk | {HLAA} => {LHLA} | 0.51053 | 0.94246 | 1.40012 | Agrega +L |
| 10 | 2 | NEWAnk | {GRTP,TPLH} => {RTPL} | 0.40438 | 0.95618 | 1.55264 | Overlapping |
| 11 | 2 | NEWAnk | {HLAA,TALH} => {LHLA} | 0.40859 | 0.96806 | 1.43816 | Overlapping |

*Fig. 6.2:* Listado de reglas presentes. Es posible escribir en los recuadros para filtrar los elementos y, por ejemplo, mostrar las reglas de una familia específica. Al hacer clic en los nombres de las columnas pueden ordenarse los elementos de forma ascendente o descendente.

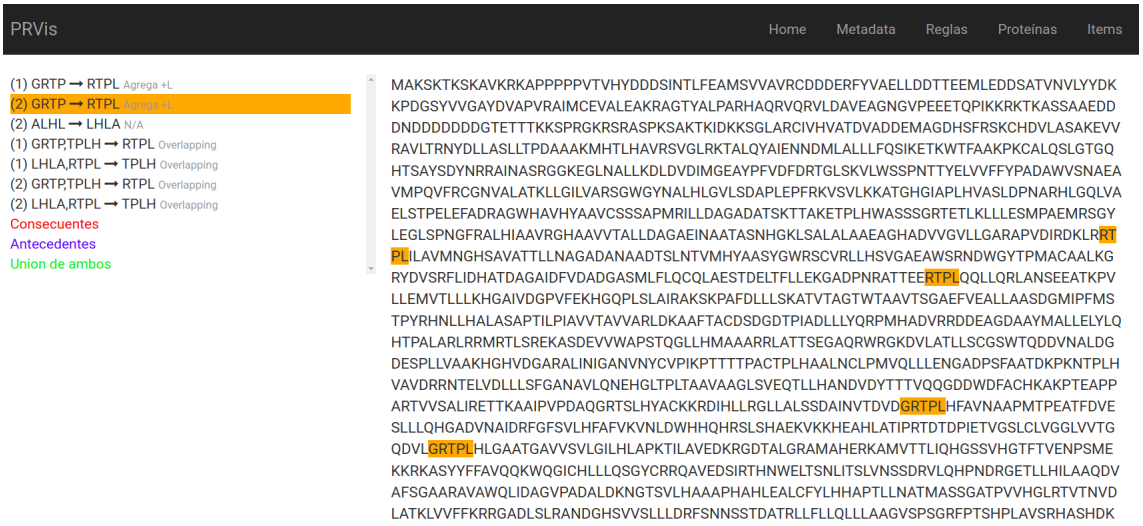



*Fig. 6.3:* Ejemplo de proteína y las reglas que aplican a su secuencia. Al seleccionar una regla, puede verse en color en qué parte de la secuencia están presentes sus ítems. El número que se encuentra entre paréntesis a la izquierda de la regla indica el *id* de su *metadata*.

# 7. CONCLUSIONES

En el presente trabajo continuamos con el análisis de coocurrencias de repeticiones maximales, iniciada en la tesis de Enríquez con la familia *Ankyrin*, y lo ampliamos a múltiples familias de proteínas. Para esto, definimos un grupo de familias comparables (`NEWAnk`, `TPR1` y `LRR1`) y generamos reglas de asociación con cada una de ellas.

Aunque para elegir las familias nos basamos en la similitud de algunas de sus características, la cantidad de reglas generadas con cada una de ellas resultó diferir en órdenes de magnitud. Para entender estas diferencias, realizamos un análisis de las frecuencias de los ítems presentes en las transacciones, tanto de forma individual como tomándolos de a pares. Analizar las frecuencias de estos pares fue suficiente para explicar las diferencias en los órdenes de magnitud de las cantidades de reglas generadas, pero insuficiente para determinar la generación de reglas de forma completa debido a la necesidad de tener en cuenta las combinaciones de tres o más ítems, que cobran relevancia al bajar el umbral de soporte mínimo.

Una vez explicadas las diferencias en la cantidad de reglas generadas, analizamos las similitudes de las reglas en sí, y observamos de que las familias comparables no comparten ninguna regla de asociación, sin importar la variación de parámetros utilizada para generarlas. Para comprender estos resultados, realizamos un análisis en las intersecciones de las repeticiones maximales de las transacciones de cada familia y vimos que eran varios los ítems compartidos. La explicación de la intersección vacía entre reglas pudimos encontrarla recién al analizar la intersección de las repeticiones maximales de alta frecuencia. Resultó que cada familia posee un núcleo propio de repeticiones maximales frecuentes, con baja intersección entre familias, lo que provoca que no existan reglas en común. Este fenómeno se profundiza al tomar repeticiones maximales cada vez más largas.

Nos propusimos optimizar (temporal y espacialmente) el proceso de generación de reglas. Para eso nos enfocamos primero en la distribución de los largos de las repeticiones maximales presentes en las transacciones. Descubrimos que, por cómo funciona el refinamiento de transacciones, la gran mayoría de los ítems presentes en ellas tienen un largo que coincide con el parámetro `min_len` que las generó. En base a este resultado, decidimos reemplazar el refinamiento que estábamos utilizando por uno más simple, el modo *minimum*, y con éste pudimos obtener los mismos resultados pero con un orden temporal de cómputo mucho más eficiente.

A continuación, intentamos mejorar la generación de transacciones en la etapa previa al refinamiento. Habíamos observado que son solo algunas de las repeticiones maximales las que determinan la generación de las reglas (las de mayor frecuencia), así que razonamos que debería existir una manera de reducir el tamaño de las transacciones y seguir obteniendo los mismos resultados. En base a una clasificación preexistente para las repeticiones maximales (NE, NN y SMR), hicimos pruebas varias para entender cómo se diferenciaban las reglas generadas usando repeticiones de cada una de estas categorías en reemplazo del conjunto completo. Para `NEWAnk`, al utilizar la categoría NE logramos reducir el tamaño de algunas transacciones a un 50 % de su tamaño original, perdiendo unas pocas reglas en el proceso. Sin embargo, para la familia `LRR1`, encontramos combinaciones de parámetros donde al utilizar esta única categoría se pierden el 40 % de las reglas. Así, esta optimización no resultó generalizable, ya que es necesario analizar cada caso particular para poder decidir

si se está dispuesto a perder reglas o no.

Como última propuesta de optimización, nos replanteamos el hecho de usar repeticiones maximales. En el algoritmo *Apriori* se eliminan los ítems de menor frecuencia como primer paso del proceso, por lo que podría no tener sentido precalcular las repeticiones maximales al ser éstas, por definición, patrones que como mínimo necesariamente se repiten. Resultó posible generar un tipo de transacciones (las transacciones KMERAS) que no utiliza las repeticiones maximales pero que finalmente genera exactamente las mismas reglas. Encontramos que la razón por la que esto es así es porque los ítems que pertenecen a las reglas terminan siendo repeticiones maximales, y éstos siempre se encuentran incluidos en el nuevo tipo de transacciones.

Por ultimo, estudiamos la medida de lift para ordenar nuestras reglas y poder encontrar aquellas que en principio son más relevantes, pero, si bien presentó cierta utilidad, encontramos casos dónde la medida resultó insuficiente al aplicarlo a nuestro dominio biológico. Para contribuir al análisis manual de reglas, extendimos el visualizador de proteínas para que pueda soportar múltiples familias, agregamos que en las reglas asociadas se muestren algunas medidas básicas, incorporamos una trazabilidad sobre con qué parámetros fueron generadas las reglas y permitimos mostrar reglas de una misma familia pero generadas con parámetros diferentes.

Si tenemos en cuenta todos los resultados obtenidos a lo largo de este trabajo, profundizamos el entendimiento de las correlaciones de las repeticiones maximales en múltiples familias y comprendimos la importancia de las frecuencias de los ítems en la generación de reglas de asociación. Vimos que las reglas generadas resultaron independientes en cada una de las familias que tomamos, siendo la razón de esto que las repeticiones maximales con mayor frecuencia forman un núcleo propio en cada familia. De esta manera, las repeticiones maximales siguen siendo una herramienta importante en el análisis de patrones en secuencias de proteínas.

# 8. TRABAJOS FUTUROS

A lo largo del desarrollo de este trabajo nos encontramos con posibles líneas de investigación que pueden ser profundizadas en el futuro. A continuación, mencionaremos algunas de ellas.

Las familias comparables que elegimos son sólo tres (`NEWAnk`, `TPR1` y `LRR1`), y fueron tomadas de un conjunto de datos formado por 26 familias repetitivas y 20 globulares. Podrían tomarse otros conjuntos de familias, quizá muy distintos al que tomamos nosotros, y repetir los experimentos para contrastar los resultados con los que obtuvimos.

En todos nuestros experimentos mantuvimos fijo el parámetro `min_confidence`. Sería interesante entender en profundidad de qué formar este valor afecta la generación de reglas.

Vimos que las tres familias no generaban ninguna regla en común, pero podría estar sucediendo que hayan reglas que sean casi iguales, diferenciándose por un único aminoácido. Se podría desarrollar un método que tenga en cuenta estas posibles nuevas equivalencias a la hora de analizar intersecciones entre conjuntos de reglas. Para esto será necesario definir algunos criterios de distancia, dado que no parece lo mismo que esta nueva equivalencia se dé en dos reglas cortas y sencillas, o en dos reglas largas y con varios ítems. A diferencia de simples caracteres, aquí podría tener influencia saber cuál es ese aminoácido distinto.

Observamos al realizar el estudio sobre las frecuencias de pares de ítems, como en el ejemplo de la Figura 3.5, que las curvas con las que decrece cada familia podrían responder a caídas exponenciales con diferentes coeficientes. Podría intentarse explicar la regularidad de estas curvas y quizá, usando otras familias, contrastarlas con las curvas que obtuvimos.

En la Sección 3.4 intentamos desarrollar un método para estimar la cantidad de reglas que se generan a partir del conjunto de datos inicial, pero no lo logramos al enfocarnos en el estudio de frecuencias de pares de ítems. Encontrar un método así podría tener utilidad si se quisiera generar una cantidad específica de reglas sin tener que ejecutar una búsqueda binaria de los valores de los parámetros ejecutando cada vez el proceso completo.

De forma similar a lo que pasaba con las intersecciones de reglas, en la Sección 4.3 encontramos que las repeticiones maximales frecuentes forman un núcleo propio en cada familia, pero las intersecciones que se utilizaron para determinar esto utilizan la comparación exacta de strings. Podrían existir dos repeticiones que sean casi iguales (por ejemplo, que se diferencien en un único caracter) que tenga sentido considerar equivalentes.

Aunque hicimos cambios en el proceso de refinamiento y éste dejó de ser un problema a nivel cómputo, el concepto de refinamiento surgió como una forma de evitar generar reglas triviales. Sería interesante realizar un estudio similar a este trabajo sin realizar el refinamiento, y pensar otras alternativas para eliminar reglas triviales. Es posible que al refinar estemos perdiendo reglas que podrían ser interesantes, como reglas donde los ítems que las componen tengan largos distintos.

La generación de la base de datos para el visualizador de reglas y proteínas es un proceso muy costoso computacionalmente. Si bien se lo modificó para ser más eficiente y poder correr de forma *multithread*, resulta muy problemático agregar a la base una gran cantidad de reglas por todos los cálculos sobre coberturas en proteínas. Debería ser posible optimizar esos cómputos, o al menos brindar la opción de poder omitirlos.

Hicimos mucho foco en analizar las transacciones y las repeticiones maximales que las componen, pero consideramos que aún hay mucho lugar para el estudio de las reglas

en sí, relacionándolas con el dominio biológico. Para esto, se vuelve necesario una buena forma de clasificarlas y determinar cuáles de ellas son relevantes. Esto no es tarea sencilla, ya que no tenemos definido un criterio claro para saber qué debe tener la regla para ser biológicamente interesante.

Finalmente, notamos que el dominio en el que trabajamos no encaja exactamente con el dominio donde se desarrollaron originalmente las reglas de asociación (el dominio de supermercados), por el hecho de que entre nuestros ítems tenemos relaciones de inclusión no presentes en productos reales. Es decir, como nuestros ítems son strings y los strings pueden solaparse entre sí, los antecedentes pueden contener al consecuente o generarse reglas triviales. Una buena idea podría ser investigar otro tipo de reglas de asociación existentes, como las reglas de asociación generalizadas, que nos permitan definir jerarquías o algún otro tipo de relación que se adecúe mas a nuestro dominio.

# 9. APÉNDICE

## 9.1. Código

Todo el código utilizado puede encontrarse en el siguiente sitio:
`https://github.com/JonSeijo/tesis-rules-code`

## 9.2. Proceso de generación de reglas

El proceso de generación de reglas está formado por tres scripts principales:

1. `generate_transactions.py` - Generación de las repeticiones maximales (MRs) y armado de transacciones preliminares. Por debajo utiliza el programa `tx-generator` creado por Pablo Turjanski, implementado en C++.
2. `clean_transactions.py` - Refinamiento y limpieza de las transacciones para eliminar ítems innecesarios. Esta etapa tiene como objetivo reducir el tamaño de las transacciones para ganar eficiencia y además evitar generar reglas inútiles. Es en esta etapa donde se implementan los filtros de modo *substring* y *superstring* descriptos en la Sección 1.4.
3. `generate_rules.r` - Generación de reglas de asociación a partir de un archivo de transacciones. Se especifican aquí los parámetros `min_support` y `min_confidence`. El resultado es un archivo csv con una regla en cada fila.

Los scripts 1. y 2. fueron escritos en python por nosotros y son *wrappers*: el código original se encontraba ya escrito en C++ y fue el utilizado en el trabajo de Enríquez. Estos wrappers agregan un mejor manejo de los parámetros por consola, valores por defecto y la posibilidad de correr en forma multithread la limpieza de transacciones (especificando el parámetro `threads`). El script 3. es un *wrapper* de la librería *arules* de R, que implementa el algoritmo *Apriori*.

El archivo de reglas que se genera como salida del último script sigue la siguiente convención en el nombre para poder identificar fácilmente cómo fue generado:

`[familia]_len[minlen]_[categoriaMR]_[refinamiento]_s[minsup]_c[minconf].csv`

Por ejemplo, el archivo `NEWAnk_len4_ALL_sub_s0.025_c0.9.csv` contiene las reglas generadas por la familia `NEWAnk`, habiendo utilizado transacciones formadas por el conjunto ALL de MRs (categoría), con `min_len` 4, refinadas en modo *substring*, y fijando para el algoritmo *Apriori* `min_support` 0.025 y `min_confidence` 0.9.

## 9.3. Lista de parámetros

Generación de MRs y transacciones preliminares (`generate_transactions.py`)

- `min_len`: Largo mínimo para los MRs. Consideramos un valor por defecto de 4, salvo que se especifique lo contrario.

- `max_len`: Largo máximo para los MRs. Dejamos el default muy alto (999999) para no acotar el máximo. En la práctica no encontramos MRs de largo mayor a 10.
- `min_proteins`: Cantidad mínima de proteínas en las que debe aparecer un MR para que aparezca en las transacciones. Por defecto no lo limitamos.

Refinamiento de transacciones (`clean_transactions.py`)

- `clean_mode`: Selección del modo de refinamiento de las transacciones. Las opciones son: `substring`, `superstring` o `minimum`. Se utiliza el modo `substring` cuando no haya aclaración.

Generación de reglas de asociación (`generate_rules.r`)

- `transactions_name`: Nombre del archivo que contiene las transacciones.
- `min_support`: Soporte mínimo para el algoritmo *Apriori*. Usamos 0.025 por defecto.
- `min_confidence`: Confianza mínima para el algoritmo *Apriori*. Usamos 0.9 por defecto.

Además de los anteriores parámetros mencionados hay otros relacionados a *filepaths*, *timeouts* y *threads*.

## 9.4. Visualizador de reglas y proteínas

El visualizador de reglas y proteínas utiliza una base de datos que contiene, además de las reglas y las secuencias de proteínas, cálculos adicionales como las posiciones donde ocurren las repeticiones, distancias promedios entre repeticiones, coberturas de reglas, entre otra información descripta en el trabajo de Enríquez.

Para generar la base de datos se utiliza el script `rule_db_generator.py`, y se le debe especificar dónde se encuentran las proteínas de la familia, cuál es el csv de reglas a utilizar y, opcionalmente, el nombre de la base de datos final y la cantidad de *threads* con la cual correr. Se recomienda aprovechar todos los *cores* de la máquina porque éste es un proceso muy intensivo computacionalmente. Ejemplo:

```
python3 -m rule_db_generator --threads=4
--protein_path=../db/canonicalFamilyDataset/familyDataset/NEWAnk/
--rule_file=output/rules/NEWAnk_len4_ALL_sub_s0.025_c0.9.csv
```

La base de datos generada como salida (por defecto `protein-rules.db`) es compatible con `sqlite3`.

El visualizador se encuentra en un repositorio separado de los generadores descriptos anteriormente: dejamos un link en el `README.md` del repositorio principal. La ubicación de la base de datos debe ser especificada dentro de los archivos del visualizador en `config/db.php`. Es necesario instalar las dependencias utilizando `composer install`. Incluimos posibles problemas y sus soluciones en el archivo `README.md`.

Como el visualizador es una aplicación web que combina varias tecnologías, se utiliza una imagen de `docker`. Puede utilizarse el comando `docker-compose up` para levantar todos los servicios. Para utilizar la aplicación una vez levantada la imagen de `docker`, basta con acceder con un navegador a `localhost:81`.

## Bibliografía


[1] Alberts, B., Johnson, A., Lewis, J., Morgan, D., Raff, M., Roberts, K., Walter, P. (2015). Molecular Biology of the Cell (6th ed.), W.W. Norton & Company. https://doi.org/10.1201/9781315735368

[2] Nelson D. L. & Cox M. M. (2005). Lehninger: Principles of Biochemistry (4th ed.), W. H. Freeman & Co., New York. https://doi.org/10.1002/cbf.1216

[3] Buxbaum, E. (2015). Fundamentals of Protein Structure and Function. Springer International Publishing. https://doi.org/10.1007/978-3-319-19920-7

[4] EMBL's European Bioinformatics Institute. What are protein families?

`https://www.ebi.ac.uk/training/online/courses/protein-classification-intro-ebi-resources/protein-classification/what-are-protein-families`. Accessed: 2022-11-25.

[5] MedlinePlus [Internet]. Bethesda (MD): National Library of Medicine (US); ANK1 gene. Last updated October 1, 2010.

`https://medlineplus.gov/genetics/gene/ank1`. Accessed 2023-02-01.

[6] Weiss, O., Jiménez-Montaño, M. A., & Herzel, H. (2000). Information Content of Protein Sequences. In Journal of Theoretical Biology (Vol. 206, Issue 3, pp. 379–386). Elsevier BV. https://doi.org/10.1006/jtbi.2000.2138

[7] Turjanski, P., Parra, R. G., Espada, R., Becher, V., & Ferreiro, D. U. (2016). Protein Repeats from First Principles. In Scientific Reports (Vol. 6, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1038/srep23959

[8] Turjanski, P., & Ferreiro, D. U. (2018). On the Natural Structure of Amino Acid Patterns in Families of Protein Sequences. In The Journal of Physical Chemistry B (Vol. 122, Issue 49, pp. 11295–11301). American Chemical Society (ACS). https://doi.org/10.1021/acs.jpcb.8b07206

[9] Becher, V., Deymonnaz, A., & Heiber, P. (2009). Efficient computation of all perfect repeats in genomic sequences of up to half a gigabyte, with a case study on the human genome. In Bioinformatics (Vol. 25, Issue 14, pp. 1746–1753). Oxford University Press (OUP). https://doi.org/10.1093/bioinformatics/btp321

[10] Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93. the 1993 ACM SIGMOD international conference. ACM Press. https://doi.org/10.1145/170035.170072

[11] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, (2005). Introduction to Data Mining, 1st edition, Pearson.

[12] Enríquez, J. M. (2018). Análisis de co-ocurrencia de repeticiones maximales en proteínas utilizando reglas de asociación. Tesis de Licenciatura en Ciencias de la Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

[13] `https://github.com/mhahsler/arules`. Accessed 2023-02-01.

[14] `https://www.ebi.ac.uk/interpro/entry/pfam/PF00023`. Accessed 2023-02-01.

[15] `https://sites.google.com/site/proteinphysiologylab`

[16] Taillefer, E., & Miller, J. (2014). Exhaustive Ccomputation of Exact Duplications Via Super and Non-Nested Local Maximal Repeats. In Journal of Bioinformatics and Computational Biology (Vol. 12, Issue 01, p. 1350018). World Scientific Pub Co Pte Lt. https://doi.org/10.1142/s0219720013500182

[17] Srikant, R., & Agrawal, R. (1997). Mining generalized association rules. In Future Generation Computer Systems (Vol. 13, Issues 2–3, pp. 161–180). Elsevier BV. https://doi.org/10.1016/s0167-739x(97)00019-8