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

Funciones de distancia para la clasificación de familias de proteínas

Tesis de Licenciatura en Ciencias de la Computación

Agustina Ciraco

Director: Pablo Turjanski
Codirector: Diego Ferreiro
Buenos Aires, 2020

FUNCIONES DE DISTANCIA PARA LA CLASIFICACIÓN DE FAMILIAS DE PROTEÍNAS

Las proteínas son grandes moléculas compuestas por cadenas de aminoácidos. Una posible abstracción de la estructura de una proteína es a través de una secuencia de caracteres, donde cada aminoácido se corresponde con un caracter. Esta representación se corresponde con lo que se denomina estructura primaria de una proteína.

En la naturaleza, existen ciertas proteínas que presentan patrones estructurales recurrentes en su estructura primaria. Estas macromoléculas pueden ser clasificadas de acuerdo al largo de la mínima unidad de repetición que las constituyen. Aquellas para las cuales sus patrones de repetición son cortos (menores o iguales a 5 aminoácidos), se denominan fibrilares; para cuyas repeticiones se componen de 6 a 60 aminoácidos se denominan repetitivas; y, finalmente las restantes, se denominan globulares.

En el presente trabajo abordamos el desafío de proponer una función de distancia entre familias de proteínas, para su clasificación, a partir de un subconjunto de sus patrones de repetición maximales (MRs).

Como paso previo a la propuesta de una función de distancia, propusimos la estructura de Trie para contener los prefijos de los MRs de las familias de las proteínas. Partiendo de esa estructura exploramos la posibilidad de utilizar algoritmos existentes dentro del campo de las redes de computadoras para la comparación de dichas estructuras. Los resultados obtenidos en esta dirección no fueron exitosos, pues no pudimos encontrar un algoritmo que cumpliera con nuestros requerimientos.

A partir del resultado anterior, decidimos proponer una función propia de distancia entre familias de proteínas. Exploramos diversas alternativas, siempre basadas en prefijos de MRs. A todas ellas las pusimos a prueba utilizando como caso de estudio más de 50 familias de proteínas naturales (repetitivas y globulares) y de control. Los resultados obtenidos nos permitieron, en algunas casos, discriminar entre familias de proteínas naturales y de control. Sin embargo, no hemos podido hallar una función que permita agrupar, por un lado familias de proteínas repetitivas, y por el otro globulares.

En vista de los resultados obtenidos se puede considerar la posibilidad de que para lograr el objetivo de separar las familias de proteınas globulares de las repetitivas, no alcance solo con los patrones ya que los mismos quizás no posean suficiente información. Tal vez, para poder distinguirlas falte, a modo de ejemplo, información acerca del código de plegado; o quizás sea necesario utilizar otra representación del alfabeto. Queda como línea futura de trabajo pensar alternativas para poder lograr incorporar nueva información.

Palabras claves: Familias de Proteínas, Repeticiones Maximales, Función de Distancia, Trie.

AGRADECIMIENTOS

A Pablo y a Diego por enseñarme sobre el proceso de investigación y sobre el área, y por tenerme mucha paciencia.
A todo UTI por la buena onda y el apoyo en esta etapa final.
A todo el DC por sus excelentes profesores y personas que lo componen, siempre tratando de mejorar y por la solidez en conocimientos que brindan.
A todos mis compañeros de carrera con quienes he compartido distintas materias o trabajos o experiencias en los que he aprendido mucho.
A Quality Girls y Silvia por enseñarme a autogestionarme y distintas técnicas para mejorar en mi gestión de proyectos y por darme el espacio de poder seguir avanzando en mi carrera.
A las chicas del colegio por toda su paciencia en mis juntadas complicadas.
A Fondo de Tabla por todas las experiencias compartidas.
A CocomielSort por los aprendizajes y competencias compartidas.
A Facu Girls por todos los Tps, finales y experiencias compartidas y por toda la fuerza que siempre me dan y me dieron para seguir en la carrera.
A todos los Pousa por el acompañamiento, apoyo constante, y por los días de niñeros.
A Mamá por su aguante incondicional, por enseñarme y brindarme curiosidad en el área científica.
A Juli por su gran soporte.
A Papá por haberme generado las primeras intrigas y conocimientos en el área de computación.
A Lisa por acompañarme en todos los paseos, días de estudio y su contención perruna.
A Garbanza por acompañarme en esta última etapa.
A Nico por la experiencia en la maternidad y las enseñanzas que eso trajo, entre ellas un mejor manejo de tiempo que me permitió llegar a mis objetivos.

A Fede por todo su amor, apoyo, mi compañero inigualable de la vida. *Come what may.*

Índice general

1. INTRODUCCIÓN Y DEFINICIONES

1.1. Introducción

Las proteínas son macromoléculas, moléculas grandes, compuestas por cadenas de aminoácidos. Los aminoácidos son compuestos orgánicos que contienen grupos característicos (del grupo amino y del grupo carboxilo), y cuyos elementos básicos son Carbono (C), Hidrógeno (H), Oxígeno (O), y Nitrógeno (N) (ver Figura 1.1(a)) [1]. Se conocen muchos aminoácidos, pero en particular, existen unos 20 aminoácidos diferentes que conforman la mayoría de las proteínas conocidas. Las agrupaciones de dos aminoácidos se las conoce como dipéptidos (ver Figura 1.1(b)); de tres aminoácidos se las denomina tripéptidos; y de 4 o más se las conoce como polipéptidos (ver Figura 1.2). A causa de esto último, a las proteínas también se las conoce como polipéptidos [1, 2].

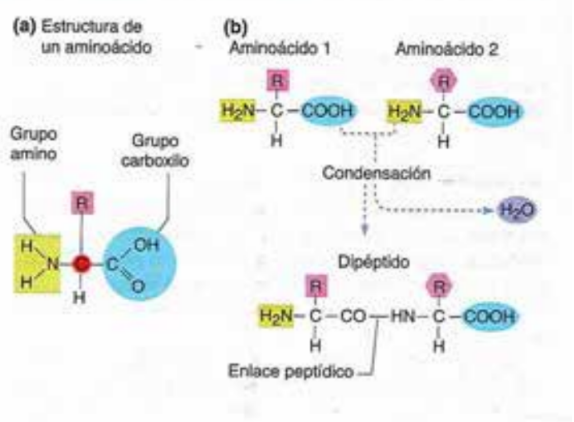


Fig. 1.1: (a) Estructura de los aminoácidos; (b) ejemplo de dipétido. Imagen obtenida de [2].

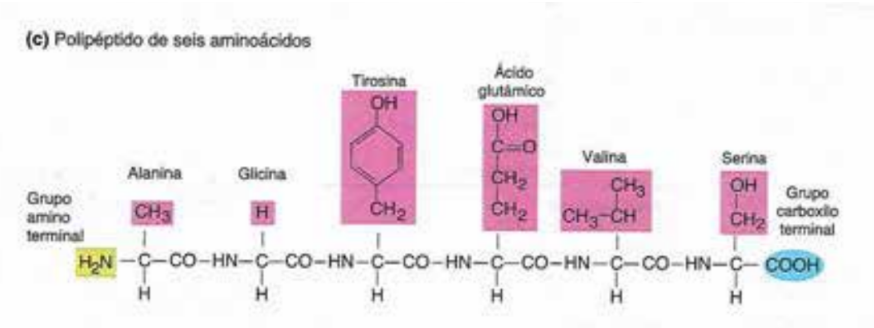


Fig. 1.2: Ejemplo de polipéptido de seis aminoácidos. Imagen obtenida de [2].

Las combinaciones de aminoácidos en las cadenas dan lugar a plegados tridimensionales específicos. En particular, las proteínas pueden ser representadas a distintos niveles de abstracción, en relación a su estructura. En primer lugar, la secuencia de aminoácidos se conoce como *estructura primaria* de la proteína. Luego, la *estructura secundaria* se refiere a una interacción estable entre los aminoácidos, que se dan entre los grupos amino

y carboxilo que tienden a plegar la cadena en una estructura secundaria repetida, como la hélice alfa o la hoja plegada beta. Por otro lado, se tiene la *estructura terciaria* que describe los aspectos de pliegues tridimensionales de un polipéptido, dados por las interacciones entre los grupos R de los aminoácidos, que a menudo suelen ser de forma globular e intrincada. Por último, se tiene la *estructura cuaternaria* que se da cuando una proteína tiene dos o más subunidades de polipéptidos que pueden actuar recíprocamente y formar una estructura cuaternaria.
En la figura 1.3 podemos ver un esquema gráfico de las estructuras primaria, secundaria, terciaria y cuaternaria de las proteínas.

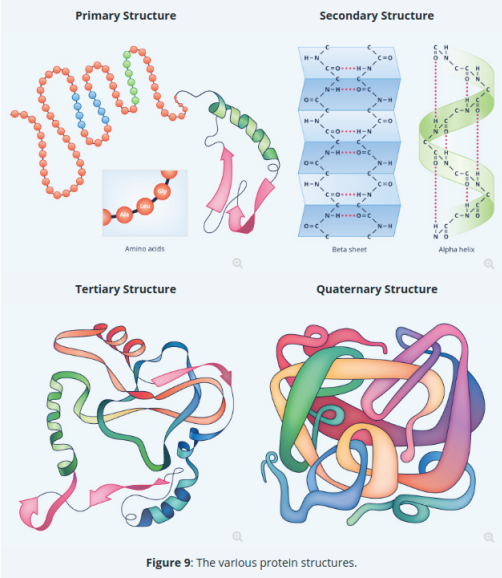


Fig. 1.3: Esquema de los diferentes niveles de abstracción para la representación de proteínas. Imagen obtenida de [3]

El orden en que se disponen los aminoácidos en la estructura primaria de una proteína es el que determina, entre otros factores, la conformación estructural tridimensional que finalmente toma y en consecuencia, la función que cumple. A modo de ejemplo, en el caso de la hemoglobina, a consecuencia de la disposición de los aminoácidos, su estructura final permite adosar moléculas de oxígeno y transportarlas. Una manera de representar la estructura primaria de las proteínas es a través de una secuencia de caracteres, donde cada uno de estos representa un aminoácido. Volviendo al ejemplo de la figura 1.2, la secuencia de los 6 aminoácidos Alanina (A), Glicina (G), Tirosina (E), Ácido glutámico (T), Valina (V), y Serina (S), puede ser representada simplemente por la cadena AGETVS [4, 5].

En la naturaleza, existen ciertas proteínas que presentan patrones estructurales recurrentes en su estructura primaria. Estas macromoléculas pueden ser clasificadas de acuerdo al largo de la mínima unidad de repetición que las constituyen. Aquellas para las cuales sus patrones de repetición son cortos (menores o iguales a 5 aminoácidos), se denominan *fibrilares*; para cuyas repeticiones se componen de 6 a 60 aminoácidos se denominan *repetitivas*; y, finalmente las restantes, se denominan *globulares*[6, 7].

En sus trabajos [8, 9], Turjanski et al. buscan y analizan ocurrencias repetidas de

secuencias en diferentes familias de proteínas. Para ello, basan sus estudios en repeticiones maximales: *Dada una secuencia s, un* ***MR*** *(**M**aximal **R**epeat, por sus siglas en inglés) es una subsecuencia que ocurre más de una vez en s, y cada una de sus extensiones (a derecha o izquierda) ocurre menos veces* [10]. A modo de ejemplo, sea la cadena $s = ARCDFARCDG$. Si bien la subcadena ARC se repite, no es maximal, ya que la extensión a derecha $ARCD$ se repite la misma cantidad de veces en s (2 ocurrencias). Por otro lado, la subcadena $ARCD$ es una repetición maximal ya que ambas extensiones (a derecha e izquierda, siempre que es posible) disminuyen la cantidad de sus ocurrencias en s.

En [8], el grupo describe que largas cadenas de repeticiones exactas son infrecuentes en proteínas particulares, incluso para aquellas que se conoce que se pliegan en estructuras de motivos estructurales recurrentes. Por otro lado, también detallan que estas proteínas con repeticiones naturales son efectivamente repetitivas dentro de sus familias, exhibiendo cadenas de aminoácidos que son repeticiones exactas dentro de la familia de referencia. Como producto de esto, proponen un algoritmo para cuantificar las chances de que una proteína particular pertenezca a una cierta familia. En [11], muestran que los MR de cadenas únicas pueden ser separados en tres clases disjuntas, según su estructura de anidamiento.

- **SMR** (Repetición súper maximal). Secuencia que ocurre más de una vez en s, mientras que no ocurren en s ninguna de sus posibles extensiones.
- **NE** (Repetición maximal anidada). Secuencia para la cual, todas sus ocurrencias, están contenidas en repeticiones maximales de mayor longitud.
- **NN** (Repetición maximal no-anidada). Secuencia para la cual, alguna de sus ocurrencias, no está contenida en alguna repetición maximal de mayor longitud y no es un SMR.

En la Figura 1.4 se puede observar un ejemplo con los distintos ejemplos de MR.

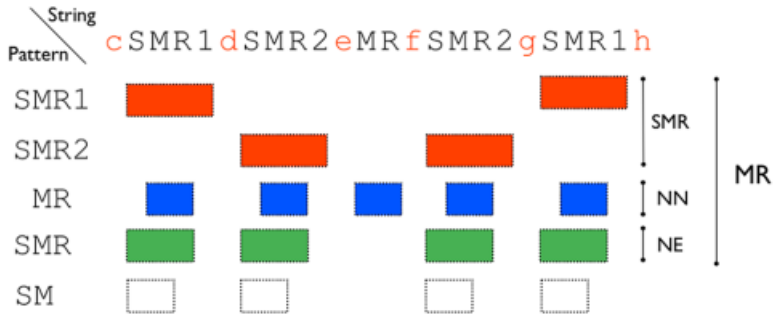


Fig. 1.4: Ejemplo con los distintos tipos de repeticiones maximales. Imagen obtenida de [9].

En [9] extienden este resultado para múltiples cadenas. Luego señalan que repeticiones cortas alcanzan para diferenciar entre familias de proteínas naturales y aquellas agrupadas o generadas al azar.

Ambos trabajos [8, 9] dan el puntapié inicial a la pregunta que da origen al presente trabajo: ¿Es posible hallar una función de distancia entre familias de proteínas, basada en los prefijos de sus repeticiones maximales (MR o alguna de sus clases), que permita:

a) discriminar entre familias naturales de las generadas/agrupadas al azar; y b) dentro de las familias naturales, puede dicha función agrupar las familias repetitivas por un lado y las globulares por otro?

1.2. Motivación

Hoy en día existen métodos para la síntesis de proteínas en laboratorio a través de la determinación de su secuencia de aminoácidos. Los mismos son costosos y se sabe que una parte muy minoritaria de las secuencias sintetizadas se comportan efectivamente como proteínas.

Sigue siendo un hecho intrigante que la mayoría de las cadenas de aminoácidos naturales parecen indistinguibles de las aleatorias según muchas pruebas estadísticas [12], y en particular que, la mayoría de las cadenas polipeptídicas sintetizadas, no se comportan como las proteínas; no se pliegan a estructuras específicas ni funcionan en un contexto celular. Así, la reducción en la descripción de proteínas a secuencias lineales de aminoácidos individuales pierde un aspecto fundamental para explicar la biofísica del plegado y función de la proteína [13, 14].

La búsqueda de "códigos estructurales" en el análisis de secuencias de proteínas debe considerar la ocurrencia de correlaciones en los patrones de grupos de aminoácidos, una tarea que resulta combinatorialmente prohibitiva a la hora de analizar exhaustivamente todas las secuencias de proteínas [15].

Múltiples heurísticas diseñadas para analizar la correlación de los patrones de aminoácidos en las proteínas han llevado a agrupar de manera aproximada ciertos conjuntos de secuencias en familias [16], conjuntos estructurales de glóbulos plegados en estas familias [17] e incluso insinuar la conexión entre la termodinámica de plegamiento [18] y la evolución de las proteínas naturales [19]. Sin embargo, la mayoría de estos métodos requieren que se alineen múltiples secuencias entre sí, a través de una matriz común, en una denominada alineación múltiple de secuencias. Dicho proceso sigue siendo un problema matemático abierto y, por lo tanto, los cálculos actuales de las alineaciones inferidas deben ser curados manualmente por expertos humanos [20]. Es por eso que tener un método matemático cerrado, que sea una métrica y permita clasificar familias de proteínas, puede dar lugar a una forma de poder determinar si dada una secuencia de aminoácidos va a corresponder a una proteína o va a ser parte de cierta familia. Esto permitiría contar con una mayor probabilidad de que la secuencia elegida para la experimentación en laboratorio sea efectiva.

1.3. Objetivos

El objetivo general de la tesis es hacer una contribución a la comprensión de las propiedades estructurales de las proteínas, a partir de los datos obtenidos a nivel de su estructura primaria.

El objetivo específico es poder proponer una función de distancia entre familias de proteínas, en base a los prefijos de sus repeticiones maximales, que permita: a) discriminar entre familias naturales y las generadas/agrupadas al azar; y b) dentro de las familias naturales, nos permita agrupar las familias repetitivas por un lado y las globulares por otro.

1.4. Organización de este trabajo

En lo que sigue del trabajo se analizará una estructura computacional para contener las repeticiones maximales, y con ella se estudiará la posibilidad de comparar dos estructuras contenedoras de repeticiones maximales de distintas familias. Primero se analizará una alternativa traduciendo esta estructura a otra equivalente (utilizada para redes de computadoras), buscando distintos algoritmos existentes para la comparación entre dos redes de computadoras.

Luego, se propondrán y analizarán distintas funciones de distancia para la clasificación de familias de proteínas, en base a la estructura presentada previamente.

El trabajo finalizará con un capítulo de conclusiones y líneas futuras de investigación.

2. DESARROLLO

En esta sección, intentaremos responder la pregunta que guía el presente trabajo: **¿en cuánto se asemejan dos familias de proteínas?** Para ello, nos basaremos en los patrones maximales de repetición de la estructura primaria de las proteínas. El esquema de trabajo propuesto consiste en seguir los siguientes pasos:

- **Paso 1:** Seleccionar 2 familias de proteínas (familia A y B).
- **Paso 2:** Obtener el conjunto de patrones maximales de repetición en base a la estructura primaria de las proteínas de la familia A. Hacer lo mismo para la familia B.
- **Paso 3:** Cuantificar cuán semejantes son ambos conjuntos.

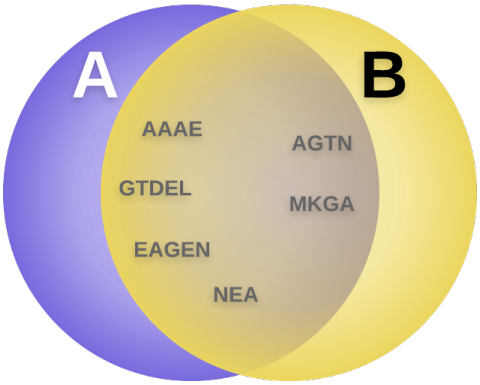
A continuación mostramos un ejemplo simplificado. Como **Paso 1**, seleccionamos dos familias de proteínas (las denominamos A y B). Luego, como **Paso 2**, obtenemos el conjunto de patrones maximales de repetición de cada una de estas familias. Los conjuntos se pueden ver en la Tabla 2.1.

Conjunto de patrones maximales de repetición	
Familia A	Familia B
AAAE	AGTN
AGTN	CTSP
EAGEN	DGRVA
GTDEL	MHEL
MKGA	MKGA
NEA	RMLL

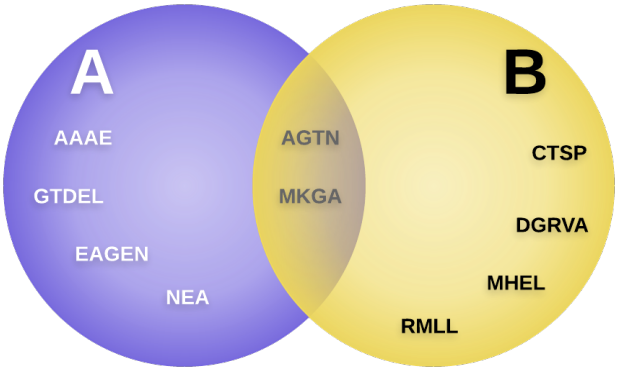
Tab. 2.1: Conjuntos de patrones maximales de repetición de las familias A y B. En azul los patrones que difieren entre ambas familias.

En la Figura 2.1b se puede observar una representación gráfica de ambos conjuntos. Es fácil notar que los conjuntos no son idénticos, ya que hay elementos que pertenecen a un conjunto y no al otro (Ej. el patrón AAAE pertenece al conjunto A y no pertenece al B), los mismos se destacan en la tabla en color azul. Tampoco son totalmente distintos ya que hay elementos que son comunes a ambos conjuntos (Ej. el patrón MKGA pertenece tanto a A como a B). Por lo tanto, sólo nos queda cuantificar cuál es el grado de semejanza entre ambos conjuntos, en base a la cantidad de elementos que comparten y los que no.

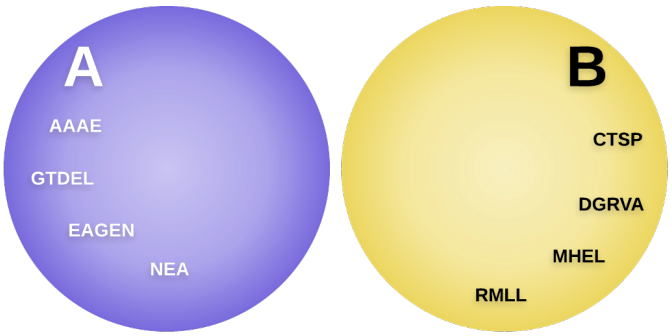
En la Figura 2.1 se pueden observar tres ejemplos distintos de familias de proteínas, donde sus conjuntos de patrones maximales de repetición son (a) exactamente iguales, (b) semejantes y (c) totalmente distintos.



(a) $A = B$



(b) $A \sim B$



(c) $A \neq B$

Fig. 2.1: Tres ejemplos distintos de familias de proteínas donde sus conjuntos de patrones maximales de repetición son (a) exactamente iguales, (b) semejantes y (c) totalmente distintos.

Finalmente, como **Paso 3**, debemos cuantificar el grado de semejanza (o de diferencia) que existe entre los conjuntos A y B. Para ello, definimos la función *dist*, que a partir de los elementos de ambos conjuntos nos devuelve un valor entre 0 y 1. En el caso de que ambos conjuntos sean exactamente iguales (Figura 2.1a) $dist(A, B)$ retornará 0. En caso de ser totalmente disjuntos (Figura 2.1c) $dist(A, B)$ retornará 1. En el caso de que compartan sólo algunos elementos $dist(A, B)$ deberá devolver un valor que cumpla $0 < dist(A, B) < 1$. Si además se logra que dicha función cumpla con las propiedades de identidad, simetría,

desigualdad triangular y ser siempre positiva (ver Tabla 2.2), se estaría en presencia de una métrica. La ventaja de contar con una métrica es que restringe más a la función de distancia, es decir, no cualquier función de distancia es una métrica ya que, por ejemplo, la distancia debe cumplir la desigualdad triangular. Utilizar una métrica también provee de un marco de referencia para poder dar respuesta a la pregunta planteada, cuantificando la semejanza entre varios conjuntos.

Propiedad	**Definición**
Identidad	$dist(A, B) = 0 \Leftrightarrow A = B$
Simétrica	$dist(A, B) = dist(B, A)$
Desigualdad triangular	$dist(A, C) \leq dist(A, B) + dist(B, C)$
La distancia debe ser positiva	$0 \leq dist(A, B)$

Tab. 2.2: Propiedades que debe cumplir la función *dist* para ser una métrica

2.1. Estructura de Datos contenedora de patrones

Al analizar los elementos de los conjuntos de repeticiones maximales de una misma familia de proteínas se puede observar que los patrones no son necesariamente disjuntos entre sí. Es decir, hay casos en que los patrones comparten parte de los elementos de sus cadenas (Ej. los patrones AHRR y AHLK comparten el prefijo AH). Así, se pueden encontrar algunos casos como los siguientes:

1. **Prefijos compartidos**: En el caso de **AK**QLN y **AK**RE, comparten **AK** como prefijo.

2. **Sufijos compartidos**: En el caso de QLN**AK** y RE**AK**, comparten **AK** como sufijo.

3. **Subcadena compartida**: En el caso de QL**AK**N y R**AK**E, comparten **AK** como subcadena.

Una manera de representar los patrones que comparten prefijos es a través de una estructura de árbol denominada Trie[21], donde cada nodo corresponde a un caracter de la secuencia a representar. Si dos secuencias comparten un prefijo, permite dar cuenta de esta situación. En la Figura 2.2 se puede observar un ejemplo de una estructura Trie en base a los patrones *BAÑO, BABOR, BEBE, BOLSA, BOMBA, SALA, SALUD y SEÑA*. En dicho ejemplo las palabras *BAÑO* y *BABOR* comparten el prefijo *BA*, por lo tanto comparten dichos nodos. Lo mismo ocurre con otros patrones.

En la Figura 2.3 se muestra otro ejemplo de un Trie. En este caso se corresponde con nuestro conjunto de repeticiones maximales de la Familia A (ver Tabla 2.1). Observar que dos de sus patrones (**A**AAE, **A**GTN) comparten el prefijo **A**, por lo tanto, tienen el primer nodo **A** como nodo común.

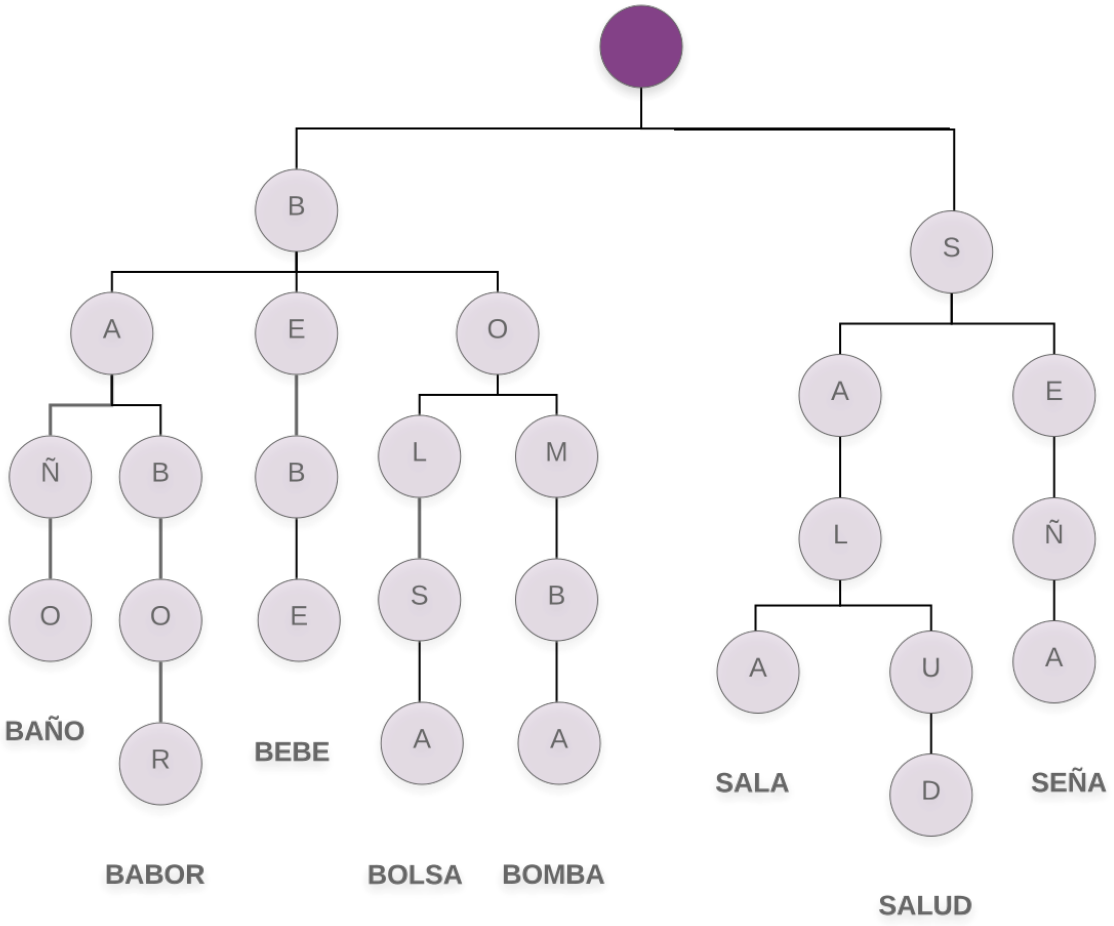


Fig. 2.2: Ejemplo de una estructura Trie en base a los patrones BAÑO, BABOR, BEBE, BOLSA, BOMBA, SALA, SALUD y SEÑA.

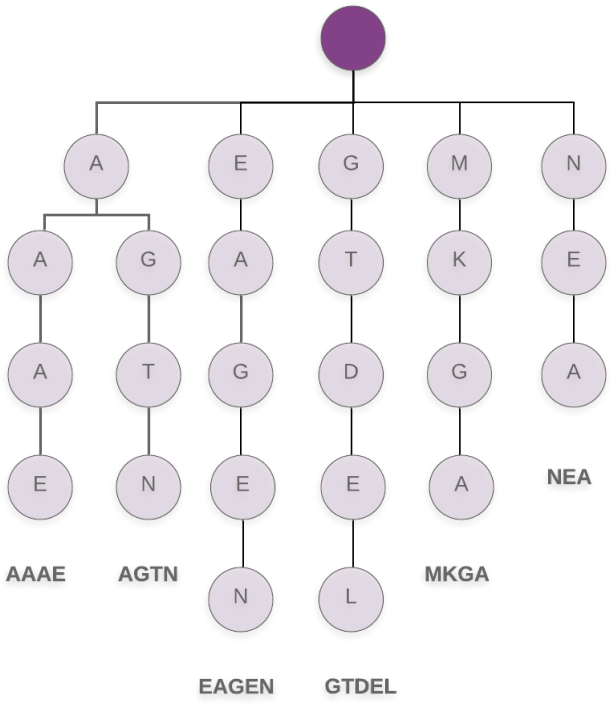


Fig. 2.3: Ejemplo de Trie en base al conjuntos de patrones maximales de repetición de la Familia A (ver Tabla 2.1).

Un problema que surge de esta manera de representar los patrones es que se pierde la información correspondiente a patrones que son prefijos de otros. A modo de ejemplo, si incorporamos un nuevo patrón, por ejemplo el AAA, obtendríamos el mismo Trie de la Figura 2.3 ya que **AAA** es prefijo de **AAA**E.

Una propuesta para solucionar dicho problema es explicitar, en cada nodo, si corresponde al caracter final de un patrón o no. En la Figura 2.4, se muestra un ejemplo donde se implementa dicha solución. Observar que el Trie, además de los patrones originales de la Tabla 2.3, contiene los patrones AAA, EA, GTD y MK. Es importante notar que de no implementar dicha solución, con estos mismos patrones, hubiésemos obtenido el mismo Trie de la Figura 2.3.

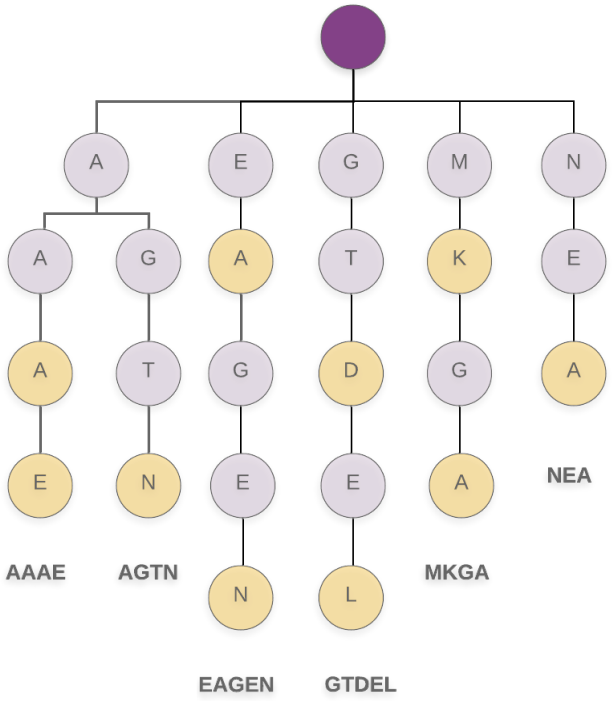


Fig. 2.4: Ejemplo de Trie explicitando, en cada nodo, si corresponde al caracter final de un patrón, para los patrones originales de la Tabla 2.3 (AAAE, AGTN, EAGEN, GTDEL, MKGA y NEA) y los patrones adicionales AAA, EA, GTD y MK. En amarillo se marcan los nodos terminales.

En el caso de querer representar la información de cuántas veces ocurre un mismo patrón dentro de una familia, nuevamente deberíamos readaptar la estructura de Trie. Una solución posible es agregar en cada nodo la cantidad de instancias que posee el patrón hasta dicho nodo. A partir de los patrones de la Tabla 2.3 construimos un nuevo Trie conteniendo dicha información (ver Figura 2.5). Notar que si quisiéramos sólo representar los patrones y no la cantidad de sus ocurrencias, podríamos hacerlo con el Trie anterior (Figura 2.4). Cabe destacar que en este modelo de Trie no es necesario destacar el nodo de finalización de un patrón, ya que el mismo se puede identificar con el valor que representa la cantidad de repeticiones.

Patrón	Ocurrencias
AAA	2
AAAE	1
AGTN	1
EA	1
EAGEN	1
GTD	1
GTDEL	1
MK	1
MKGA	1
NEA	1

Tab. 2.3: Conjuntos de patrones maximales de repetición de una familia y sus ocurrencias dentro de la misma.

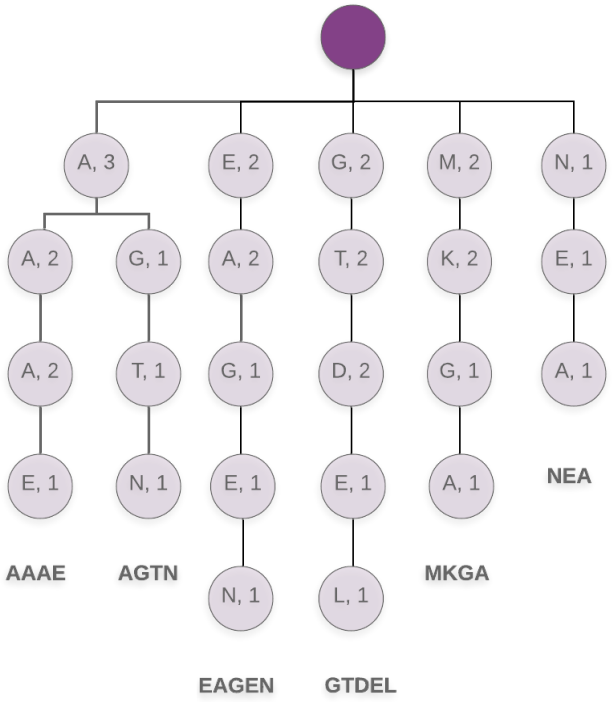
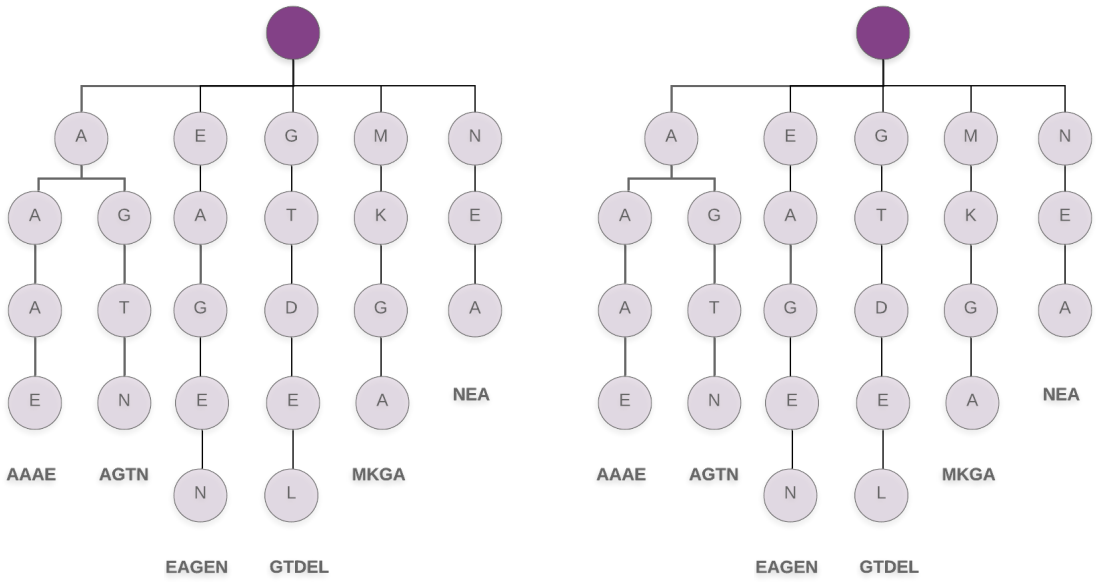


Fig. 2.5: Ejemplo de Trie, explicitando en cada nodo la cantidad de instancias que posee el patrón hasta dicho nodo, para los patrones y sus instancias de la Tabla 2.3

La estructura de Trie es bastante eficiente para insertar y buscar patrones. Para insertar un patrón, la complejidad computacional es $O(N)$, donde N es la longitud del patrón. Para buscar, su complejidad computacional es $O(MN)$ siendo M la cantidad de letras distintas posibles en cada nivel del árbol. En el caso de nuestras proteínas, el valor de M está acotado a 20 (cantidad de aminoácidos distintos posibles). Notar que, al ser M una constante, se puede reducir el orden a $O(N)$.

Considerando entonces la estructura de Trie para contener los patrones de una familia de proteínas, nuestra nueva hipótesis de trabajo es que **si dos familias de proteínas son semejantes, entonces sus Trie son semejantes**. Por simplicidad, por ahora tomemos la forma más simple de Trie que vimos (Ej. Figura 2.3) y veamos cómo funciona esta nueva hipótesis con respecto a los conjuntos de patrones de nuestra figura inicial, la Figura 2.1.

En la Figura 2.6 se pueden observar los Trie construidos en base a los patrones de la Figura 2.1a. Notar que ambos Trie son exactamente iguales.

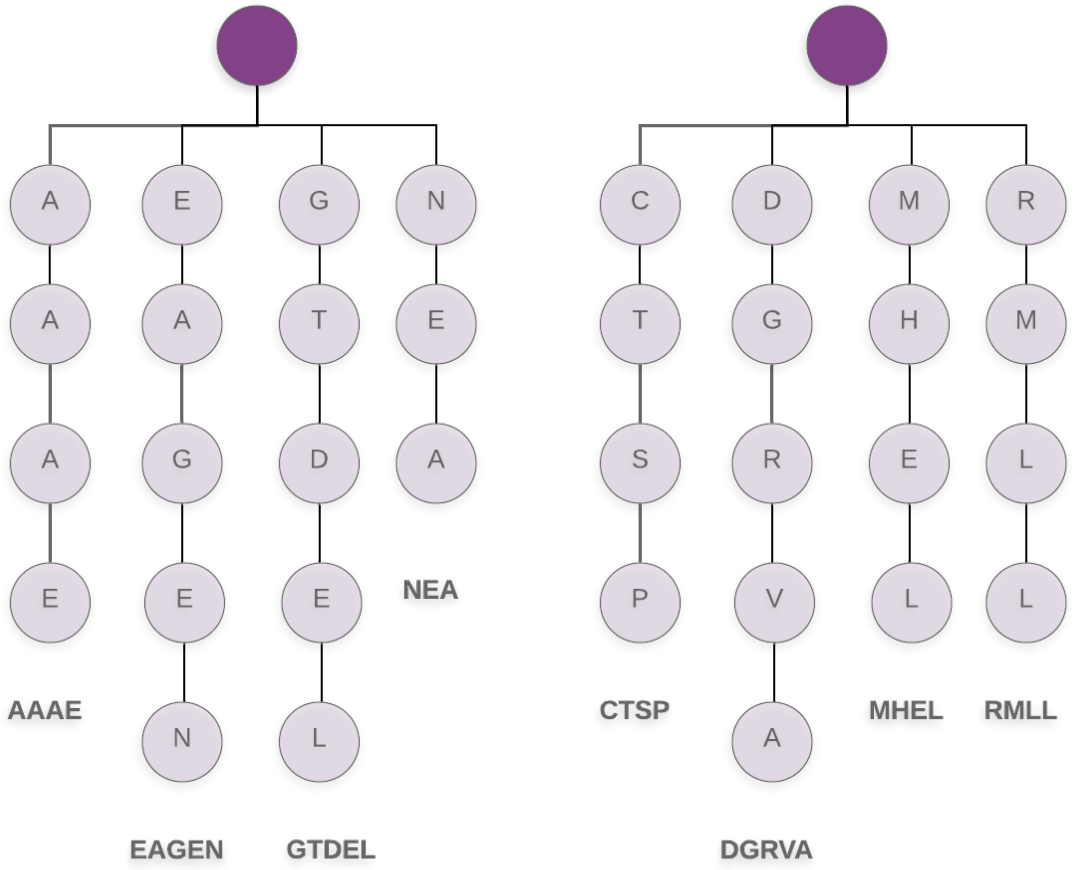


(a) Trie A *(b)* Trie B

Fig. 2.6: Trie construidos en base a los patrones de la Figura 2.1a. Ambos Trie son iguales.

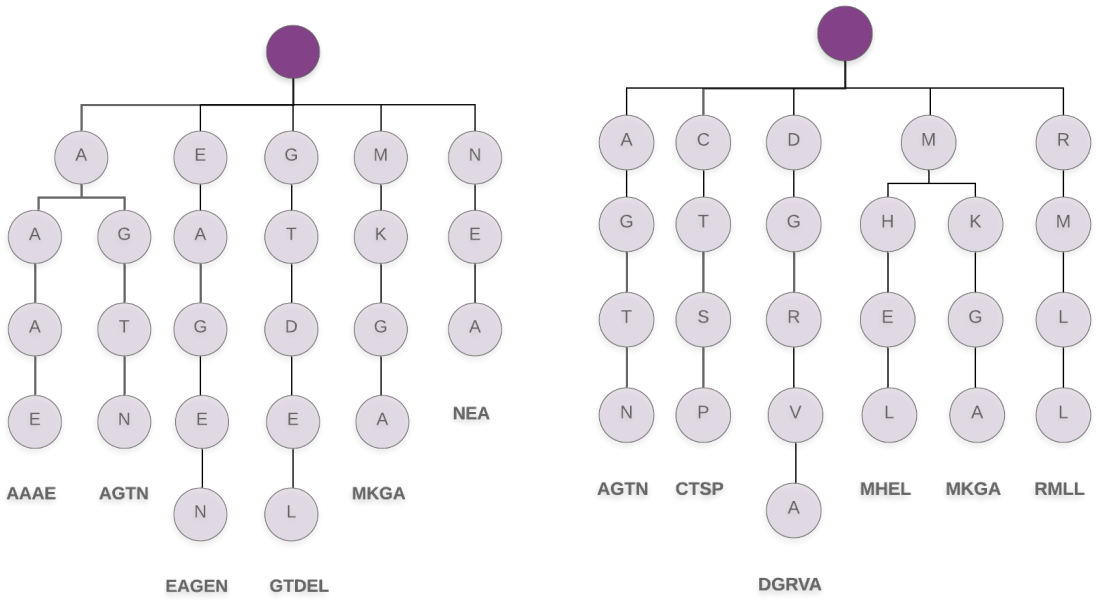
En la Figura 2.7 se pueden observar los Trie construidos en base a los patrones de la Figura 2.1c. Notar que ambos Trie son visualmente totalmente distintos.

En la Figura 2.8 se pueden observar los Trie construidos en base a los patrones de la Figura 2.1b. Notar que ambos Trie tienen parte de su estructura en común, como las ramas correspondientes a los patrones $\{AGTN, MKGA\}$.



(a) Trie A *(b)* Trie B

Fig. 2.7: Trie construidos en base a los patrones de la Figura 2.1c. De manera visual se puede observar que ambos Trie son totalmente distintos.



(a) Trie A *(b)* Trie B

Fig. 2.8: Trie construidos en base a los patrones de la Figura 2.1b. Ambos Trie comparten algunas ramas del Trie, como las ramas correspondientes a los patrones $\{AGTN, MKGA\}$.

Como pudimos observar en los tres ejemplos (Figuras 2.1, 2.6, 2.7 y 2.8), la similitud de los **conjuntos** de patrones está relacionada con la semejanza de los **Trie**. Es decir, **cuando los Trie son semejantes los conjuntos se asemejan**. Lo mismo ocurre en sentido inverso.

2.2. Algoritmos para el análisis de redes de computadoras

En el campo de la Teoría de las Comunicaciones se estudia, entre otros temas, las redes de comunicación entre computadoras. Dentro de esta área del conocimiento es de sumo interés poder analizar las características de las redes (topología, rendimiento, etc.). Así, surge de forma natural, la necesidad de poder comparar la similitud entre distintos tipos de redes. La hipótesis de trabajo es que, **si dos redes de computadoras son similares entonces sus grafos también lo son.**

En la sección anterior, utilizamos la estructura de Trie [] para representar las repeticiones maximales de una familia. Una estructura Trie es un árbol, es decir, un grafo que no contiene ciclos. Así, **si dos Trie son similares entonces sus grafos también lo son.**

Decidimos explorar herramientas existentes en el área de Teoría de las Comunicaciones para ver si, adaptando cada uno de nuestros Trie a una red, podíamos analizar las semejanzas entre ellos.

A continuación se muestran ejemplos de redes de computadoras, y sus respectivas representaciones en grafos, y por otro lado ejemplos de Trie con sus respectivas representaciones en grafos.

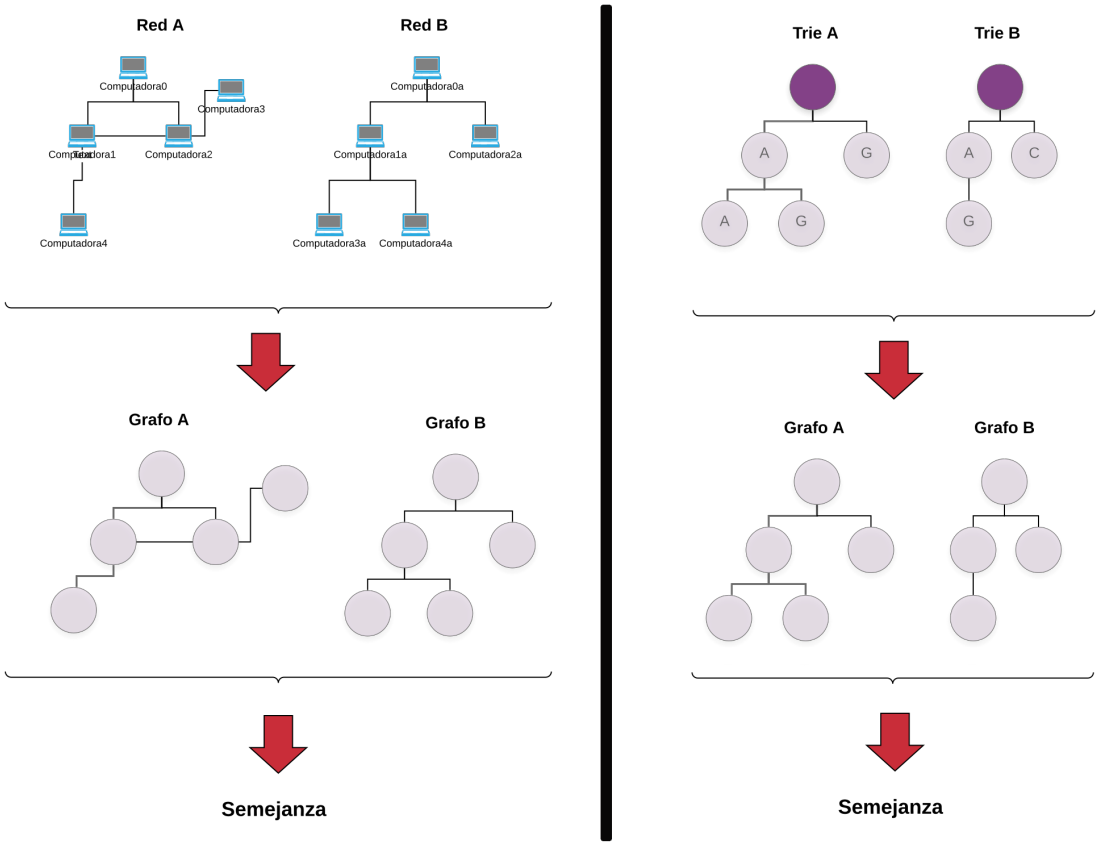


Fig. 2.9: Esquema de comparación de redes de computadoras y Tries.

Como se puede ver en el ejemplo de la Figura 2.9, una red de computadoras se puede modelar utilizando grafos. Lo mismo ocurre con los Trie. Una diferencia que se observa entre ambas representaciones es que, los grafos de las redes de computadoras, pueden contener ciclos mientras que los grafos de los Trie no. Esto se debe a que los Trie se modelan mediante árboles, conocidos también como grafos sin ciclos, es decir, un caso particular de los grafos. Notar que el pasaje de una red de computadoras o un Trie a un grafo se puede realizar de manera mecánica y directa.

Finalmente, en la figura 2.9 se puede observar que si dos redes de computadoras son semejantes entonces sus grafos son semejantes. Lo mismo ocurre con los Trie.

Así, para cuantificar cuán semejantes son dos Trie intentaremos transitar el camino de convertirlos a redes de computadoras y utilizar algún método existente para cuantificar la semejanza entre dichas redes. En resumen, nuestro esquema de trabajo a seguir será:

1. **Paso 1**: Seleccionar 2 familias de proteínas (familia A y B).

2. **Paso 2**: Obtener el conjunto de patrones maximales de repetición en base a la estructura primaria de las proteínas de la familia A. Hacer lo mismo para la familia B.

3. **Paso 3**: Convertir ambos conjuntos a Trie.

4. **Paso 4**: Convertir ambos Trie a redes de computadoras.

5. **Paso 5**: Ejecutar algún programa existente que cuantifique semejanzas entre redes, utilizando como entradas las redes del paso anterior.

En la Figura 2.10 se puede observar el esquema de trabajo descripto.

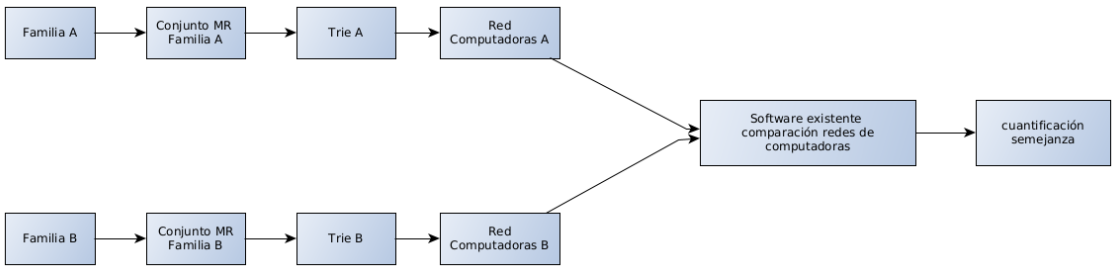


Fig. 2.10: Esquema de trabajo para computar el valor de diferencia entre dos familias de proteínas, utilizando software existente de comparación de redes de computadoras.

A continuación describiremos los distintos métodos existentes explorados.

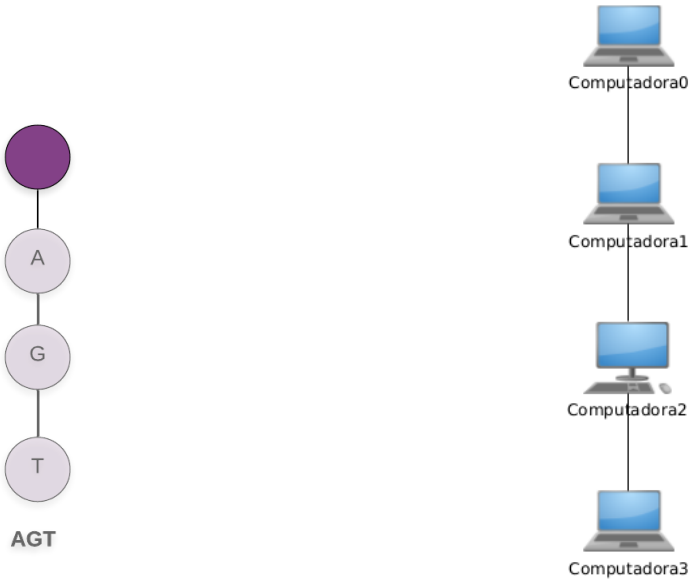
Si bien existe una gran variedad de programas para el análisis y comparación de redes de computadoras, nos centraremos en tres de ellos: *Deltacon*, *Gmatch4pyeg* y *Networkx*. En lo que sigue, describiremos a cada una de estas herramientas.

2.2.1. Deltacon

Deltacon [22] fue la primer herramienta que analizamos para la comparación de redes. En su trabajo Koutra et al. introducen una serie de axiomas y propiedades deseadas de una función de similaridad entre grafos, y proponen un algoritmo que evalúa la semejanza entre dos grafos que cumplen los axiomas formulados. Para probar el algoritmo utilizamos el código disponible en el repositorio github Deltacon[1].

Para la generación de las entradas desarrollamos un script que convierte un Trie de una familia de proteínas a una entrada válida de la herramienta. El script traduce cada nodo del Trie a un nodo de la red. Adicionalmente, por cada arista del Trie, genera una arista en la red. Dado que Deltacon requiere que las aristas tengan peso, se les asignó a todas ellas el valor constante 1 ya que, de momento, no nos interesa utilizar dicho valor. En la Figura 2.11 se puede observar cómo se traduce un Trie de 3 nodos (Figura 2.11a) a una red de 3 computadoras conectadas entre sí por aristas de peso 1 (Figura 2.11b). El formato del archivo generado para el ejemplo se puede observar en la Figura 2.12. Cada fila representa una arista. La primer columna corresponde al nodo origen, la segunda columna al nodo destino y la tercera a su peso. Un detalle importante es que las computadoras (nodos) se representan con valores numéricos del dominio de los enteros.

[1] Deltacon: `https://github.com/ZhenguoChen/DeltaCon/blob/master/DeltaCon.py`



(a) Trie *(b)* Red de computadoras

Fig. 2.11: Ejemplo de (a) un Trie y (b) su red de computadoras equivalente.

```
0 1 1
1 2 1
2 3 1
```

Fig. 2.12: Archivo generado como entrada para Deltacon a partir del Trie de la Figura 2.11

Dado que una red "equivalente" a un Trie asociado a una familia de proteínas debería poseer más de 10.000 computadoras, nuestro primer paso fue evaluar la escalabilidad de la herramienta ya que, si la misma no permite entradas de dimensiones similares, realmente no nos va a ser de utilidad. Así, probamos distintos ejemplos de Trie con pocos nodos para luego ir incrementándolos en cantidad y, finalmente, poder evaluar hasta qué punto escala la herramienta. En una primer aproximación realizamos una comparación entre dos redes exactamente iguales, es decir, generamos una red de 1.000 nodos (denominada G1) y a Deltacon le suministramos como entrada el par (G1, G1). Esto nos permitió, también, chequear el valor de salida para este caso ya que, las dos redes de entrada, eran iguales entre sí.
En la Figura 2.13 se puede observar un análisis del tiempo de ejecución de la herramienta en función de la cantidad de computadoras de la red suministrada como entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. La cantidad de nodos del eje x expresa la cantidad de computadoras de un solo elemento del par utilizado como entrada. En el gráfico se muestran con puntos azules los tiempos de ejecución obtenidos con Deltacon, y con una línea roja la función teórica utilizada para aproximarlos a través de una curva. Se observa que para unos 1.000 nodos la herramienta se demora unos 256 segundos en ejecutar, pero luego para 2.000 nodos ya se demora unos 1.019 segundos. La función de ajuste (línea roja) ajusta muy bien a los valores de una

función cuadrática (aX^2), con $a = 0{,}00025396$ y un valor de coeficiente de determinación R^2 de la predicción de $R^2 = 0{,}9999811670877109$ de la función de regresión lineal de scikitlearn[2]. En base a esta función de ajuste, se puede estimar que Deltacon demorará aproximadamente 8 años para una entrada de 1.000.000 nodos (tamano aproximado de las redes que deseamos utilizar como entrada), lo que no está dentro del rango de tiempos que esperamos para nuestro trabajo.

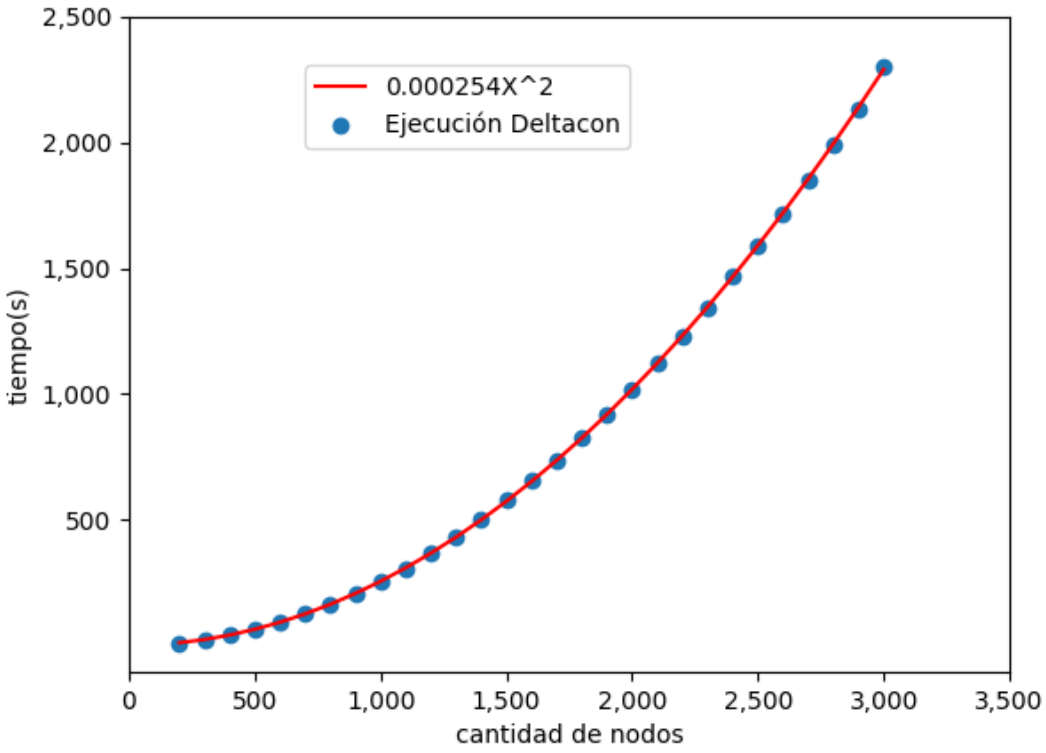


Fig. 2.13: Tiempo de ejecución de la herramienta Deltacon en función de la cantidad de nodos de uno solo de los elementos del par de entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. Para cada valor del eje x se utilizó una sola red de ejemplo.

Un inconveniente adicional que encontramos en esta herramienta es que requiere que las redes a comparar posean la misma cantidad de computadoras, algo que no podemos garantizar entre dos Tries de distintas familias. Un análisis exploratorio nos permite afirmar que no parece ser viable adaptar el algoritmo para que funcione con redes de distinto tamaño.

Debido a todo lo expuesto anteriormente, decidimos descartar esta herramienta. El script de traducción y las pruebas realizadas se encuentran disponibles en el repositorio de código[3]

[2] Deltacon: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

[3] repositorio de código: `https://gitlab.com/agusciraco/tesis-grado`

2.2.2. Gmatch4pyeg

Gmatch4pyeg[4] fue la segunda herramienta que analizamos. La misma utiliza la biblioteca *Networkx*[5] para representar los grafos a comparar. El algoritmo utilizado por *Gmatch4pyeg* define una métrica de similaridad entre redes utilizando la métrica **distancia de edición**, tomando como operaciones de edición a) agregar y eliminar computadoras, y b) agregar, eliminar y modificar conexiones entre computadoras. Luego aplica un algoritmo de optimización de problemas de asignación. A diferencia de *Deltacon*, *Gmatch4pyeg* no requiere que los grafos a comparar posean la misma cantidad de nodos. El algoritmo retorna una matriz de similaridad o distancia entre los grafos de entrada que contabiliza la cantidad de operaciones realizadas para transformar una red en la otra. La matriz de salida es simétrica. Para probar esta herramienta utilizamos las misma entradas generadas para *Deltacon* ya que, esta herramienta, utiliza el mismo formato.

Como primer paso, decidimos evaluar la escalabilidad de la herramienta. En la Figura 2.14 se puede observar un análisis del tiempo de ejecución de la herramienta en función de la cantidad de computadoras de la red suministrada como entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. La cantidad de nodos del eje x expresa la cantidad de computadoras de un solo elemento del par utilizado como entrada. En el gráfico se muestran con puntos azules los valores obtenidos con *Gmatch4pyeg*, y con una línea roja la función teórica utilizada para aproximar los puntos. Se observa que para unos 1.000 nodos la herramienta se demora unos 40 segundos en ejecutar, pero luego para 2.000 nodos ya se demora unos 246 segundos. La función de ajuste (aX^2, línea roja), con $a = 9{,}10716056e-05$ y un valor de coeficiente de determinación R^2 de la predicción de $R^2 = 0{,}9778085199059364$, no ajusta tan bien como para la herramienta anterior. Si bien su predicción es bastante cercana, puede mejorarse utilizando una función cuadrática con más términos, es decir, utilizando un polinomio del estilo $aX^2 + bX + c$. En base a la primer función de ajuste se predice que, para una entrada de 1.000.000 nodos, $GMatch4py$ demorará considerablemente menos que $DeltaCon$, pasando de 8 a 2 años. La salida de la herramienta es una matriz, sin embargo nosotros buscamos un valor unidimensional de semejanza y no uno multidimensional. Exploramos la alternativa de transformar dicha matriz a un valor unidimensional, pero en un análisis preliminar de la documentación no pareció ser factible dado que no se especifica con claridad la matriz de retorno. De esta manera, optamos por descartar el uso de esta herramienta.

[4] Gmatch4pyeg: `https://github.com/Jacobe2169/GMatch4py`
[5] Networkx: `https://networkx.github.io/`

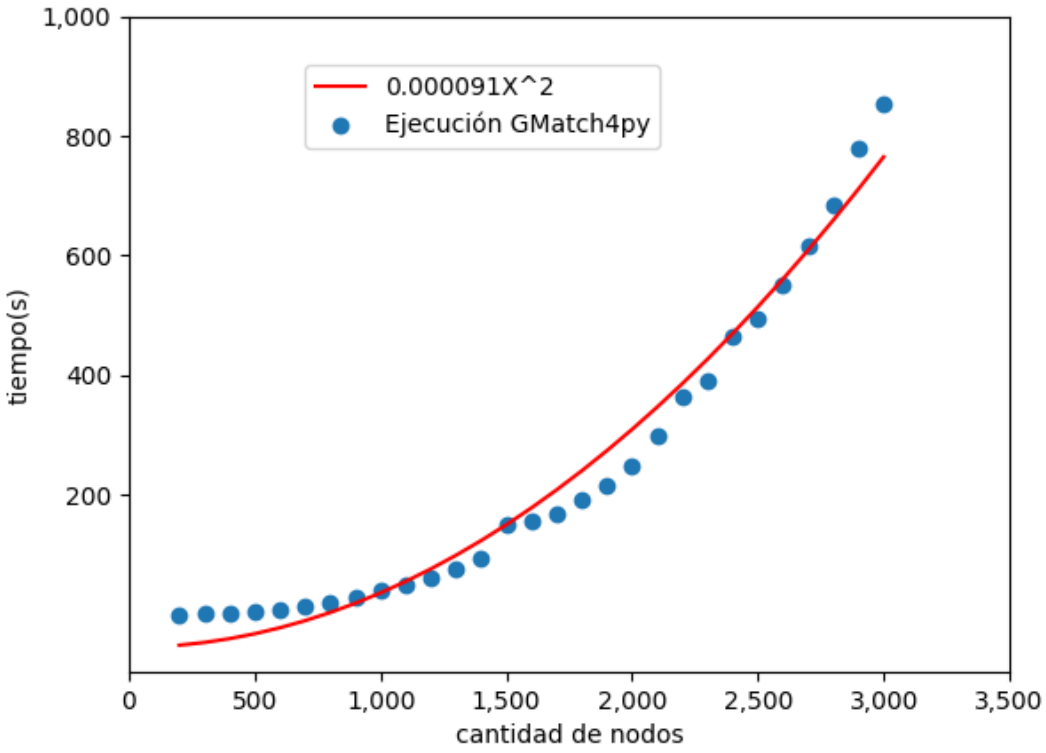


Fig. 2.14: Tiempo de ejecución de la herramienta *GMatch4py* en función de la cantidad de nodos de uno solo de los elementos del par de entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. Para cada valor de eje del eje x se utilizó una sola red de ejemplo.

2.2.3. Networkx

La última alternativa que exploramos fue *Networkx*[6]. Esta herramienta provee una función de distancia entre dos grafos y no restringe la cantidad de nodos que puede tener cada uno de ellos. La misma compara dos grafos utilizando la función de **distancia de edición** y el algoritmo $A*$ para la comparación [23]. $A*$ es un algoritmo para encontrar un camino entre dos nodos del grafo. A diferencia de la herramienta *GMatch4py*, *Networkx* tiene como salida una medida de similaridad undimensional en vez de una matriz.

Nuevamente, como primer paso, decidimos evaluar la escalabilidad de la herramienta. En la Figura 2.15 se puede observar un análisis del tiempo de ejecución de la herramienta en función de la cantidad de computadoras de la red suministrada como entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. La cantidad de nodos del eje x expresa la cantidad de computadoras de un solo elemento del par utilizado como entrada. En el gráfico se muestran con puntos azules los tiempos de ejecución obtenidos con *Networkx*, y con una línea roja la función teórica utilizada para aproximar dichos valores. En este caso, la función teórica (línea roja) ajusta a una función cuadrática $a = 0{,}00036054$ con coeficiente de determinación R^2 de la predicción

[6] Networkx: `https://networkx.github.io/`

de $R^2 = 0{,}9763500258606105$. Se puede observar que para dos redes de 200 computadoras cada una, demora unos pocos segundos (1.8 segundos). Luego el tiempo se va incrementando con el tamaño de la entrada, hasta alcanzar 250 segundos con dos redes de 900 computadoras cada una. Estos tiempos no parecen ser excesivos, sin embargo a medida que aumentamos el tamaño de las redes notamos un impacto llamativo en el consumo de memoria, por ello realizamos un análisis complementario de utilización de memoria, utilizando la biblioteca *memory-profiler*[7]. En la Figura 2.16 se puede observar un gráfico análogo al de la Figura 2.15 pero analizando el consumo de memoria en vez del tiempo de ejecución. En dicho gráfico se puede notar cómo se incrementa el consumo de memoria a medida que aumentan la cantidad de nodos de las redes entrada. En este caso, la función teórica (línea roja) ajusta a una función cuadrática $a = 0{,}01576337$ con coeficiente de determinación R^2 de la predicción de $R^2 = 0{,}9712017862133502$. Si bien se puede mejorar el ajuste, es posible estimar que para una entrada correspondiente a dos redes de 700 computadoras cada una, el consumo de memoria es prácticamente 10 veces más que con 400 computadoras. Así, se estima que para redes de 10.000 nodos va a requerir excesiva memoria (del orden de 1.500 GB), por lo que finalmente también se optó por descartar esta herramienta.

2.2.4. Conclusiones

Después de buscar herramientas existentes en el campo de la Teoría de las Comunicaciones, y luego de estudiar tres de ellas en profundidad (*Deltacon*, *Gmatch4pyeg* y *Networkx*), decidimos descartar este camino ya que ninguna de las herramientas halladas nos permiten responder nuestra pregunta. En el caso de *DeltaCon*, demora mucho tiempo en ejecutar el volumen de nodos que tienen nuestros Tries y, además, requiere que ambas estructuras posean la misma cantidad de nodos. En el caso de *GMatch4py* la salida es multidimensional y, para nuestra pregunta, necesitamos una respuesta unidimensional. Por último, en el caso de *Networkx*, el consumo de memoria es excesivo para el volumen de nodos de nuestras familias.

Sabemos que la pregunta más general sobre cuán parecidos son dos grafos es una pregunta difícil de responder en tiempos razonables, pues es una pregunta que se puede traducir fácilmente a si dos grafos son isomorfos y, sin restricciones, es un problema considerado difícil o NP [24].

Finalmente, optamos por intentar responder nuestra pregunta por otra vía que detallaremos en las próximas secciones.

2.3. Cantidad de diferencias entre Tries

A partir de los resultados negativos de la sección anterior, decidimos retomar la pregunta de cuán parecidos son dos Tries entre sí, intentando responderla en base a una función de distancia propuesta por nosotros mismos. Para ello, nos basamos en la siguiente premisa:

«Sean los Tries A y B. Un nodo a de A y un nodo b de B son iguales si ambos representan el mismo caracter, y todos sus ancestros también son iguales. Es decir, son iguales si

[7] memory-profiler: `https://pypi.org/project/memory-profiler/`

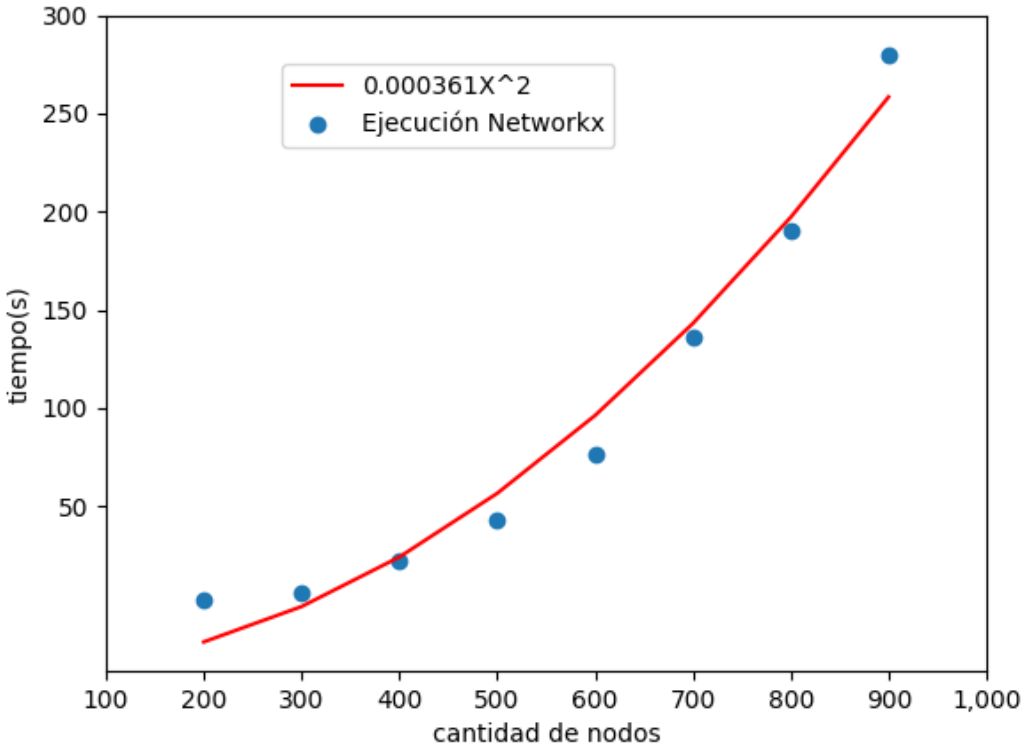


Fig. 2.15: Tiempo de ejecución de la herramienta *Networkx* en función de la cantidad de nodos de uno solo de los elementos del par de entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. Para cada valor de eje del eje x se utilizó una sola red de ejemplo.

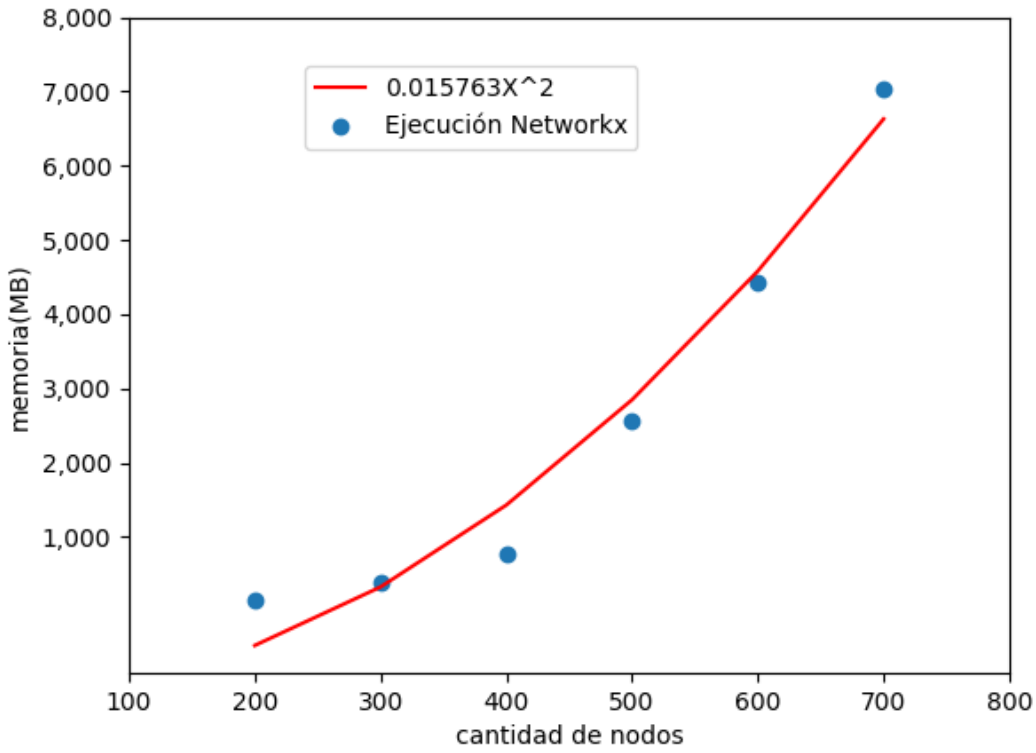


Fig. 2.16: Consumo de memoria de la herramienta *Networkx* en función de la cantidad de nodos de uno solo de los elementos del par de entrada. El par de entrada utilizado siempre fue conformado por dos redes exactamente iguales. Para cada valor de eje del eje x se utilizó una sola red de ejemplo.

ambas ramas del Trie A y B, que finalizan en los nodos a y b, representan al mismo patrón.»

Veamos un ejemplo. En la Figura 2.17 se pueden observar dos Tries (A y B). Los tres nodos correspondientes al patrón AGT son iguales en ambos Tries, ya que representan el mismo caracter en ambas estructuras y todos sus ancestros son iguales. Enmarcado en rojo podemos observar las ramas (y sus nodos) que difieren entre ambos Tries (4 nodos del Trie A que no pertenecen al B; 3 nodos del Trie B que no pertenecen al A). Podemos definir la función $diferenciaDeTries$ que, dado dos Tries, devuelve la cantidad de nodos que se obtienen luego de eliminar los nodos de la superposición entre ambas estructuras. En el ejemplo de la Figura 2.17 $diferenciaDeTries(Trie\,A, Trie\,B) = 7$, es decir, la cantidad de nodos enmarcados en rojo.

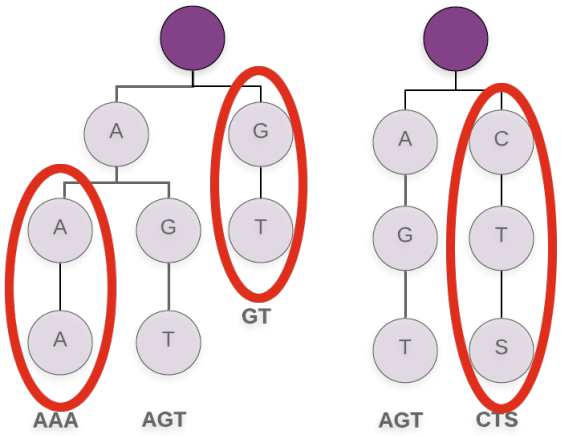


Fig. 2.17: Ejemplo de Trie A (izquierda) y Trie B (derecha). Enmarcado en rojo se pueden observar los nodos que difieren entre ambas estructuras.

Volvamos entonces al problema original. Definir cuán semejantes son dos familias de proteínas. O, por la negativa que para nuestro problema es equivalente, definir a través de una función de distancia cuán diferentes son dos familias de proteínas. Presentamos nuestro nuevo esquema de trabajo en la Figura 2.18.

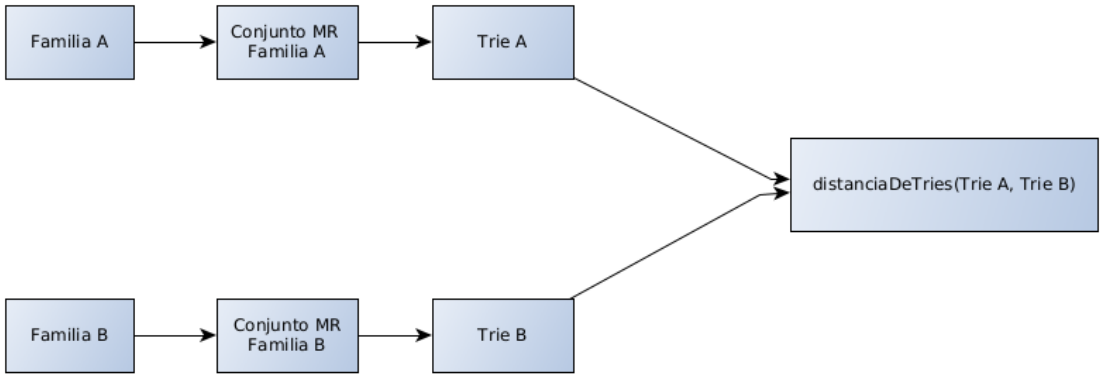


Fig. 2.18: Esquema de trabajo para computar el valor de distancia entre dos familias de proteínas.

Para obtener la distancia ($distanciaDeTries$), simplemente dividimos la cantidad de nodos diferentes entre ambos Tries sobre la cantidad total de nodos totales. En el ejemplo 2.17 $distanciaDeTries(Trie\,A, Trie\,B) = \frac{7}{15}$. En el Algorithm 1 se puede ver el pseudocódigo que proponemos para computar esta función.

Para contar los nodos diferentes, proponemos la función $diferenciaDeTries$. Su pseu-

docódigo se expone en el Algorithm 2.

Algorithm 1: Algoritmo que determina la distancia entre dos Tries

```
1 Function distanciaDeTries(Trie1,Trie2):
2     numerador = diferenciaDeTries(Trie1,Trie2)
3     divisor = cantidadDeNodos(Trie1) + cantidadDeNodos(Trie2)
4     distancia = numerador / divisor
5     return distancia
```

Algorithm 2: Algoritmo que cuenta la cantidad de nodos diferentes entre 2 Tries

input : *TrieA* and *TrieB*
output: cantidad de differencias entre ambos Tries

```
1  Function diferenciaDeTries(Trie1,Trie2):
       // Caso base
2      if Trie1 es vacio AND Trie2 es vacio then
3          return 0
4      if Trie1 es vacio then
5          return cantidadDeNodos(Trie2)
6      if Trie2 es vacio then
7          return cantidadDeNodos(Trie1)
       // Caso Recursivo
8      cantidadDiferencias = 0                    // Inicializacion del contador
       // ... lista de subtries ordenados lexicograficamente
9      hijosDelRootTrie1 = triesHijosDirectosDelRootOrdenados(Trie1)
10     hijosDelRootTrie2 = triesHijosDirectosDelRootOrdenados(Trie2)
       // Comparamos los subtries de ambas listas
11     while (|hijosDelRootTrie1| > 0 AND |hijosDelRootTrie2| > 0) do
           // subtries a comparar (los primeros de cada una de las listas)
12         subtrie1=head(hijosDelRootTrie1)
13         subtrie2=head(hijosDelRootTrie2)
           // Si sus nodos root coinciden. Computamos la diferencia de sus hijos
14         if (root(subtrie1).letra()==root(subtrie2).letra()) then
15             cantidadDiferencias+=diferenciaDeTries(subtrie1, subtrie2)
               // Ya computamos la diferencia de hijos, seguimos con el resto de los
                  Tries
16             hijosDelRootTrie1 = tail(hijosDelRootTrie1)
17             hijosDelRootTrie2 = tail(hijosDelRootTrie2)
           // Si el root(subtrie1)<root(subtrie2) contamos todo subtrie1
18         if (root(subtrie1).letra() < root(subtrie2).letra()) then
19             cantidadDiferencias+=cantidadDeNodos(subtrie1)
               // Ya contabilizamos subtrie1. Seguimos con el resto del Trie.
20             hijosDelRootTrie1 = tail(hijosDelRootTrie1)
           // Si el root(subtrie1)>root(subtrie2) contamos todo subtrie2
21         if (root(subtrie1).letra() > root(subtrie2).letra()) then
22             cantidadDiferencias+=cantidadDeNodos(subtrie2)
               // Ya contabilizamos subtrie2. Seguimos con el resto del Trie.
23             hijosDelRootTrie2 = tail(hijosDelRootTrie2)
```

```
23
    // Si quedaron elementos sin revisar de la lista de subtries del Trie1
24  while (|hijosDelRootTrie1| > 0) do
        // subtrie a contabilizar
25      subtrie1=head(hijosDelRootTrie1)
        // Se contabiliza la totalidad de nodos del subtrie
26      cantidadDiferencias+=cantidadDeNodos(subtrie1)
        // Ya contabilizamos subtrie1. Seguimos con el resto del Trie.
27      hijosDelRootTrie1 = tail(hijosDelRootTrie1)
    // Si quedaron elementos sin revisar de la lista de subtries del Trie2
28  while (|hijosDelRootTrie2| > 0) do
        // subtrie a contabilizar
29      subtrie2=head(hijosDelRootTrie2)
        // Se contabiliza la totalidad de nodos del subtrie
30      cantidadDiferencias+=cantidadDeNodos(subtrie2)
        // Ya contabilizamos subtrie2. Seguimos con el resto del Trie.
31      hijosDelRootTrie2 = tail(hijosDelRootTrie2)
32  return cantidadDiferencias
```

Algorithm 3: Algoritmo que contabiliza la cantidad de nodos de un Trie

```
1 Function cantidadDeNodos(aTrie):
2     cantidad = 1                                   ▷ Root
3     for each subtrie ∈ triesHijosDirectosDelRoot(aTrie) do
4         cantidad+= cantidadDeNodos(subtrie)
5     return cantidad
```

A continuación mostramos un ejemplo para que se comprenda mejor cómo funciona tanto el esquema de trabajo que detallamos en la Figura 2.18 como el algoritmo descripto para el cómputo de distancia entre Tries. Como primer paso del esquema de trabajo, seleccionamos las familias de proteínas a comparar. En este caso, elegimos a modo de ejemplo simplificado, las familias A y B. Luego, siguiendo el esquema de trabajo, obtenemos los patrones maximales de repetición de ambas familias. Los mismos se encuentran listados en la Tabla 2.4. A partir de estos patrones, generamos los Tries asociados (ver Figura 2.19). Por último, utilizando ambos Tries como entrada, ejecutamos el algoritmo *distanciaDeTries*, que llama a *diferenciaDeTries* para cuantificar las diferencias entre ambos Tries, y finalmente las divide por la suma de la cantidad de nodos de cada Trie. Veamos cómo es la ejecución paso a paso.

Conjunto de patrones maximales de repetición	
Familia A	Familia B
AAA AGT GT	AGT CTS

Tab. 2.4: Conjuntos de patrones maximales de repetición de las familias A y B.

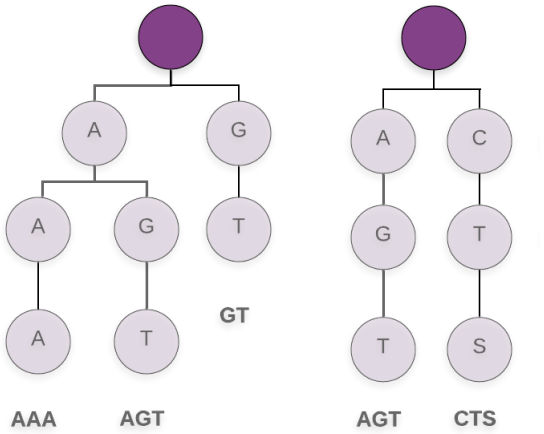


Fig. 2.19: Trie A (izquierda) y Trie B (derecha) correspondientes a los conjunto de patrones maximales de repetición de la Familia A y B, respectivamente.

Inicialmente la función $distanciaDeTries$ recibe como entrada al $Trie\,A$ y al $Trie\,B$. Como primer paso llama a la función $diferenciaDeTries$ (paso 2 del Algorithm 1), la cual también recibe como entrada al $Trie\,A$ y al $Trie\,B$. Dentro de la función $diferenciaDeTries$, como primer paso se chequea si alguno de los Tries es vacío, es decir, si alguna de las dos estructuras no contiene nodos (pasos 2-7 del Algorithm 2). En este caso ninguna es vacía por lo que, a continuación, se generan las listas de los nodos hijos de la raíz de cada Trie, ordenados lexicográficamente (pasos 9-10 del Algorithm 2). En la Figura 2.20, las flechas apuntan al nodo raíz de cada árbol. Las listas de hijos de los nodos root, ordenados lexicográficamente, generadas son:

- $hijosDelRootTrie1 = \{A, G\}$ correspondiente al Trie A

- $hijosDelRootTrie2 = \{A, C\}$ correspondiente al Trie B

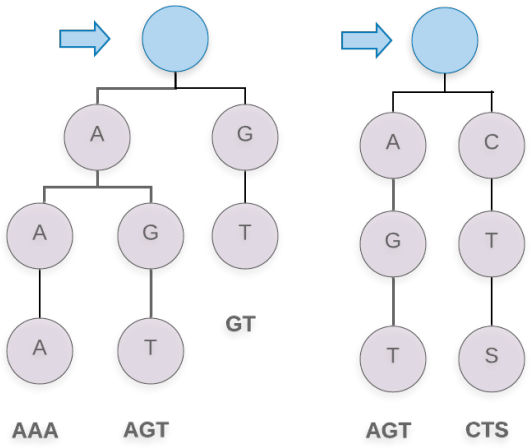


Fig. 2.20: Trie A (izquierda) y Trie B (derecha), que se pasan como parámetro en el Paso 2 desde el Algorithm 1 al Algorithm 2.

Como ambas listas poseen elementos y están ordenadas lexicográficamente, el siguiente paso es comparar si sus cabezas son iguales (paso 14 del Algorithm 2). En la Figura 2.21 se puede notar cuál es el sentido de esta comparación. En este caso, ambos nodos (A correspondiente al Trie A y A correspondiente al Trie B) coinciden. El siguiente paso a ejecutar es comparar ambos sub-tries que se originan a partir de estos nodos, ya que podrían diferir (de hecho lo hacen). Para ello, se ejecuta una llamada recursiva de la misma función con los sub-tries correspondientes a las ramas A de cada Trie, tal como se ve en la Figura 2.22 (paso 15 del Algorithm 2).

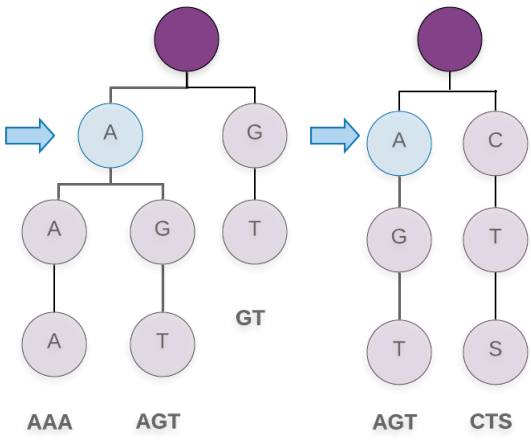


Fig. 2.21: Comparación de Nodo A (Trie izquierda) con Nodo A (Trie derecha).

El primer paso de la llamada recursiva a $diferenciaDeTries$ verifica si alguno de los sub-tries ingresados es vacío. No es el caso, por lo tanto se generan las listas de los nodos hijos de la raíz de cada Trie, ordenados lexicográficamente. En la Figura 2.22, las

flechas apuntan al nodo raíz de cada árbol. Las listas de hijos de los nodos root, ordenados lexicográficamente, generadas son:

- $hijosDelRootTrie1 = \{A, G\}$ correspondiente al sub-trie del Trie A
- $hijosDelRootTrie2 = \{G\}$ correspondiente al sub-trie del Trie B

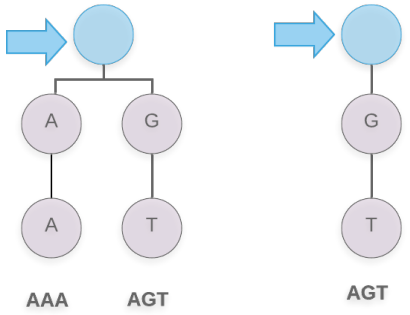


Fig. 2.22: Cómputo de $diferenciaDeTries$ entre el subtrie de la derecha y el subtrie de la izquierda.

Tomamos las cabezas de ambas listas. Como se puede ver en la Figura 2.23, los nodos (A y G) no coinciden. Ya que A es menor lexicográficamente a G, entonces, se realiza una llamada a la función auxiliar *cantidadDeNodos* (ver Algorithm 3) que contabiliza todos los nodos del sub-trie de la rama A del Trie A (ver Figura 2.24). Esto se debe a que todos estos nodos difieren del Trie B, por lo tanto deberán ser contabilizados.

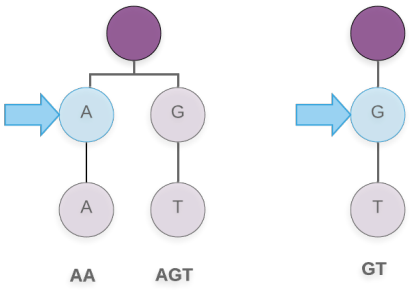


Fig. 2.23: Comparación de Nodo A (Trie izquierda) con Nodo G (Trie derecha)

La función auxiliar *cantidadDeNodos* (Algorithm 3) recorre todo el Trie que recibe como parámetro, en este caso el sub-trie del nodo A de la Figura 2.24, y retorna el valor correspondiente a la cantidad total de nodos que posee (incluido el nodo raíz). En este caso el sub-trie contiene dos nodos, por lo que retorna 2.

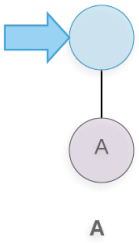


Fig. 2.24: Cómputo de *cantidadDeNodos* del subtrie.

Obtenido este valor, se descarta de la lista al sub-trie A de A, quedando:

- $hijosDelRootTrie1 = \{G\}$ correspondiente al sub-trie del Trie A
- $hijosDelRootTrie2 = \{G\}$ correspondiente al sub-trie del Trie B

Como ambas listas siguen siendo no vacía, tomamos la cabeza de ambas y las comparamos (ver Figura 2.25). En este caso ambos nodos coinciden, por lo que se genera una nueva llamada recursiva, la misma consta del sub-trie T correspondiente al Trie A y del sub-trie T correspondiente al Trie B. En esta nueva llamada volverán a ser iguales las cabezas de la listas, generando una nueva llamada recursiva. Finalmente ambos sub-tries serán vacíos y la función retornará cero, finalizando de esta manera la rama de esta recursión.

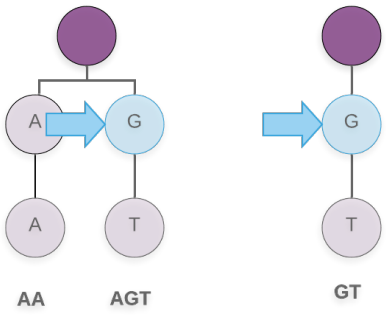


Fig. 2.25: Comparación de Nodo G (Trie izquierda) con Nodo G (Trie derecha).

Así, la cantidad parcial de nodos diferentes correspondientes al paso iniciado en las ramas A de la Figura 2.21 es de 2. Resueltas estas ramas, se continúa con las ramas restantes. Se actualizan las listas de hijos de los nodos root de ambos Tries (eliminando sus cabezas) y de esta manera obtenemos:

- $hijosDelRootTrie1 = \{G\}$ correspondiente al Trie A
- $hijosDelRootTrie2 = \{C\}$ correspondiente al Trie B

En este caso los nodos a comparar son G correspondiente al Trie A y C correspondiente al Trie B (ver Figura 2.26). Como C es menor lexicográficamente que G, se contabiliza la

totalidad de nodos del sub-trie que depende de C (observar que difieren todos los nodos). Esta acción se realiza a través de la función auxiliar *cantidadDeNodos* (Algorithm 3) utilizando como parámetro el sub-trie C del Trie B (que contiene además los nodos T y S, ver Figura 2.27). La función retorna 3.

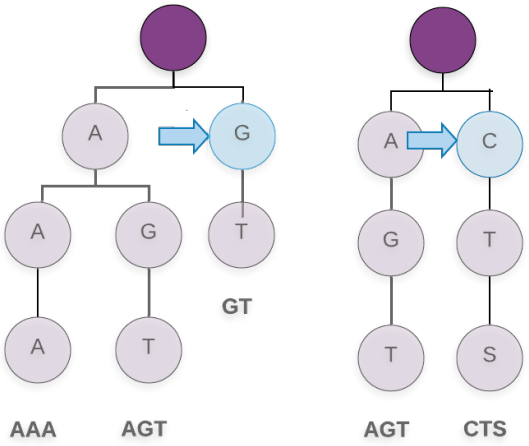


Fig. 2.26: Comparación de Nodo G (Trie izquierda) con Nodo C (Trie derecha).

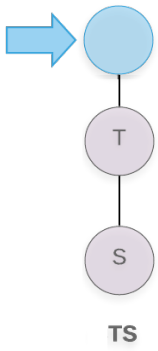


Fig. 2.27: Cómputo de *cantidadDeNodos* del subtrie.

Resuelta la rama C correspondiente al Trie B de la Figura 2.26, se continúa con las ramas restantes. Se actualiza las lista de hijos de los nodos root del Trie B (eliminando su cabeza) y de esta manera obtenemos:

- $hijosDelRootTrie1 = \{G\}$ correspondiente al Trie A
- $hijosDelRootTrie2 = \{\}$ correspondiente al Trie B

Dado que una de las listas es vacía, los nodos que nos quedaron no están contenidos en el otro Trie. Por lo tanto, debemos contabilizarlos. Así, se realiza la llamada a la función

auxiliar *cantidadDeNodos* (Algorithm 3) con el sub-trie G, T correspondiente al Trie A (ver Figura 2.28). La función retorna 2.

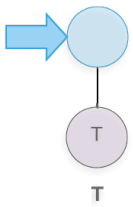


Fig. 2.28: Cómputo de *cantidadDeNodos* del subtrie.

Finalmente, nuestra función $diferenciaDeTries$ termina de recorrer ambos Tries y devuelve la cantidad total de nodos que difieren, es decir, $2 + 3 + 2 = 7$.

Aquí se retoma la función *distanciaDeTries* dónde ahora en la variable distancia tiene un valor de 7. Luego, llama a la función *cantidadDeNodos* para cada uno de los tries de entrada (paso 3 del Algorithm 1), es decir que, para el $Trie\, A$ retorna 8 nodos, y para el $Trie\, B$ retorna 7 nodos, quedando en la variable *divisor* el total de 15.
Finalmente el calculo de distancia es $distancia = \frac{7}{15}$ (paso 4 del Algorithm 1) y la función $distanciaDeTries$ retorna $0,466666667$ (paso 5 del Algorithm 1).

2.3.1. Análisis sobre la función distanciaDeTries

Para comprender el comportamiento de la función propuesta $distanciaDeTries$, generamos diversos escenarios, basados en conjuntos de patrones maximales. En la Tabla 2.5 se encuentran detallados cada uno de ellos en función de los elementos de entradas y las salidas respectivas esperadas de la función **distanciaDeTries**.

Entradas A y B	**Salida esperada**
$A \equiv B$	0
$A \sim B$	Se espera que a medida que crezca la intersección (sin variar la cantidad de elementos de A ni B) la salida tienda a 0 ya que los Tries cada vez se asemejan más.
$A \subset B$	Se espera que a medida que B crezca la salida tienda a 1 ya que A y B se diferencian más entre sí.
$A \neq B$	1

Tab. 2.5: Detalle de los escenarios propuestos según la relación entre los elementos de entrada A y B, y la salida esperada para la función $distanciaDeTries$

Los escenarios $A \equiv B$ y $A \neq B$ son los más sencillos de generar y verificar. Comencemos por ahí. Para el primero, generamos un solo conjunto de patrones A, armamos su respectivo Trie, y finalmente ejecutamos la función $distanciaDeTries(Trie\, A, Trie\, A)$.

Como valor de salida obtuvimos un 0, es decir, el resultado esperado. En el caso de $A \neq B$, generamos dos conjuntos disjuntos de patrones (A y B). Para dichos conjuntos armamos sus respectivos Tries, en este caso, $Trie\,A$ y $Trie\,B$, y finalmente ejecutamos a la función $distanciaDeTries(Trie\,A, Trie\,B)$. Como valor de salida obtuvimos nuevamente el resultado esperado, es decir, un valor de 1.

Para evaluar el escenario $A \sim B$ generamos 3 conjuntos de patrones a los cuales denominamos C1, C2 y C3[8], todos ellos disjuntos entre sí. Generamos el $Trie\,A$ en base a los elementos de C1, al $Trie\,B$ en base a los de C2, y los elementos de C3 los agregamos a ambos Tries, representando los elementos de la intersección. Ejecutamos múltiples veces la función $distanciaDeTries$, variando la cantidad de elementos de C3 de 0 a 3.000. En todas las corridas mantuvimos constante los elementos de los conjuntos C1 y C2, el conjunto C1 constaba de 4 elementos y el conjunto C2 constaba de 3 elementos. En la Figura 2.29 se puede observar el valor de $distanciaDeTries(Trie\,A, Trie\,B)$ en función de la cantidad de elementos presentes en la intersección de los elementos de entrada. Se puede notar que a medida que aumentan los elementos en la intersección, el valor de la función tiende a 0 (puntos azules), confirmando el resultado esperado. Este descenso no sucede de manera lineal. Realizamos un ajuste de la función $a * x^n/(x^n + b)$ (línea roja) con $a = 0{,}285640$, $n = -1{,}000028$ y $b = 0{,}999856$, obteniendo un $R^2 = 0{,}9999999852885731$, lo cual muestra que ajusta muy bien. Es decir, la función $distanciaDeTries$ decrece de forma asintótica a medida que crece la cantidad de elementos de la intersección entre ambas entradas.

Para evaluar el escenario $A \subset B$ generamos 2 conjuntos de patrones a los cuales denominamos C1 y C2, todos ellos disjuntos entre sí. Asignamos los elementos de C1 tanto al $Trie\,A$ como al $Trie\,B$. Los de C2 los utilizamos para ir incrementando el tamaño del TrieB. Ejecutamos múltiples veces el algoritmo, variando la cantidad de elementos del $Trie\,B$ dejando los elementos originales y agregándole de C2 de 0 a 3.000 nuevos elementos. En todas las corridas mantuvimos constante los elementos del conjuntos C1 en el $Trie\,A$ y en el $Trie\,B$. C1 consta de 4 elementos. En la Figura 2.30 se puede observar el valor de la función $distanciaDeTries(Trie\,A, Trie\,B)$ en función de la cantidad de elementos fuera de la intersección entre los parámetros de entrada. Se puede notar que a medida que aumentan los elementos fuera de la intersección, el valor de la función tiende a 1 (puntos azules), confirmando el resultado esperado. Este aumento no sucede de manera lineal. Realizamos un ajuste de la función $f(X) = 1 - a * (1/X)$ (línea roja) con $a = 5{,}722079$, obteniendo un $R^2 = 0{,}9728696130112844$, lo cual muestra que ajusta muy bien. Es decir, la función $distanciaDeTries$ crece de forma asintótica a medida que crecen la cantidad de elementos fuera de la intersección entre ambas entradas. Si bien la curva de ajuste no se ajusta de manera tan satisfactoria como el caso anterior, esto puede deberse a que no se están contemplando ciertas constantes.

[8] Los conjuntos se generaron de forma aleatoria en base al alfabeto de 20 aminoácidos.

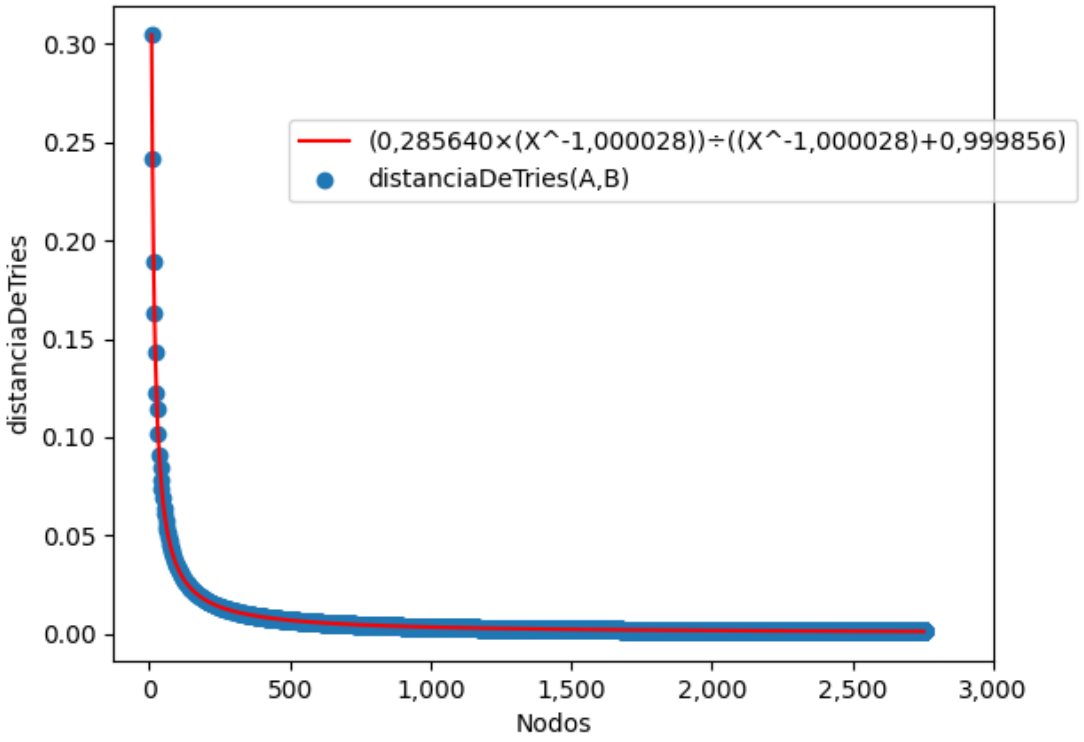


Fig. 2.29: Escenario $A \sim B$. Resultados de distanciaDeTries Algorithm 1 según el cambio de elementos de intersección del conjunto C3.

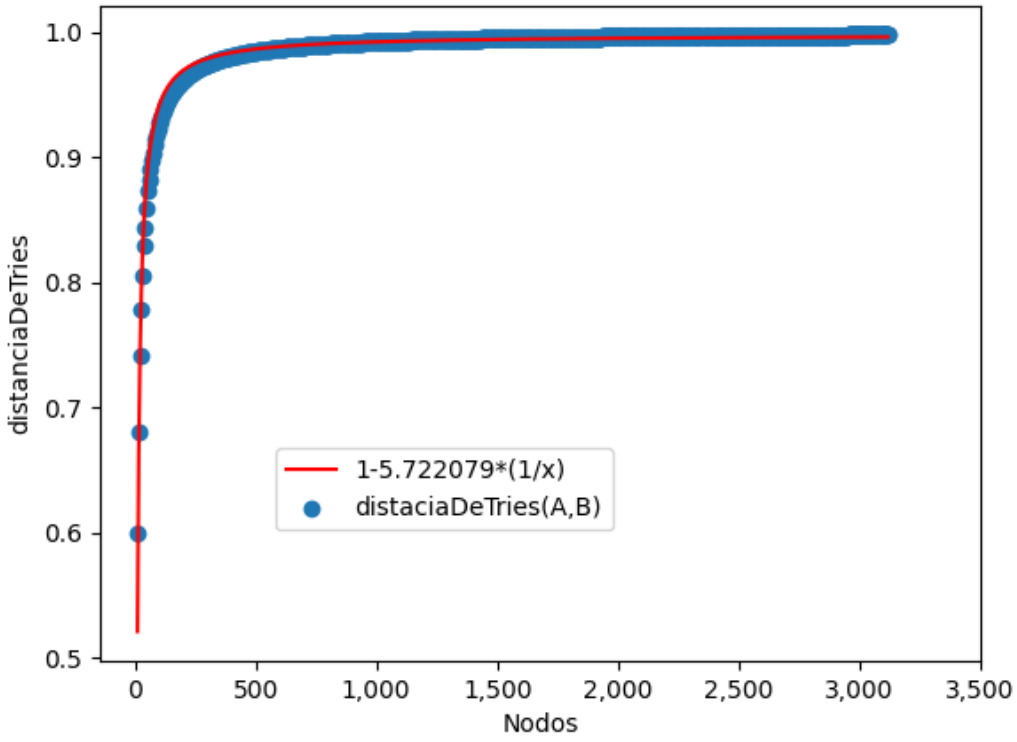


Fig. 2.30: Escenario $A \subset B$ Resultados de distanciaDeTries Algorithm 1 según la cantidad de elementos fuera de la intersección entre el conjunto C1 y C2.

En esta sección pudimos ver cómo se comporta nuestra función bajo distintos escenarios. En la próxima sección lo probaremos con un conjunto de datos de familias de proteínas reales.

2.4. distanciaDeTries - Diferencia de Tries

Como caso de estudio, nos basamos en 54 familias de proteínas utilizadas en el artículo [9]. De estas familias, 26 corresponden a familias de proteínas repetitivas, 20 a globulares y 8 a familias construidas a modo de control. Una descripción más detallada de las mismas se puede encontrar en la sección 5.2. Tomamos todos los posibles pares de a dos de dichas familias. A todos estos pares, les aplicamos el esquema de trabajo de la figura 2.18. Los conjuntos de patrones maximales de repetición utilizados se corresponden con la repeticiones maximales anidadas (NE) que se mencionan en [9]. Finalmente, obtuvimos los valores de $distanciaDeTries$ de todas las familias entre sí.

En la Figura 2.31 se muestran los resultados obtenidos, a través de un mapa de calor. En el eje x, se encuentran las 54 familias de nuestro caso de estudio. En el eje y se encuentran las mismas familias, conservando el mismo orden que el utilizado en el eje x. Los prefijos $R\_$, $G\_$ y $C\_$ en los nombres de las familias refieren a si corresponden a familias Repetitivas, Globulares o Control, respectivamente. En el punto (x, y) se encuentra el valor correspondiente al resultado de aplicar la función $distanciaDeTries$ entre los tries asociados a las repeticiones maximales de las familias x e y. Valores cercanos a 0 se representan con un color violeta oscuro y los cercanos a 1 con un color amarillo claro.

En la figura 2.31 se destaca notablemente la diagonal con un color violeta oscuro, que se condice con lo esperado, ya que en la comparación de una familia consigo misma se espera que la distancia dé 0. Otro valor que se destaca en violeta oscuro es el del par $NewAnk$ y $NewAnk\_WithoutP25963$. Este resultado tampoco sorprende ya que se trata de las mismas familias, excepto que la segunda posee una secuencia menos (la proteína $P25963$). Otro resultado que se destaca es la comparación de las familias Ldlrecepta y Ldlreceptb, siendo este un resultado que se esperaba dado que son familias que se las considera evolutivamente relacionadas. Por otro lado, se pueden observar algunos puntos en color violeta un poco más claros entre las familias control ABCtran_UNIFORM, ABCtran_SCRAMBLED, mixScrambled, mix, NewAnk_UNIFORM, NewAnk_SCRAMBLED, y mixUniformDistributed. Esto es algo buscado, es decir, que se distingan las familias de proteínas naturales de las familias de control. Otros resultados que se destacan en esta visualización es la cercanía (en gamas de naranja) entre las familias Arm-HEAT y PD40-NEWWD40, que se corresponde con los resultados reportados en el trabajo [9].

La visualización de la figura 2.31 no nos permite observar rápidamente las agrupaciones generadas, en particular en aquellos casos entre pares donde la distancia no es muy marcada. Para poder analizar en detalle las agrupaciones y niveles de cercanía entre las familias generamos una visualización en forma de dendograma, como se puede apreciar en la Figura 2.32. En dicha figura, el valor numérico del eje y se corresponde con la distancia entre los clusters de las agrupaciones calculada utilizando el método *complete* de la función *linkage* de la biblioteca *scipy.cluster.hierarchy*[9]. Los valores de la función $distanciaDeTries$ fue utilizada en una instancia previa para la generación de los clusters.

[9] scipy.cluster.hierarchy.linkage: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html`

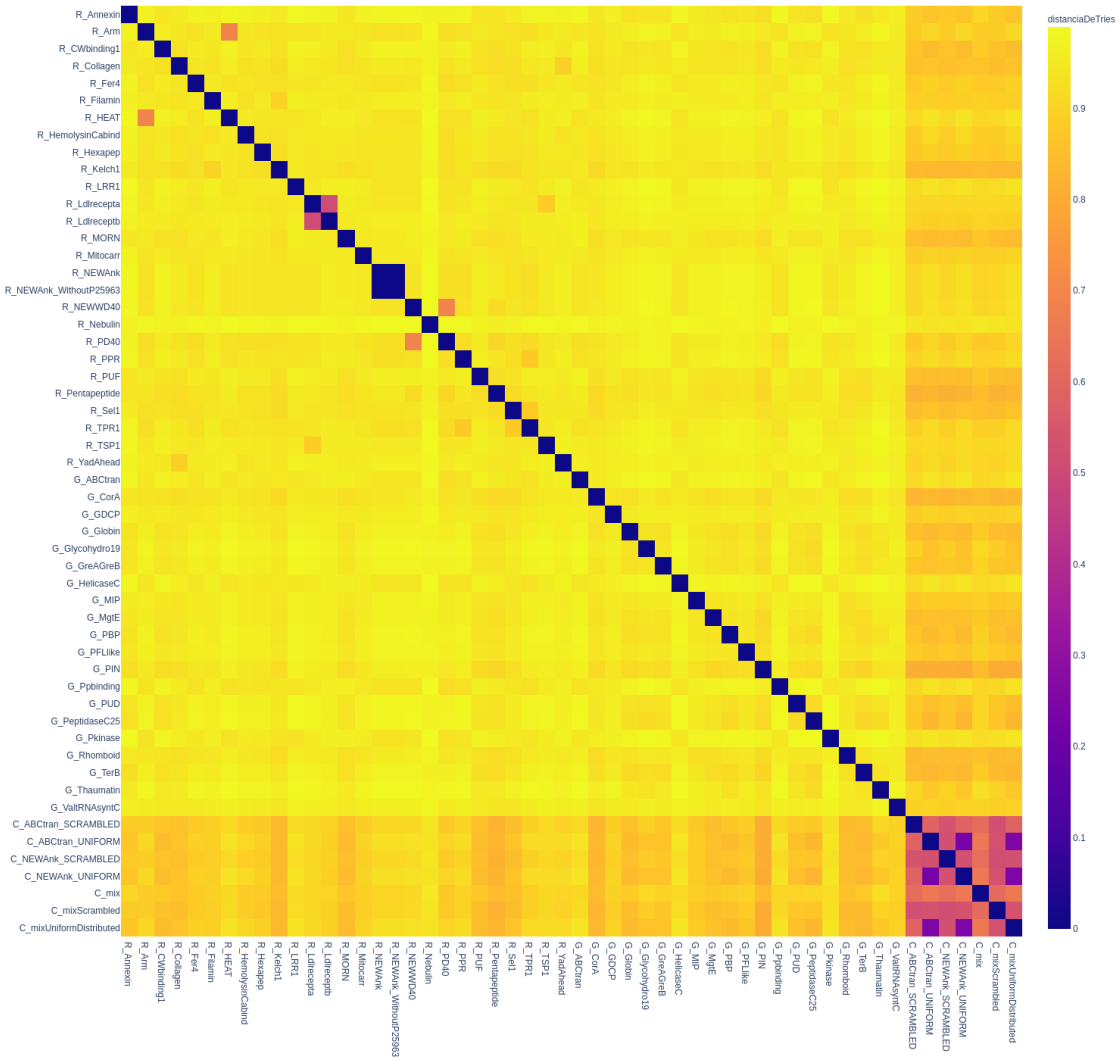


Fig. 2.31: Mapa de calor en base a la función *distanciaDeTries* aplicada entre todos los pares posibles de familias de nuestro caso de estudio.

En la figura se puede observar que las familias control se encuentran en una rama distinta de las familias naturales. Ese es un resultado muy importante ya que nuestro método estaría discriminando entre familias naturales de artificiales. Por otro lado, se puede observar que las familias $NewAnk$ y $NewAnk\_WithoutP25963$ dependen de la misma rama. Esto era esperable por lo explicado en el análisis del mapa de calor. Otras familias que esperábamos que se encuentren cercanas son las familias *Ldrecepta* y *Ldreceptb*, ya que son familias que están relacionadas entre sí. En el dendograma aparecen ambas muy cercanas, asociadas a la misma rama.

Por otro lado, tenemos los casos de las familias Arm-HEAT y PD40-NEWWD40 asociadas a una misma rama, cada par. Notar que en la figura 2.31 se destacan también compartiendo un color anaranjado entre sí. Este resultado se corresponde con los resultados obtenidos en el trabajo [9].

También se observa como un cluster separado, compartiendo una rama distinta, las 8 familias de control. No se observa la misma separación entre las familias globulares y repetitivas ya que, por ejemplo, en la agrupación que se observa en color rojo hay familias de ambas procedencias como los son Rhomboid y GDCP (globulares) y CWbinding y Kelch1 (repetitivas).

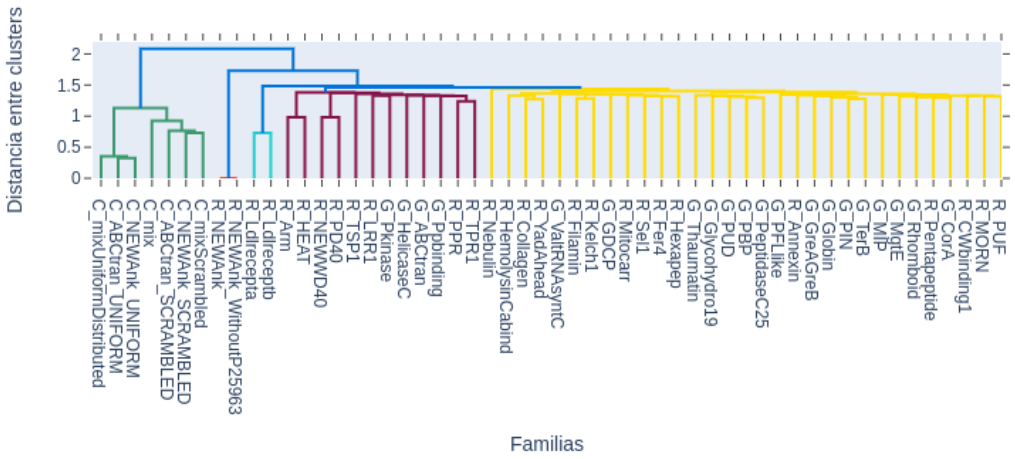


Fig. 2.32: Dendograma en base a valores de *distanciaDeTries* para todos los pares posibles de familias de nuestro caso de estudio. Como función de distancia entre grupos se utilizó la función *complete* de la biblioteca scipy.cluster.hierarchy.linkage

En la Figura 2.33 se puede observar un gráfico que unifica el mapa de calor y el dendograma en un misma figura. Notar que el orden del eje x (y el eje y) se modifica para que familias más cercanas se muestren juntas.

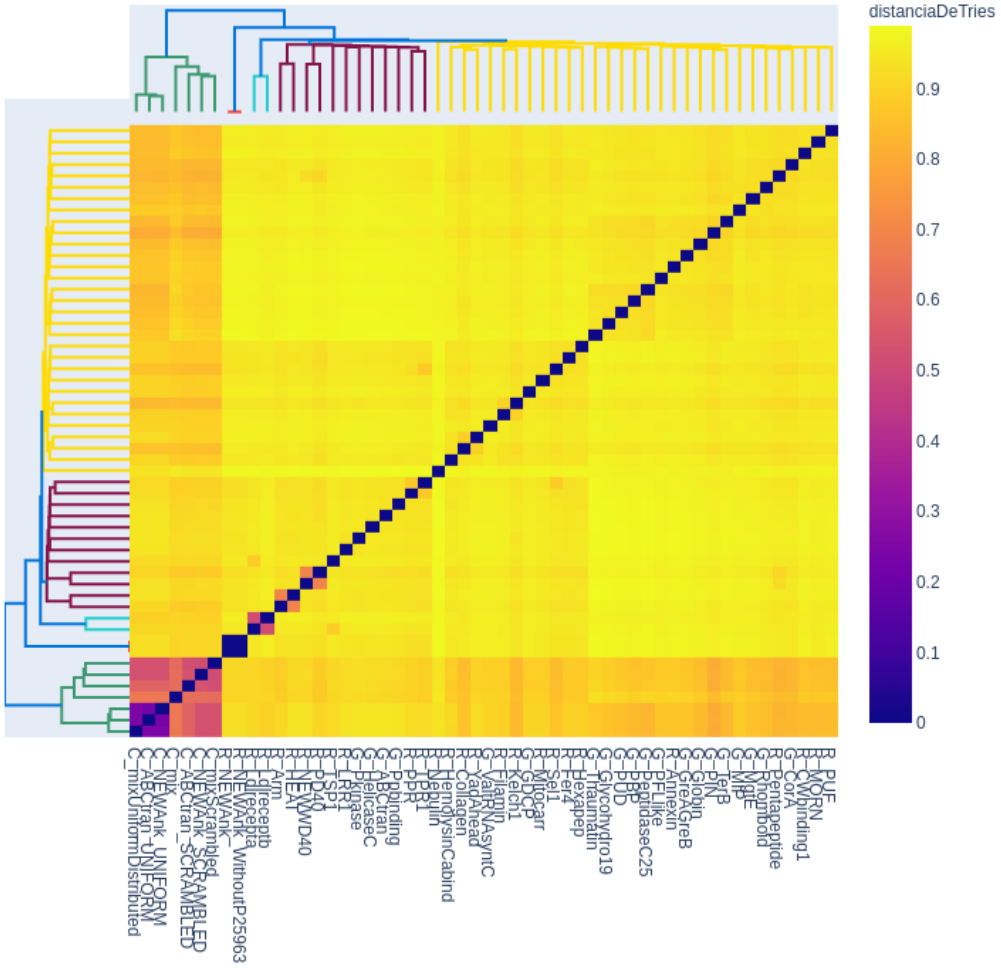


Fig. 2.33: Mapa de calor y dendograma en base a valores de *distanciaDeTries* para todos los pares posibles de familias de nuestro caso de estudio.

Estudio por Niveles

Con el fin de profundizar el análisis de la función *distanciaDeTries*, decidimos estudiar su comportamiento centrándonos en distintos cortes de los tries de entrada, en base a la altura de su árbol asociado. Considerando que un nivel es un corte de un trie, tomamos como nivel 0 a su raíz; nivel 1 al nivel 0 y los hijos directos de la raíz; nivel 2 al nivel 1 y los nietos de la raíz, y así siguiendo. Podemos interpretar al nivel n como el subconjunto de los patrones de longitud n utilizados para la construcción del trie. En la figura 2.34 se observa un ejemplo de los distintos niveles del trie A de la figura 2.19.

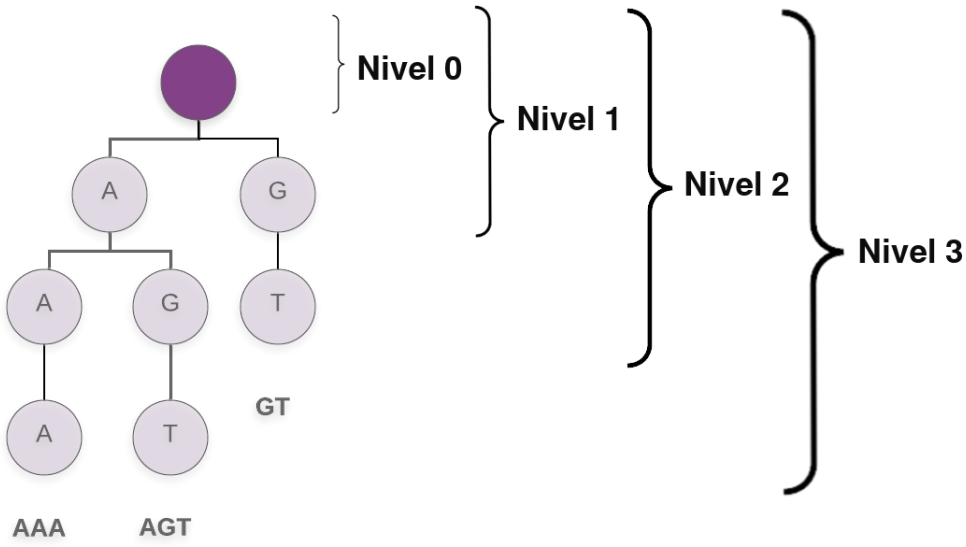


Fig. 2.34: Ejemplo de trie y sus distintos niveles.

Al computar la función *distanciaDeTries*, tomando tries de entrada acotados a un determinado nivel, se espera observar lo siguiente:

- **Nivel 1**. Corresponde a patrones de longitud 1, es decir, a las letras que representan a cada uno de los aminoácidos. Las familias que estudiamos están compuestas de varios patrones. Dada la diversidad de patrones es probable que siempre exista al menos uno que empiece con alguno de los 20 aminoácidos canónicos. De esta manera, al cubrir ambas familias todos los caracteres en el nivel 1, se espera que estas estén a distancia 0 entre sí.
- **Nivel 2**. Razonando de manera similar a como lo hicimos con el nivel 1, es probable que todas las combinaciones de 2 letras aparezcan en el comienzo de los patrones de ambas familias, por lo que también ambas distancias deberían ser 0 (o un valor similar).
- **Nivel 3**. En este nivel ya se espera que lentamente se empiecen a separar las familias, ya que compartan combinaciones de longitud 3 puede llegar a ser menos probable.
- **Nivel 4 y 5**. Se espera que sean los niveles de quiebre en el valor de distancia entre familias.

De manera arbitraria, comenzamos con la familia ABCtran (globular), para un análisis exploratorio por niveles. En la figura 2.35 se puede observar un mapa de calor correspondiente a los valores obtenidos con las función *distanciaDeTries* utilizando como parámetros a) el trie asociado a la familia ABCtran, recortado al nivel señalado en el eje y de la figura, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran. Tal como es de esperar, la columna correspondiente ABCtran posee en su totalidad *distanciaDeTries* igual a 0 (color violeta oscuro). Las restantes familias, poseen valores de *distanciaDeTries* bajos hasta el nivel 4 inclusive. A partir del nivel 5, se nota un quiebre en los valores (cambio brusco de color). Esto evidencia que patrones de

longitudes menores o iguales a 4 no permitirían distinguir entre familias, es decir, tienden a encontrarse en la mayoría de las familias estudiadas. Una de las familias que se destaca, en cercanía, es la familia globular Pkinase.

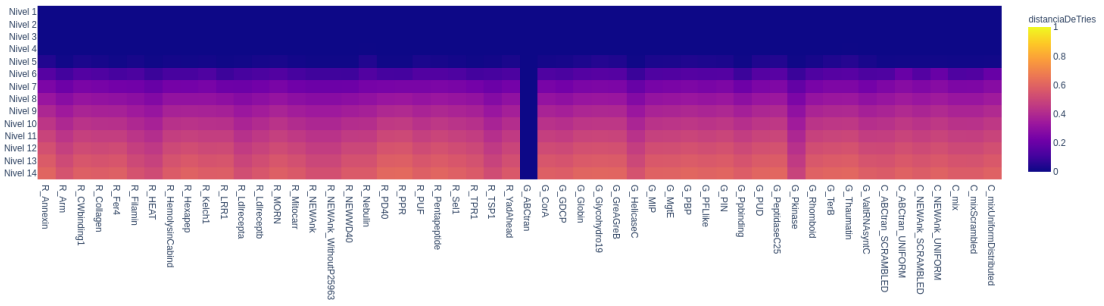


Fig. 2.35: Mapa de calor en base a la función *distanciaDeTries*, utilizando como entradas a) el trie asociado a la familia ABCtran, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran.

Definimos como *Trie teórico completo de nivel n* ($TrieTeorico_n$) al árbol conformado por todas las combinaciones posibles de patrones de longitud n. En el nivel 0, el $TrieTeorico_0$ contendrá $20^0 = 1$ nodos (nodo raíz), en el nivel 1 el $TrieTeorico_1$ tendrá $20^0 + 20^1 = 21$ nodos, en el nivel 2 el $TrieTeorico_2$ poseerá $20^0 + 20^1 + 20^2 = 421$ nodos, y así sucesivamente con cada nivel. Si computamos el porcentaje de nodos que posee a nivel 4 el trie de la familia estudiada (ABCtran), con respecto al $TrieTeorico_4$, obtenemos un valor de 73,67 %. Si calculamos lo mismo, pero ahora para nivel 5, obtenemos un valor de 15,85 %. Este descenso brusco, estaría arrojando cierta evidencia de que los patrones de nivel 5 en adelante son los que caracterizan más particularmente a la familia analizada, en concordancia por lo descripto en [9].

Avanzamos en el análisis estudiando el comportamiento de la familia ABCtran, pero en esta ocasión con sus aminoácidos permutados (ABCtran_SCRAMBLED). Una de las características principales de esta familia modificada es que contiene todas las combinaciones de patrones a nivel 1 y 2. A nivel 3 ya se nota un breve descenso, presentando a este nivel un 99, 85 % de nodos del $TrieTeorico_3$. A partir del nivel 4 el porcentaje desciende a un 72,15 %, y a nivel 5 decrece abruptamente a sólo un 9,38 %. Ya a nivel 6, el porcentaje es de 0,56 % nodos con respecto al $TrieTeorico_6$. Este resultado nos hace pensar que esta familia presentará un mayor distanciamiento con respecto a otras familias, en particular a niveles altos del trie.

En la figura 2.36 se puede observar, en el mapa de calor por niveles en base a la familia ABCtran_SCRAMBLED, un mayor cambio en la gama de color en el nivel 5, con respecto a la familia sin permutar (ABCtran). Es decir, como esperábamos que ocurriese, a nivel 5 esta familia se distancia más del resto. También se observa que, con respecto a las familias control (SCRAMBLED y UNIFORM), a nivel 5 hay poco distanciamiento (este grupo de familias se construyen de manera similar). A nivel 6, para estas familias control, ya se ve un cambio más abrupto de color para luego, a niveles más altos, estancarse en un valor de distancia casi constante. Esto se puede deber a la baja cantidad de nodos que poseen dichas familias a estos niveles.

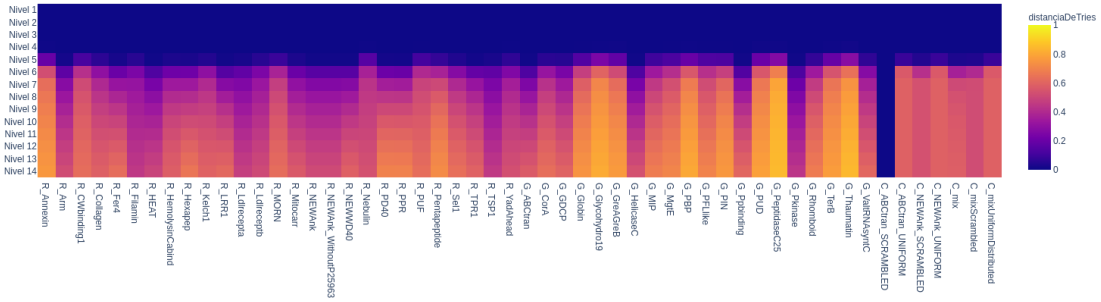


Fig. 2.36: Mapa de calor en base a la función *distanciaDeTries*, utilizando como entradas a) el trie asociado a la familia ABCtran_SCRAMBLED, recortado al nivel señalado en el eje y, y b) el Trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran_SCRAMBLED.

Extendimos el mismo estudio a la familia ABCtran_UNIFORM que, a diferencia de ABCtran_SCRAMBLED, está conformada por secuencias generadas con aminoácidos seleccionados al azar bajo una distribución uniforme. En la figura 2.37 se pueden observar los resultados obtenidos. Una comparación con ABCtran_SCRAMBLED arroja que ABCtran_UNIFORM se distancia más rápidamente de las familias naturales. Sólo se mantiene más cercana con otras familias control generadas con aminoácidos también tomados al azar.

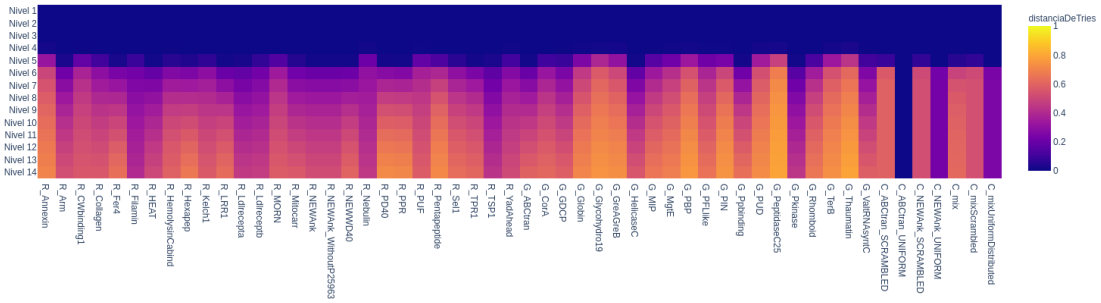


Fig. 2.37: Mapa de calor en base a la función *distanciaDeTries*, utilizando como entradas a) el trie asociado a la familia ABCtran_UNIFORM, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran_UNIFORM.

Por último, dado que ya analizamos una familia globular (ABCtran) y otras control (ABCtran_SCRAMBLED y ABCtran_UNIFORM), incluimos una familia repetitiva (Annexin). En la figura 2.38 se observa el mapa de calor por niveles de esta última familia. Se puede observar un color violeta oscuro, correspondiente a distancias cercanas a 0, con respecto a otras familias hasta el nivel 4 inclusive. A partir del nivel 5 ya se comienzan a distanciar de las restantes familias. Se puede observar un dictanciasmiento más rápido (a niveles más bajos) con respecto a las familias de control (escala de color amarillo más claro). También se puede notar la cercanía con las familias Filamin (repetitiva), Nebulin

(repetitiva) y Pkinase (globular).

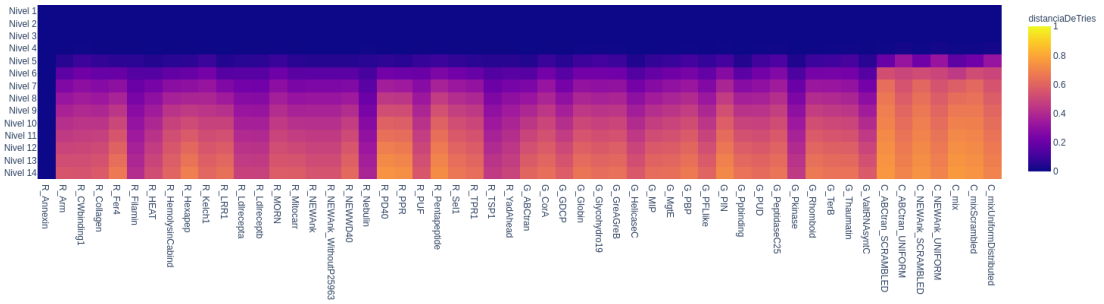


Fig. 2.38: Mapa de calor en base a la función $distanciaDeTries$, utilizando como entradas a) el trie asociado a la familia Annexin, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia Annexin.

Análisis de aporte por cada familia

Una manera simplificada de pensar cómo obtener el valor de la función $distanciaDeTries$ es: a) calcular la cantidad de nodos del trie A que no están en el trie B (de acá en más denominaremos a esta cantidad **aporte** de la familia A); b) calcular la cantidad de nodos del trie B que no están en el trie A (de acá en más denominaremos a esta cantidad **aporte** de la familia B); c) sumar ambas cantidades y dividirlas por el total de nodos de ambos tries. A modo de ejemplo, en la Figura 2.17, podemos contabilizar a) 4 nodos del Trie A que no pertenecen al Trie B; b) 3 nodos del Trie B que no pertenecen al Trie A; c) sumamos ambas cantidades $(4+3=7)$ y la dividimos por la cantidad de nodos diferentes entre ambos Tries (15). Así, en nuestro ejemplo, $distanciaDeTries(Trie\,A, Trie\,B) = \frac{7}{15}$. Como parte del estudio de la función $distanciaDeTries$, decidimos analizar cómo es el **aporte** de las cantidades de los puntos a) y b) previamente mencionados, en función del nivel al que comparamos los tries.

En la Figura 2.39 mostramos los valores a) y b) antes mencionados en función del nivel de corte de ambos tries. A modo exploratorio, tomamos de manera arbitraria las familias HelicaseC (globular) y ABCtran_SCRAMBLED. Se puede observar que las diferencias comienzan a aparecer a partir del nivel 5. Es decir, hasta el nivel 4 inclusive no hay nodos que distingan a ninguna de ambas familias. A partir del nivel 5 se diferencian marcadamente ambas curvas. Por un lado, se tiene el aporte correspondiente a la familia HelicaseC que sigue creciendo notablemente. Esto significa que desde el nivel 5 la familia HelicaseC aporta nodos a la función de distancia que no están en la familia ABCtran_SCRAMBLED. Por otro lado, se tiene la curva de aporte de la familia ABCtran_SCRAMBLED, en la cual se observa un crecimiento en el nivel 5 que luego no prospera mucho más, de hecho tiende a una constante a partir de nivel 7. Esto significa que la familia ABCtran_SCRAMBLED no aporta más nodos a la diferencia a partir del nivel 7. En particular teniendo en cuenta las características de la familia ABCtran_SCRAMBLED y el estudio anterior por niveles, se relaciona con la longitud de los patrones dentro de la familia. Viendo el estancamiento de la curva se puede decir que no tiene patrones más largos de 7 u 8 caracteres, por lo que no va a aportar más a la diferencia a partir de esos niveles.

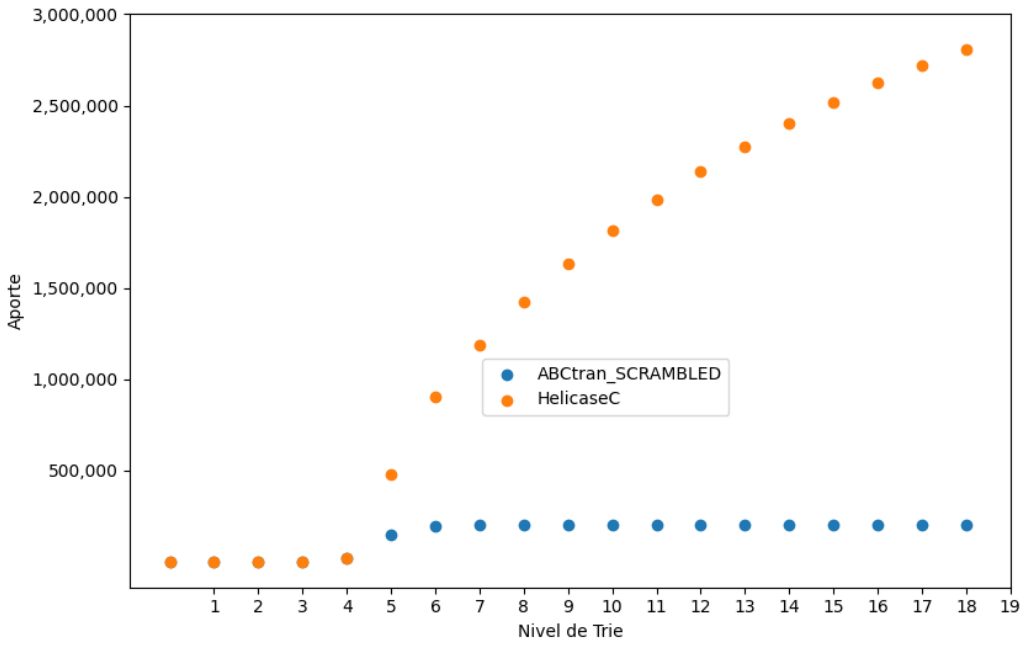


Fig. 2.39: Aporte de cada familia en función del nivel de su trie, para las familias HelicaseC (globular) y ABCtran_SCRAMBLED (control).

Para mostrar que no se trata de una particularidad de una sola familia, en el gráfico 2.40 mostramos los valores correspondientes a las familias ABCtran y ABCtran_SCRAMBLED. Se pueden observar curvas similares al gráfico anterior. La curva de aporte de ABCtran crece con mayor pendiente a partir del nivel 5. Se puede onservar que, en comparación a la curva de ABCtran_SCRAMBLED, ABCtran aporta las diferencias en patrones de longitudes mayores a 5. También se puede notar que la curva de ABCtran_SCRAMBLED aporta hasta nivel 7, pero siempre en muy baja proporción.

Los análisis realizados en esta sección fueron preliminares y no avanzamos en esta dirección. Dejamos planteado como trabajo a futuro profundizar el estudio de estos aportes.

2.4.1. Discusión

Encontramos en la función *distanciaDeTries* ciertas características satisfactorias: a) refleja la cercanía esperada entre algunas familias ya conocidas, como por ejemplo, Ldlrecepta-Ldlreceptb, NewAnk-NewAnk_WithoutP25963, Arm-HEAT y PD40-NEWWD40; b) permite diferenciar las familias control scrambled y uniform de las familias compuestas por proteínas naturales. Sin embargo, aún no permite discriminar entre familias globulares y familias repetitivas. Intentaremos, en las próximas secciones, abordar este problema proponiendo pequeños cambios a la función *distanciaDeTries* ya vista.

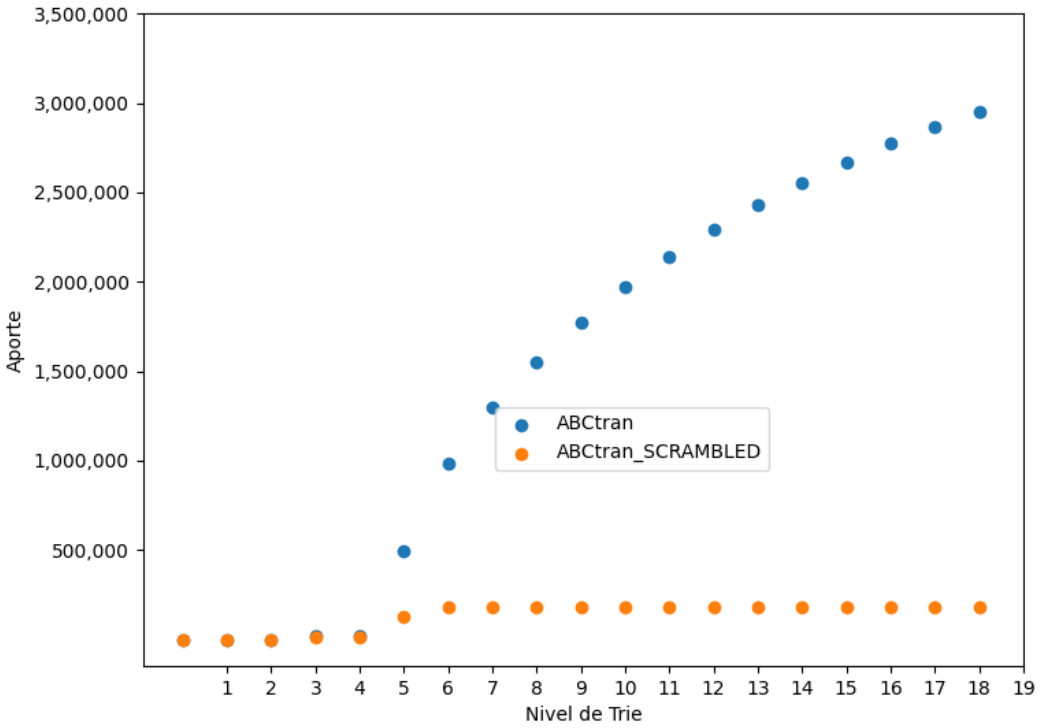


Fig. 2.40: Aporte de cada familia en función del nivel de su trie, para las familias ABCtran (globular) y ABCtran_SCRAMBLED (control).

2.5. distanciaDeTries - Contador Prefijos

Una característica que hasta el momento no se tuvo en cuenta en la función de *distanciaDeTries* propuesta es la cantidad de veces que se repite un prefijo de un patrón dentro del Trie. En esta sección presentaremos una función *distanciaDeTries* alternativa, contemplando la cantidad de repeticiones de los prefijos de los patrones.

Volviendo al ejemplo de la sección anterior (Tabla 2.4, *Sección 2.3 Cantidad de diferencias entre Tries*), en el cual la Familia A contiene los patrones AAA y AGT, se puede ver que ambas secuencias comparten la primera A, por lo que en su Trie asociaremos el valor 2 a dicho nodo A. Los demás nodos (correspondientes a AA y a GT, respectivamente) se repiten sólo una vez, por lo que les asociaremos un valor de 1. Lo mismo ocurre con el patrón más corto GT, al cual les asociaremos a todos sus nodos el valor 1. En el caso de la familia B, sus patrones no tienen nodos en común, por lo tanto, les asociaremos a todos ellos el valor 1. Una representación gráfica de cómo quedan ambos Tries se puede observar en la Figura 2.41.

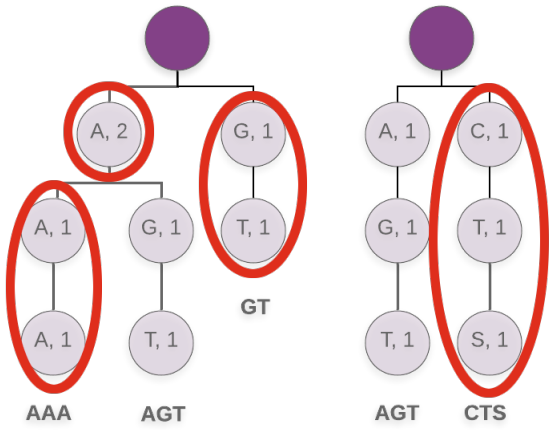


Fig. 2.41: Ejemplo de Trie A (izquierda) y Trie B (derecha). Enmarcado en rojo se pueden observar los nodos que difieren entre ambas estructuras. Notar que en el trie se representa la cantidad de veces que se repite el prefijo del patrón.

Para calcular nuestra nueva función de distancia, a la que denominaremos *distanciaDeTriesContadorPrefijos*, debemos proceder igual que la función *distanciaDeTries*, pero teniendo en cuenta el valor que posee asociado cada nodo. En el caso de la Figura 2.41, al contar las diferencia entre ambos tries, obtenemos: a) 5 nodos del Trie A que no pertenecen al B (con respecto a la función anterior, esta vez se agregó la primer A de AAA/AGT que cuenta doble); b) 3 nodos del Trie B que no pertenecen al Trie A. La función *diferenciaDeTriesContadorPrefijos* obtiene la suma de ambos valores. En el ejemplo de la Figura 2.41 $diferenciaDeTriesContadorPrefijos(Trie\,A, Trie\,B) = 8$, es decir, la cantidad de nodos enmarcados en rojo.

Para obtener la distancia ($distanciaDeTriesContadorPrefijos$), simplemente dividimos el valor resultante de $diferenciaDeTriesContadorPrefijos$ por la suma de los 2 valores máximos de los nodos de cada Trie (finalmente termina siendo la cantidad de repeticiones que posee el nodo hijo del root de mayor frecuencia de cada Trie). En el ejemplo de la Figura 2.41, la letra con mayor repetición del Trie A es 2, correspondiente al nodo A y en el caso del Trie B todas tienen asociado el valor uno por lo que el valor con mayor repetición es 1. Al sumar ambos números obtenemos el valor 3. Así, $distanciaDeTriesContadorPrefijos(Trie\,A, Trie\,B) = \frac{8}{3}$.

En el Algorithm 4 se puede ver el pseudocódigo que proponemos para computar esta función. Para contar los nodos diferentes, proponemos la función *diferenciaDeTriesContadorPrefijos*. Su pseudocódigo se expone en el Algorithm 5. Para este último, destacamos en color rojo las líneas que difieren con respecto al algoritmo presentado en la sección

anterior.

Algorithm 4: Algoritmo que determina la distancia entre dos tries en base a la cantidad de veces que se repite un prefijo de un patrón en un trie

```
1 Function distanciaDeTriesContadorPrefijos(Trie1,Trie2):
2     numerador = diferenciaDeTriesContadorPrefijos(Trie1,Trie2)
3     divisor = letraConMaryorRepeticion(Trie1, 0) +
       letraConMaryorRepeticion(Trie2, 0)
4     distancia = numerador / divisor
5     return distancia
```

Algorithm 5: Algoritmo que cuenta las diferencias entre dos tries en base a la cantidad de veces que se repite un prefijo de un patrón

```
1  Function diferenciaDeTriesContadorPrefijos(Trie1,Trie2):
       // Caso base
2      if Trie1 es vacio AND Trie2 es vacio then
3          return 0
4      if Trie1 es vacio then
5          return valoresDeNodos (Trie2)
6      if Trie2 es vacio then
7          return valoresDeNodos (Trie1)
       // Caso Recursivo
8      diferencias = 0                                  // Inicializacion del contador
       // ... lista de subtries ordenados lexicograficamente
9      hijosDelRootTrie1 = triesHijosDirectosDelRootOrdenados(Trie1)
10     hijosDelRootTrie2 = triesHijosDirectosDelRootOrdenados(Trie2)
       // Comparamos los subtries de ambas listas
11     while (|hijosDelRootTrie1| > 0 AND |hijosDelRootTrie2| > 0) do
           // subtries a comparar (los primeros de cada una de las listas)
12         subtrie1=head(hijosDelRootTrie1)
13         subtrie2=head(hijosDelRootTrie2)
           // Si las letras de sus nodos root coinciden. Computamos la diferencia de
              repeticiones entre ellos y las de sus hijos
14         if (root(subtrie1).letra()==root(subtrie2).letra()) then
               // Diferencia de repeticiones entre ambos nodos
15             diferencias+=abs(root(subtrie1).valor() - root(subtrie2).valor())
               // Diferencia de repeticiones entre los los hijos de ambos nodos
16             diferencias+=diferenciaDeTriesContadorPrefijos (subtrie1,
                subtrie2)
               // Ya computamos la diferencias de estos nodos y sus hijos, seguimos con
                  el resto de los Tries
17             hijosDelRootTrie1 = tail(hijosDelRootTrie1)
18             hijosDelRootTrie2 = tail(hijosDelRootTrie2)
           // Si el root(subtrie1).letra()<root(subtrie2).letra() contamos todo el
              subtrie1
19         if (root(subtrie1).letra() < root(subtrie2).letra()) then
20             diferencias+=valoresDeNodos (subtrie1)
               // Ya contabilizamos subtrie1. Seguimos con el resto del Trie.
21             hijosDelRootTrie1 = tail(hijosDelRootTrie1)
           // Si el root(subtrie1).letra()>root(subtrie2).letra() contamos todo el
              subtrie2
22         if (root(subtrie1).letra() > root(subtrie2).letra()) then
23             diferencias+=valoresDeNodos (subtrie2)
               // Ya contabilizamos subtrie2. Seguimos con el resto del Trie.
24             hijosDelRootTrie2 = tail(hijosDelRootTrie2)
```

```
24
    // Si quedaron elementos sin revisar de la lista de subtries del Trie1
25  while (|hijosDelRootTrie1| > 0) do
        // subtrie a computar
26      subtrie1=head(hijosDelRootTrie1)
        // Se computa la totalidad del subtrie1
27      diferencias+=valoresDeNodos (Trie1)
        // Ya contabilizamos subtrie1. Seguimos con el resto del Trie.
28      hijosDelRootTrie1 = tail(hijosDelRootTrie1)
    // Si quedaron elementos sin revisar de la lista de subtries del Trie2
29  while (|hijosDelRootTrie2| > 0) do
        // subtrie a computar
30      subtrie2=head(hijosDelRootTrie2)
        // Se computa la totalidad del subtrie2
31      diferencias+=valoresDeNodos (Trie2)
        // Ya contabilizamos subtrie2. Seguimos con el resto del Trie.
32      hijosDelRootTrie2 = tail(hijosDelRootTrie2)
33  return diferencias
```

Algorithm 6: Algoritmo que contabiliza la letra con mayor repetición de un Trie

```
1 Function letraConMaryorRepeticion(aTrie, currentMax):
2     currVal = root.valor()
3     if currVal > currentMax then
4         currentMax = currVal
5     for each subtrie ∈ triesHijosDirectosDelRoot(aTrie) do
6         currentMax = letraConMaryorRepeticion (subtrie, currentMax)
7     return currentMax
```

Algorithm 7: Algoritmo que acumula los valores asociados a los nodos de un Trie

```
1 Function valoresDeNodos(aTrie):
2     acumulador = root(aTrie).valor()                    ▷ Root
3     for each subtrie ∈ triesHijosDirectosDelRoot(aTrie) do
4         acumulador+= valoresDeNodos(subtrie)
5     return acumulador
```

A partir de la nueva función *distanciaDeTriesContadorPrefijos*, procedimos a ver cómo es su comportamiento, aplicándola en nuestro caso de estudio: las familias de proteínas de la sección anterior.

En la Figura 2.42 se muestran los resultados obtenidos a través de un mapa de calor. Cabe destacar que para esta función (*distanciaDeTriesContadorPrefijos*) no se esperan valores acotados al rango [0;1], ya que en su formulación, el numerador puede superar al denominador. De hecho, hay muchos valores que lo superan, obteniendo para nuestra figura una escala que va de 0 a 150. Más adelante, en la próxima sección, buscaremos acotarlos al rango [0;1]. Volviendo a la figura, nuevamente se destaca la diagonal con el color violeta

oscuro, como es esperado. También se destaca, próximo a la diagonal, un cuadrado de 2x2 correspondiente a las familias NEWAnk y NEWAnk_WithoutP25963. En la parte baja de la diagonal se puede ver un bloque de 7x7, correspondiente a las familias de control. Con respecto al mapa de calor en general, se notan gamas de colores más oscuros que los vistos en la sección anterior, dando la noción de la poca separación entre las familias que genera esta nueva función. Para profundizar sobre este punto, realizaremos otras visualizaciones.

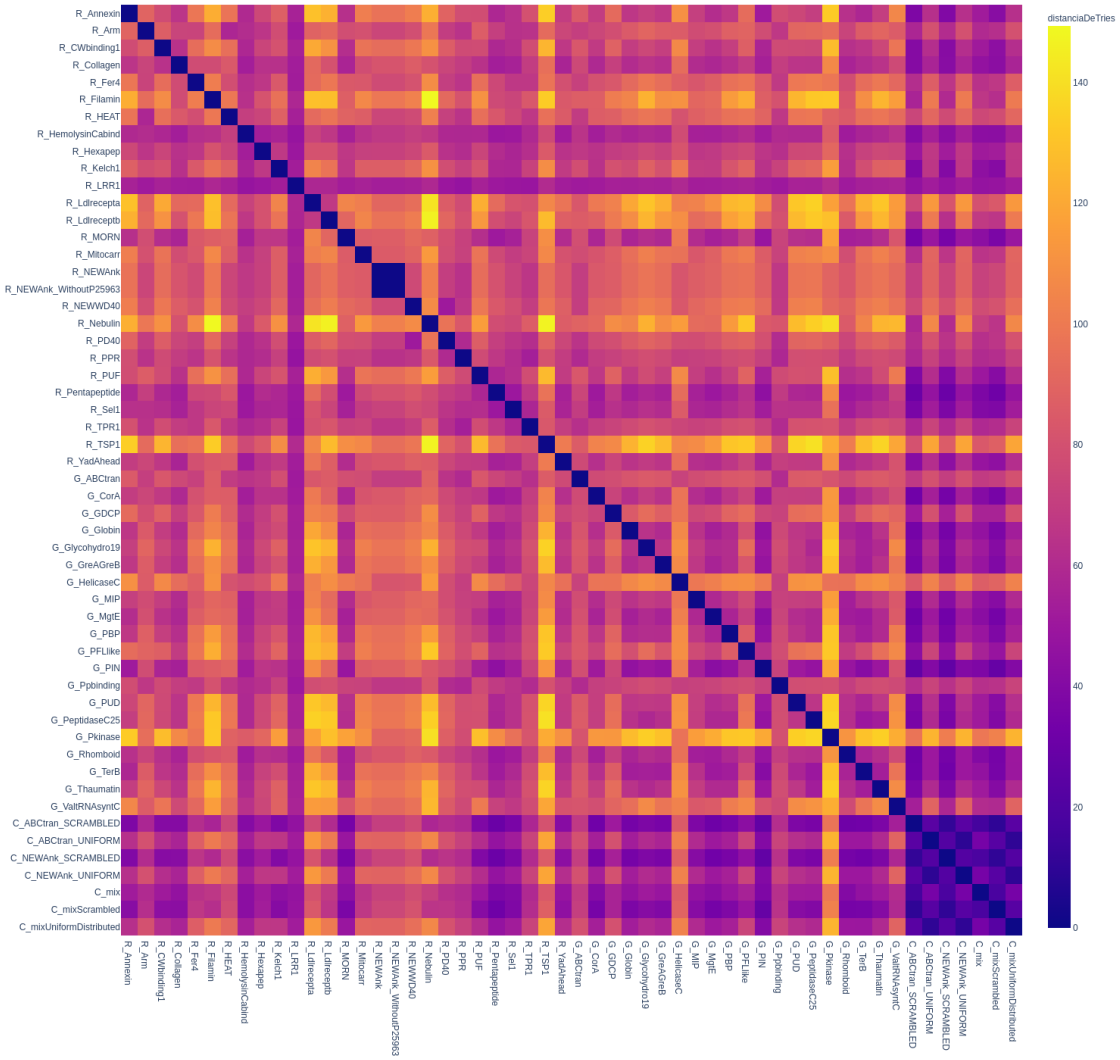


Fig. 2.42: Mapa de calor en base a la función *distanciaDeTriesContadorPrefijos* aplicada entre todos los pares posibles de familias de nuestro caso de estudio.

En la Figura 2.43 se presenta un dendograma generado, utilizando la función *distanciaDeTriesContadorPrefijos*, y la función *complete* de la biblioteca *scipy.cluster.hierarchy.linkage* para la distancia entre grupos o clusters. Se pueden observar algunas de las agrupaciones locales esperadas: a) NEWAnk y NEWAnk_WithoutP25963; b) Ldlrecepta y Ldlreceptb; c) Arm y HEAT; d) las 3 familias del tipo scrambled; y e) las 3 familias del tipo uni-

form. Sin embargo y lamentablemente, con esta nueva función perdimos la separación que habíamos logrado en la sección anterior entre familias artificiales (scrambled/uniform) por un lado y familias naturales por otro. También perdimos la separación entre las familias PD40 y NEWWD40. Como novedad, podemos mencionar que conseguimos agrupar por un lado las familias artificiales scrambled y por otro las familias artificiales uniform. Finalmente, debemos mencionar que no logramos una separación entre familias repetitivas y globulares.

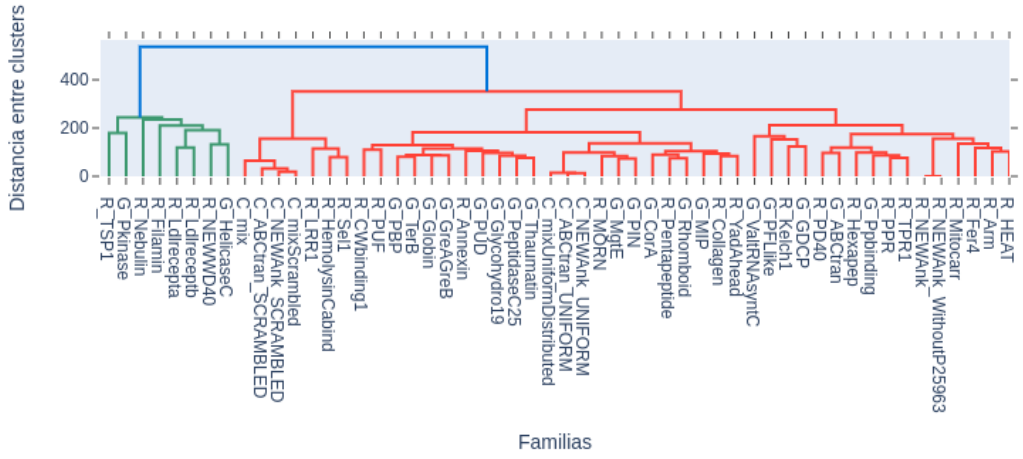


Fig. 2.43: Dendograma en base a valores de $distanciaDeTriesContadorPrefijos$ para todos los pares posibles de familias de nuestro caso de estudio.

En la figura 2.44 se muestra una visualización combinada de los dos gráficos anteriores (mapa de calor y dendograma), con el objetivo de realizar un análisis integrado. En dicha figura se puede apreciar que se separan las familias de control uniform y scrambled. Por otro lado, se destacan las familias NEWAnk y NEWAnk_WithoutP25963 y más sutilmente se destacan Ldlrecepta y Ldlreceptb. Finalmente, se puede notar una separación del grupo de familias que están en la rama verde del dendograma, formado por las familias TSP1, Pkinase, Nebulin, Filamin, Ldlrecepta, Ldlreceptb, NEWWD40, y HelicaseC. No pudimos encontrar ninguna propiedad biológica que a priori nos haga pensar que conformen un grupo.

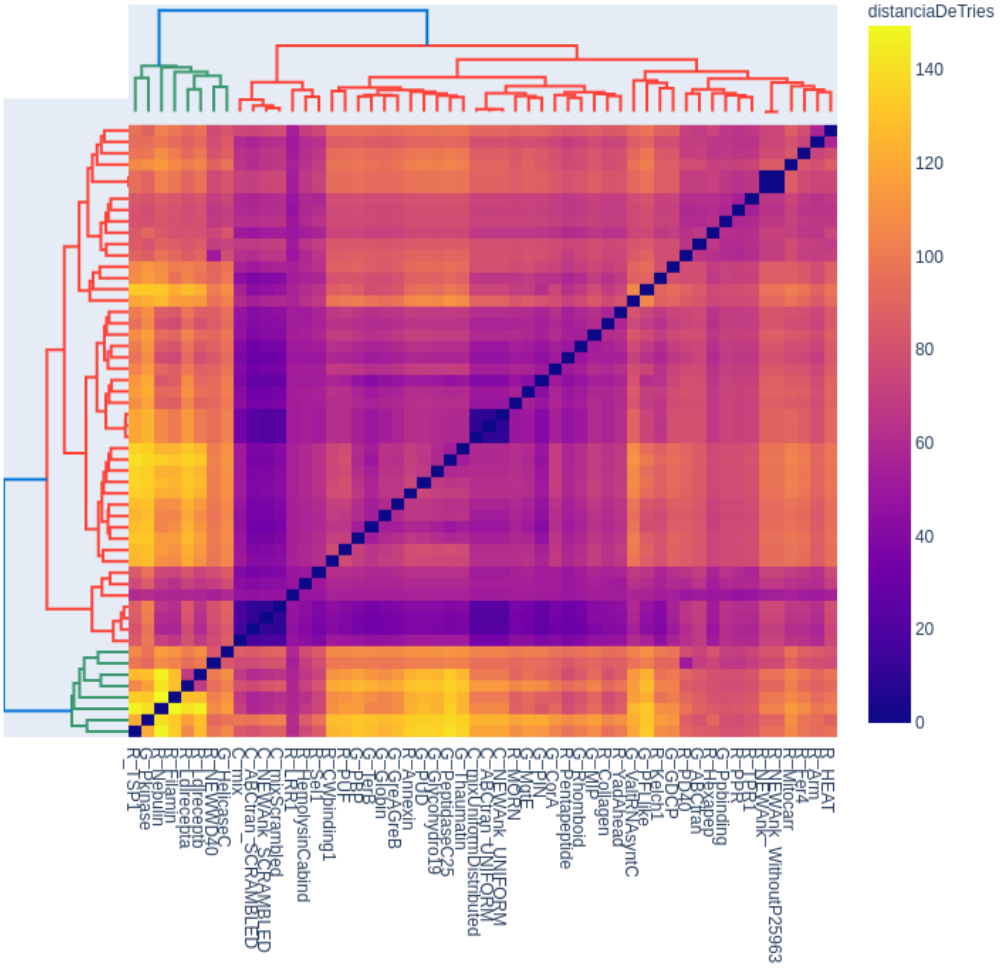


Fig. 2.44: Mapa de calor y dendograma en base a valores de $distanciaDeTriesContadorPrefijos$ para todos los pares posibles de familias de nuestro caso de estudio.

2.5.1. Conclusiones

La función $distanciaDeTriesContadorPrefijos$ pareciera alejarse de los resultados esperados. En particular, perdimos la separación que habíamos logrado en la sección anterior entre familias artificiales y familias naturales. En principio, como novedad, sólo conseguimos agrupar por un lado las familias artificiales scrambled y por otro las familias artificiales uniform. Debido a la mala performance que presentó esta función en general, decidimos no seguir avanzando en el estudio de ella.

2.6. distanciaDeTries - Contador Prefijos Normalizado

En la sección anterior analizamos la función *distanciaDeTriesContadorPrefijos*, que tenía en cuenta las repeticiones de los prefijos de los patrones, sin embargo, no obtuvimos

los resultados esperados. En esta sección se propone una variante de dicha función. La diferencia principal se encuentra en el denominador, ya que en esta nueva versión se buscará que los resultados estén dentro del rango [0;1]. En lo que respecta al valor utilizado en el numerador, seguiremos utilizando la misma función *diferenciaDeTriesContadorPrefijos*. Retomando el ejemplo de la figura 2.41, la nueva función propuesta *distanciaDeTriesContadorPrefijosNormalizado(Trie A, Trie B)*=$\frac{8}{14}$, donde el numerador continúa siendo *diferenciaDeTriesContadorPrefijos(Trie A, Trie B)=8* y el denominador se obtiene de la sumatoria de todos los valores asociados a cada uno de los nodos de ambos tries.

Algorithm 8: Algoritmo que determina la distancia entre dos tries en base a la cantidad de veces que se repite un prefijo de un patrón en un trie

```
1 Function distanciaDeTriesContadorPrefijosNormalizado(Trie1,Trie2):
2     numerador = diferenciaDeTriesContadorPrefijos(Trie1,Trie2)
3     divisor = valoresDeNodos (Trie1) + valoresDeNodos (Trie2)
4     distancia = numerador / divisor
5     return distancia
```

Aplicamos la nueva función *distanciaDeTriesContadorPrefijosNormalizado* a las familias de proteínas de nuestro caso de estudio. En la figura 2.45 se muestran los resultados obtenidos utilizando la visualización de un mapa de calor. En esta versión de la función, se puede observar que la escala es la esperada, ya que los valores de distancia están dentro del rango [0;1]. En el gráfico, se destacan la diagonal, las agrupaciones de las familias NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb, PD40-NEWWD40 y Arm-HEAT. También se puede observar un agrupamiento entre las familias de control.

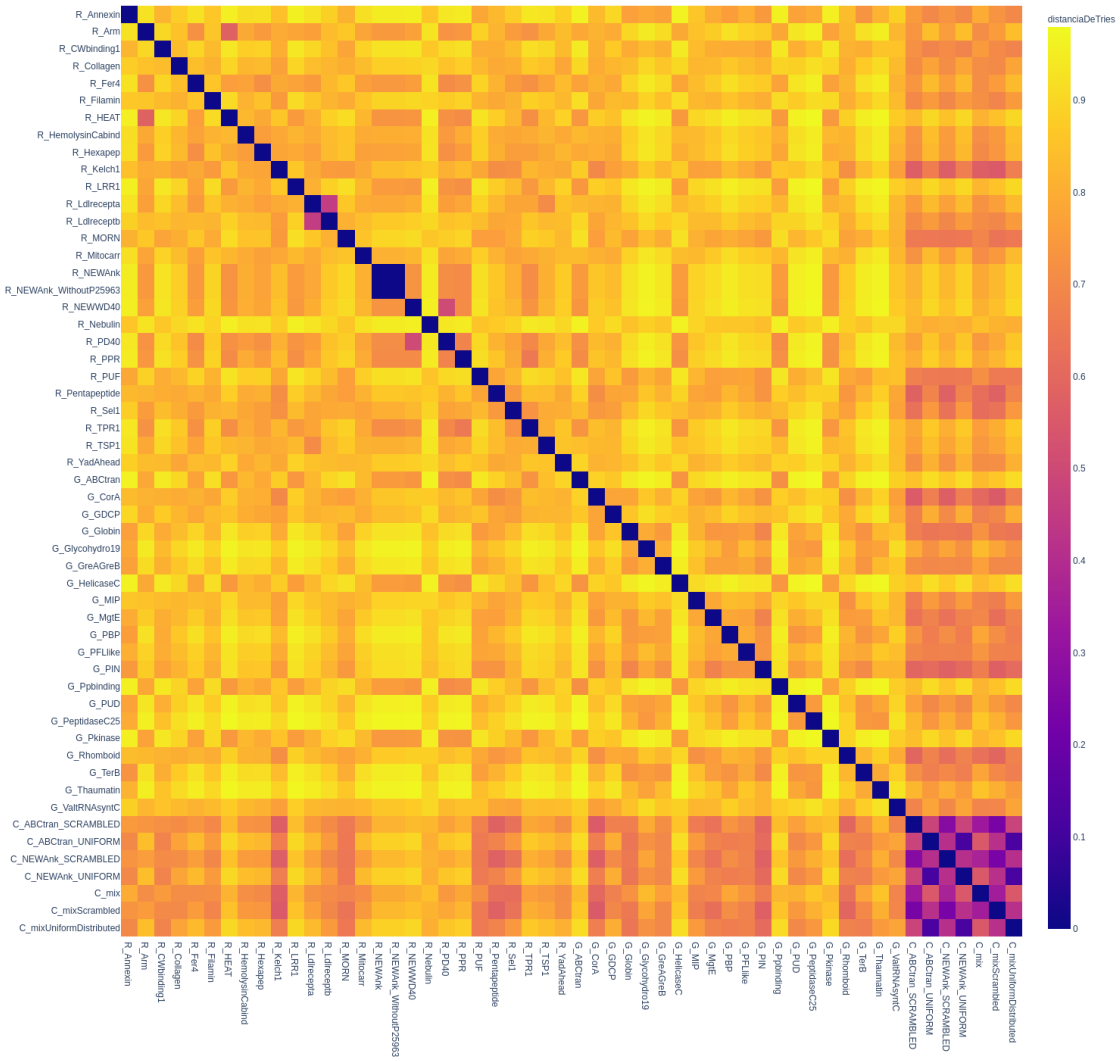


Fig. 2.45: Mapa de calor en base a la función $distanciaDeTriesContadorPrefijosNormalizado$ aplicada entre todos los pares posibles de familias de nuestro caso de estudio.

Para analizar las agrupaciones dadas por esta nueva función generamos la visualización de dendograma utilizando la función *complete* de la biblioteca *scipy.cluster.hierarchy.linkage* para la distancia entre grupos o clústeres. En la figura 2.46 se pueden observar 4 bloques bien diferenciados. Por un lado, en verde, se puede ver el grupo de las familias de control; luego, en rojo, celeste y púrpura las demás familias. Si bien no se distinguen las familias globulares de las repetitivas, en el grupo rojo se observa una separación de varias familias repetitivas y muy pocas globulares (HelicaseC, ABCtran, PPbinding, y Pkinase, solo 4 de 20 de las familias globulares), lo cual parecería ser muy prometedor.

En la figura 2.47, se presenta una visualización donde se combinan el mapa de calor y el dendograma. Se puede observar la separación entre familias control (dendograma verde) y las restantes familias. También se puede observar que dentro de las primeras, se separan uniform por un lado y scrambled por otro. La familia mix se encuentra más

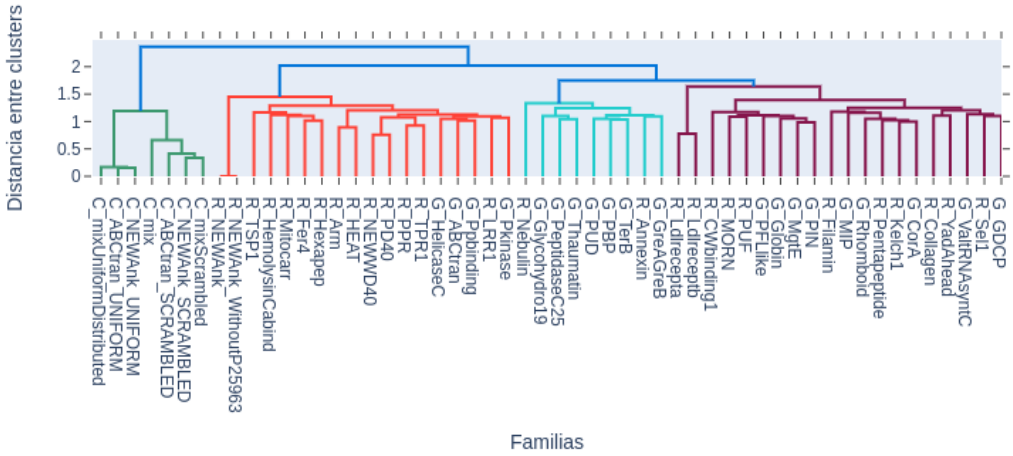


Fig. 2.46: Dendograma en base a valores de $distanciaDeTriesContadorPrefijosNormalizado$ para todos los pares posibles de familias de nuestro caso de estudio.

cercana a las scrambled que a las uniform, lo cual es deseable. Por otro lado, se observa un recuadro anaranjado (dendograma rojo), correspondiente a un agrupamiento de varias familias, en su mayoría, repetitivas. También se observan las asociaciones NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb, PD40-NEWWD40 y Arm-HEAT.

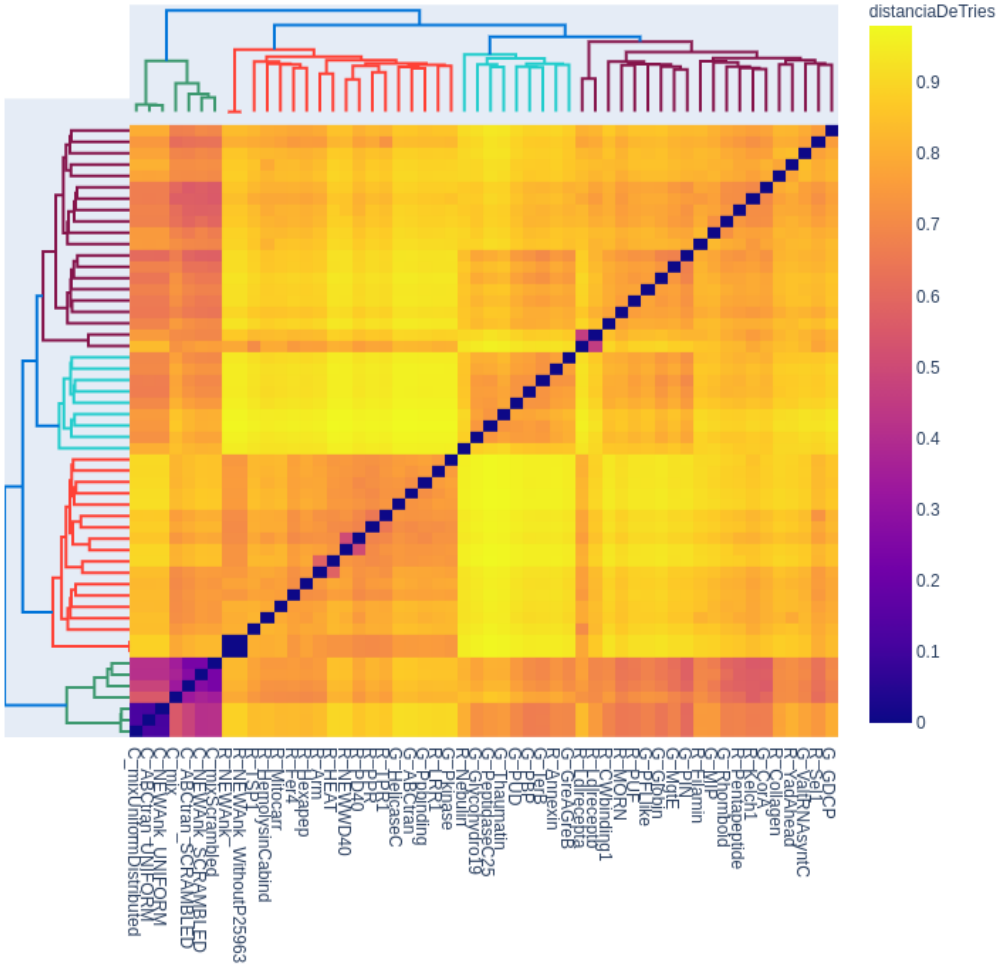


Fig. 2.47: Mapa de calor y dendograma en base a valores de *distanciaDeTriesContadorPrefijosNormalizado* para todos los pares posibles de familias de nuestro caso de estudio.

Estudio por Niveles

En un estudio por niveles de la función *distanciaDeTriesContadorPrefijosNormalizado*, a diferencia de lo que observamos en *distanciaDeTries*, esperamos observar un distanciamiento más temprano, es decir, en niveles previos a nivel 4, ya que los valores asociados a la repetición de prefijos debería diferenciar a las familias ya desde los primeros niveles.

En la figura 2.48 se muestran los resultados de la comparación por niveles en base a la familia ABCtran. Se observa que, ya desde nivel 2, comienza la separación entre las familias y ABCTran. Esta separación se acrecienta incluso hasta niveles altos. En algunas familias, como Annexin y Nebulin, esto ocurre de forma pronunciada, y en otras no tanto, como en el caso de HelicaseC o Pkinase. Queda como línea futura seguir analizando lo que ocurre con otras familias globulares distintas a ABCtran.

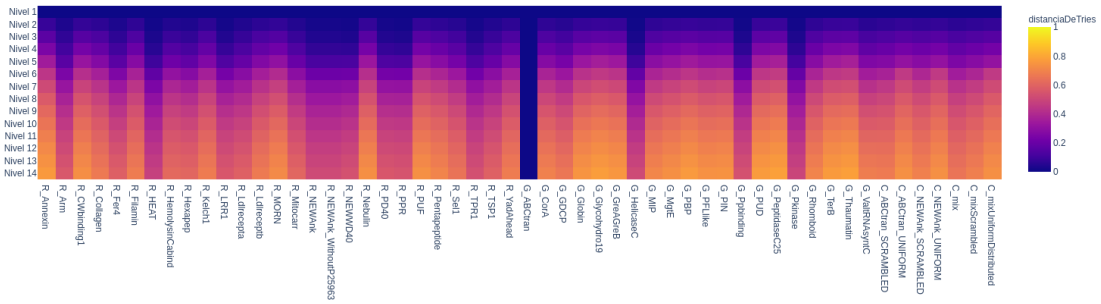


Fig. 2.48: Mapa de calor en base a la función *distanciaDeTriesContadorPrefijosNormalizado*, utilizando como entradas a) el trie asociado a la familia ABCtran, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran.

De la misma manera que lo hicimos previamente, analizaremos familias control (scrambled y uniform) y una familia repetitiva.

En la figura 2.49 se muestran los resultados de la comparación por niveles en base a la familia ABCtran_SCRAMBLED. Se observa un distanciamiento, como vimos con ABCtran, también a partir del nivel 2. Luego se destacan algunas familias que se alejan desde niveles bajos, como es el caso de PeptidaseC25 y Thaumatin. Otras familias se mantienen cercanas, coincidiendo en general con las familias de control.

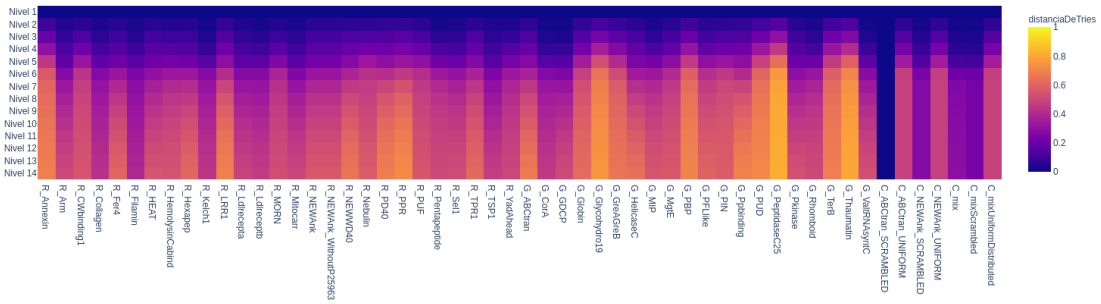


Fig. 2.49: Mapa de calor en base a la función *distanciaDeTriesContadorPrefijosNormalizado*, utilizando como entradas a) el trie asociado a la familia ABCtran_SCRAMBLED, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran_SCRAMBLED.

En la figura 2.50, realizada en base a la familia ABCtran_UNIFORM, se observa el distanciamiento desde el nivel 2. También se puede observar cómo algunas familias se alejan en distancia desde niveles bajos, como es el caso de Annexin y PPR. Otras familias se mantienen bastante cercanas, como es el caso de las familias de control.

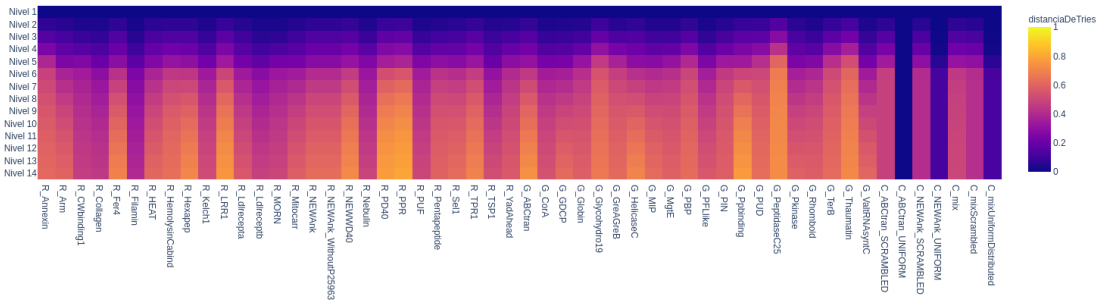


Fig. 2.50: Mapa de calor en base a la función *distanciaDeTriesContadorPrefijosNormalizado*, utilizando como entradas a) el trie asociado a la familia ABCtran_UNIFORM, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia ABCtran_UNIFORM.

En la figura 2.51, correspondiente a la familia Annexin, también se sigue observando el distanciamiento desde el nivel 2. Luego, se destacan algunas familias que se alejan desde niveles bajos como es el caso de PD40 y PPR. Otras se mantienen bastante cercanas, como es el caso de Nebulin y Filamin, las cuales también son del tipo repetitivas.

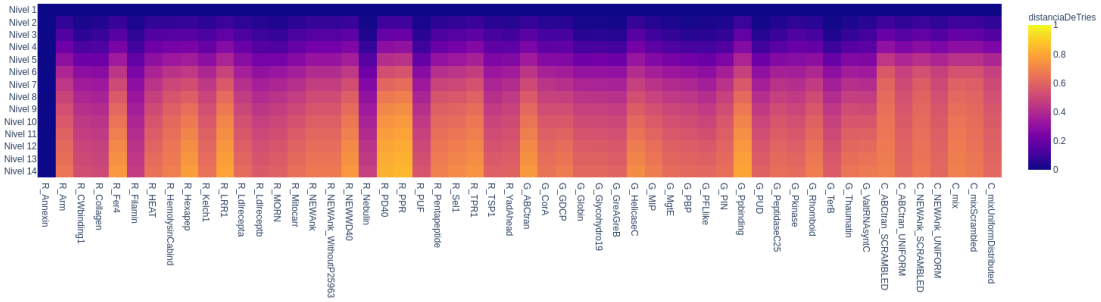


Fig. 2.51: Mapa de calor en base a la función *distanciaDeTriesContadorPrefijosNormalizado*, utilizando como entradas a) el trie asociado a la familia Annexin, recortado al nivel señalado en el eje y, y b) el trie asociado a la familia del eje x, recortado al mismo nivel que la familia Annexin.

Análisis de aporte por cada familia

Realizamos un análisis exploratorio del aporte que hace cada familia a la función *distanciaDeTriesContadorPrefijosNormalizado*. En la Figura 2.52 mostramos los resultados correspondientes a las familias HelicaseC (globular) y ABCtran_SCRAMBLED (control). Se puede observar que las diferencias comienzan ya desde el nivel 1. Esta diferencia crece con una pendiente pronunciada hasta llegar al nivel 8 donde la pendiente de HelicaseC comienza a disminuir. Se puede observar que la familia ABCtran_SCRAMBLED tiene una curva que es casi constante, creciendo muy poco hasta el nivel 6, y luego se vuelve constante. Su comportamiento es similiar a lo que sucedió en la Figura 2.39, con la función *distanciaDeTries*.

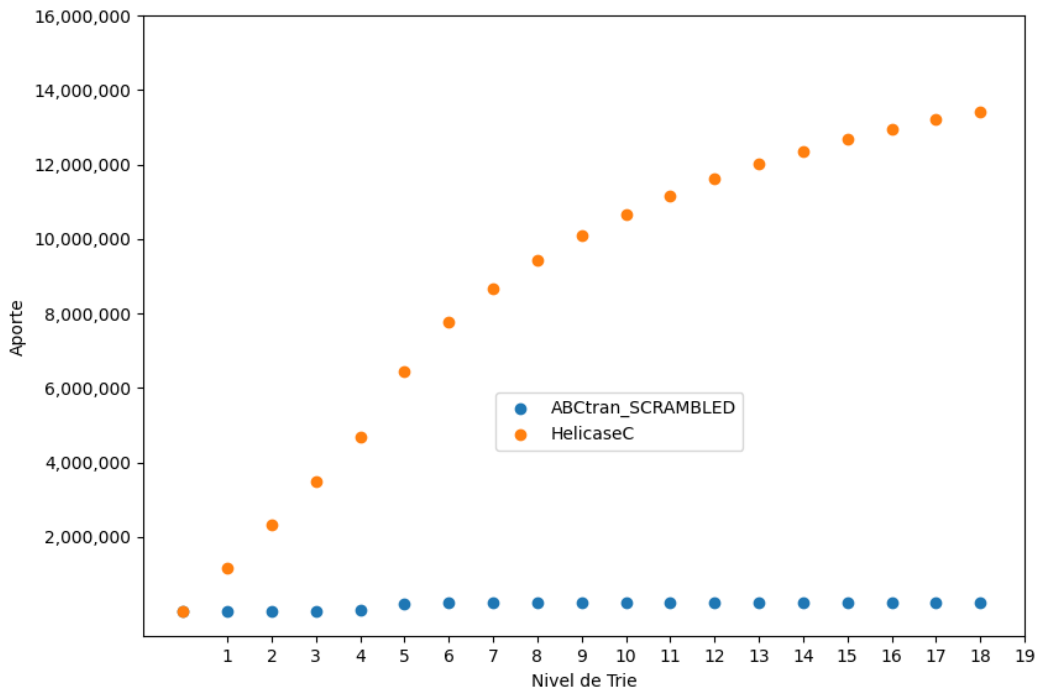


Fig. 2.52: Aporte de cada familia en función del nivel de su trie, para las familias HelicaseC (globular) y ABCtran_SCRAMBLED (control).

Para continuar la comparación con lo realizado en la sección previa, en la Figura 2.53 mostramos los resultados correspondientes a las familias ABCtran y ABCtran_SCRAMBLED. Se pueden observar curvas similares a la Figura 2.52. La curva de aporte de ABCtran crece con mayor pendiente a partir de nivel 1 hasta nivel 8, donde luego la pendiente disminuye. En contraste, la curva de ABCtran_SCRAMBLED aporta más diferencias al principio (en patrones de longitudes hasta 6) aunque valores muy acotados, y luego se mantiene constante.

Si bien es interesante estudiar el aporte individual que realiza cada familia, queda para un trabajo futuro continuar este análisis con las restantes familias.

2.6.1. Discusión

La función *distanciaDeTriesContadorPrefijosNormalizado* mostró ciertas características satisfactorias: a) refleja la cercanía esperada entre algunas familias ya conocidas, como por ejemplo, NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb, PD40-NEWWD40 y Arm-HEAT; b) permite diferenciar las familias control de las otras familias; c) dentro de las familias control, permite separar entre uniform y scrambled; y d) la familia mix se encuentra más cercana a scrambled que a uniform. Sin embargo, esta función aún no nos permite discriminar entre familias globulares y repetitivas. Intentaremos, en las próximas secciones, seguir introduciendo pequeños cambios a la función de distancia para lograr este último objetivo.

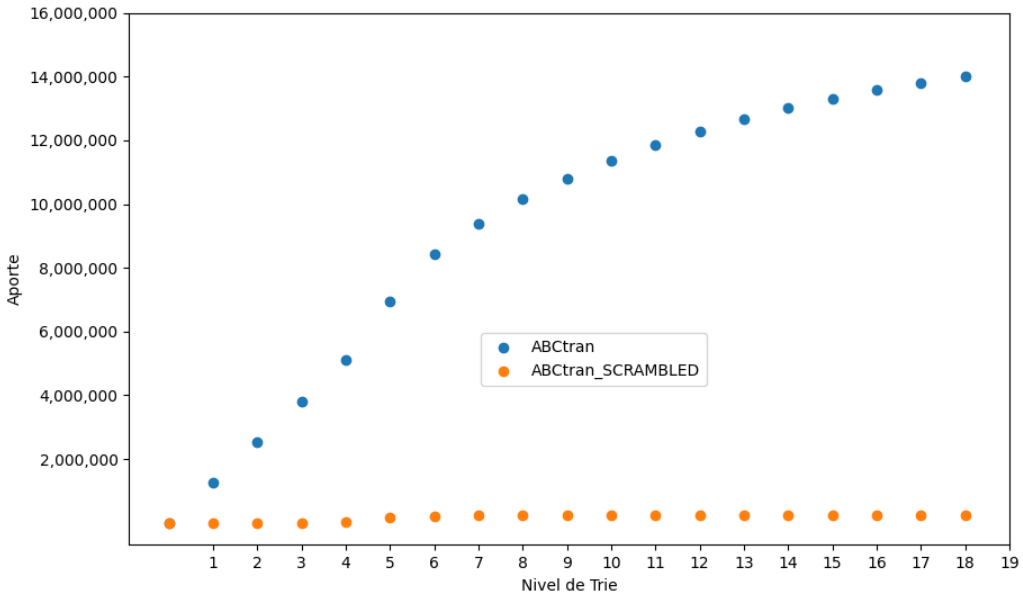


Fig. 2.53: Aporte de cada familia en función del nivel de su trie, para las familias ABCtran (globular) y ABCtran_SCRAMBLED (control).

2.7. distanciaDeTries - Cantidad de Instancias

Con el objetivo de discriminar entre familias globulares y repetitivas, exploramos incorporar a la función de distancia la información correspondiente a la cantidad de instancias con las que se presenta una patrón en sus respectivas familias. Para ello, es suficiente con modificar solamente la construcción del trie. Al momento de incorporar un nuevo patrón al mismo y tener que determinar el valor asociado a cada nodo, se le asigna el máximo valor entre a) el valor que poseía previamente dicho nodo y b) la cantidad de instancias asociada al patrón a incorporar. En caso de no existir aún el nodo en el trie, y por lo tanto no contar con su valor asociado a), se considera que el mismo poseía un valor asociado de 0. Es importante destacar que para computar la distancia, una vez construido el trie, se utiliza el mismo algoritmo de la sección anterior (Sección 2.6 distanciaDeTries - Contador Prefijos Normalizado).

A modo de ejemplo, en la Tabla 2.6, se encuentran los patrones maximales de la familia A y B junto con la cantidad de instancias asociadas a cada uno de ellos. Para el proceso de construcción del trie de la familia A, se puede comenzar por incorporar cualquiera de sus patrones. Comenzamos, de manera arbitraria, por el patrón A. Al mismo se le asocia el valor 10 (previamente no había ningún nodo, por lo tanto le asignamos el máximo entre 0 y la cantidad de instancias de dicho patrón, es decir, $max(0, 10) = 10$). Luego incorporamos el patrón AAA. La primer A de dicho patrón tiene asociado un valor de 4 instancias y la A del nodo previamente incorporado al trie, como vimos, posee un valor de 10. Por lo tanto, a dicho nodo se le asigna $max(4, 10) = 10$. Los restantes nodos AA, al no superponerse con otros nodos, se les asigna $max(0, 4) = 4$. Con los patrones AGT y GT se procede de manera similar. El trie final, se puede observar en la Figura 2.54.

Conjunto de patrones maximales de repetición			
Familia A	**Instancias**	**Familia B**	**Instancias**
A	10	AGT	5
AAA	4	CTS	3
AGT	6		
GT	8		

Tab. 2.6: Conjuntos de patrones maximales de repetición de las familias A y B, junto con sus instancias.

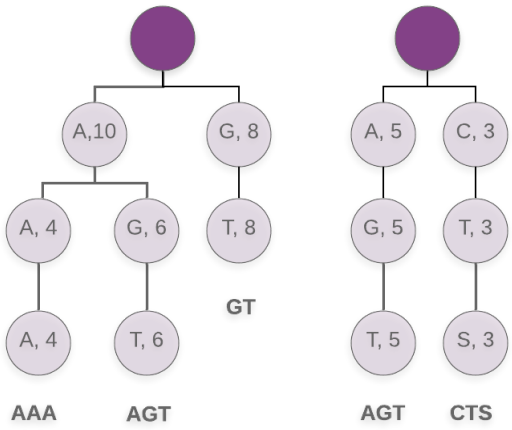


Fig. 2.54: Trie A (izquierda) y Trie B (derecha) correspondientes a los conjunto de patrones maximales de repetición de la Familia A y B, junto con sus instancias de repeción, respectivamente.

Para calcular el valor de nuestra nueva función de distancia, a la que denominaremos *distanciaDeTriesInstancias*, debemos proceder igual que la función *distanciaDeTriesContadorPrefijosNormalizado*, es decir, teniendo en cuenta el valor que posee asociado cada nodo. En el caso de la Figura 2.54, al contar las diferencias entre ambos tries, obtenemos: a) 31 instancias del Trie A que no pertenecen al B (5 de la primer A,4+4 de las dos AA siguientes, 1+1 del subpatrón GT que depende de A y 8+8 del patrón GT que depende de la raíz); b) 9 nodos del Trie B que no pertenecen al Trie A (3+3+3 correspondientes a los nodos del patrón CTS). La función *diferenciaDeTriesInstancias* tiene la suma de ambos valores. En el ejemplo de la Figura 2.54 *diferenciaDeTriesInstancias(TrieA,TrieB) = 40.*

Para obtener la distancia (*distanciaDeTriesInstancias*), simplemente dividimos el valor resultante de *diferenciaDeTriesInstancias* por la sumatoria de los valores asociados a los nodos de cada Trie. En el ejemplo de la Figura 2.54, la sumatoria del Trie A es de 46 y la del Trie B es 24. Al sumarlos obtenemos un total de 70. Así, *distanciaDeTriesInstancias(Trie A, Trie B)*= $\frac{40}{70}$.

Los algoritmos de *distanciaDeTriesInstancias* y *diferenciaDeTriesInstancias* no se presentan ya que son equivalentes a *distanciaDeTriesContadorPrefijosNormalizado* y *diferenciaDeTriesContadorPrefijosNormalizado* de la sección anterior.

Aplicamos la nueva función *distanciaDeTriesInstancias* a las familias de proteínas de

nuestro caso de estudio. En la figura 2.55 se muestran los resultados obtenidos utilizando la visualización de un mapa de calor. Se pueden ver agrupadas las familias de control. También parecerían observarse a corta distancia NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb, PD40-NEWWD40 y Arm-HEAT. Además de estas familias, ahora hay más familias cercanas entre sí, un ejemplo de ello es el caso de PUD-PeptidaseC25.

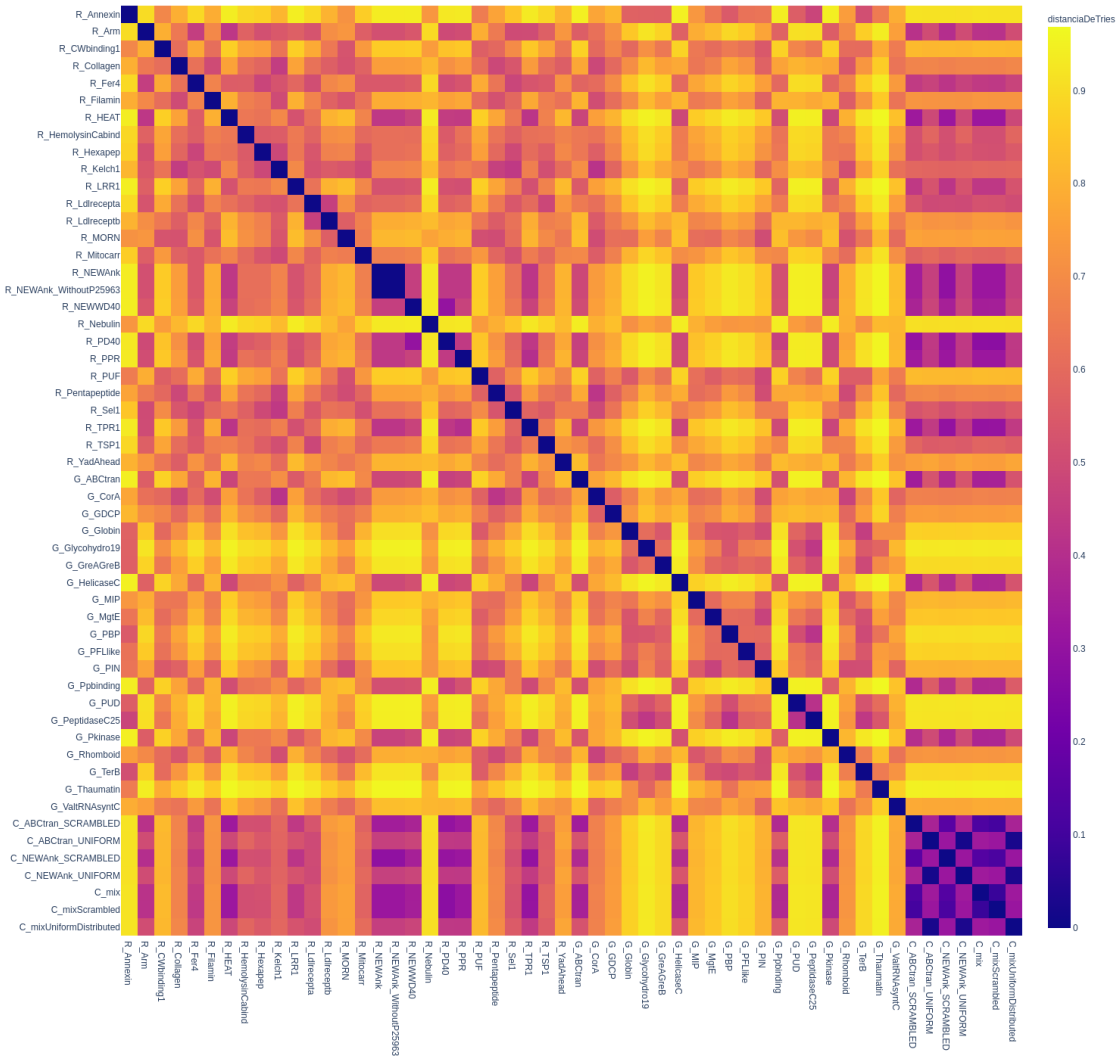


Fig. 2.55: Mapa de calor en base a la función *distanciaDeTriesInstancias* aplicada entre todos los pares posibles de familias de nuestro caso de estudio.

En la Figura 2.56, se muestra el dendograma que surge de aplicar la función *distanciaDeTriesInstancias* y la función *complete* de la biblioteca *scipy.cluster.hierarchy.linkage*. Se puede observar una separación en 2 grandes grupos (verde y rojo). Dentro del grupo verde se encuentran, en una subrama separada, las familias de control. En particular, se destacan dos subramas: a) una con las familias uniform y b) otra con las scrambled y mix. Notar que la familia mix se encuentra cercana a su homónima scrambled. Lamentablemente, el dendograma no muestra una separación entre familias globulares y repetitivas. También

notamos que las familias Arm-HEAT se encuentran un poco más separadas, comparado con lo observado en secciones anteriores. Incluso las familias Ldlrecepta-Ldlreceptb se encuentran en grupos distintos (verde y rojo, respectivamente).

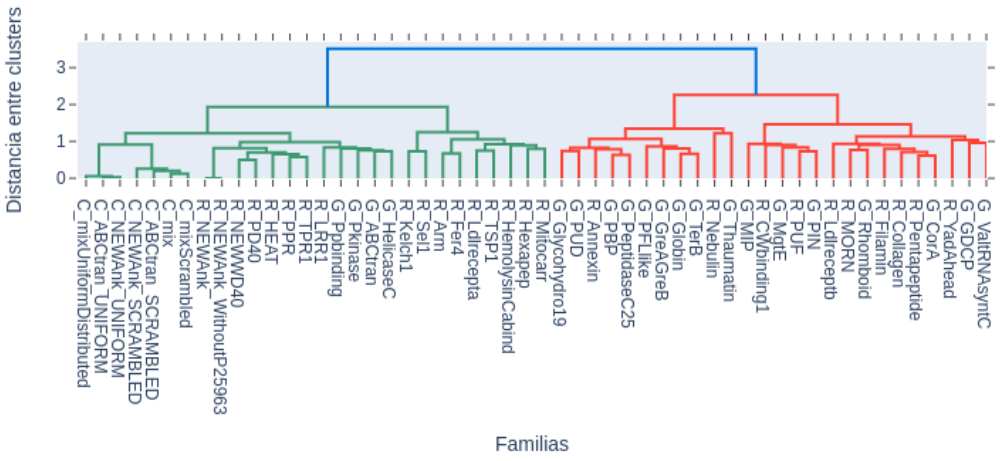


Fig. 2.56: Dendograma en base a valores de *distanciaDeTriesInstancias* para todos los pares posibles de familias de nuestro caso de estudio

En la Figura 2.57 se muestra una combinación del mapa de calor con el dendograma. Se puede observar claramente el agrupamiento de a) las familias de control; b) NEWAnk-NEWAnk_WithoutP25963 y c) NEWWD40-PD40. Por otro lado, se puede notar que Ldlrecepta y Ldlreceptb, si bien se encuentran en grupos distintos (verde y rojo del dendograma, respectivamente), la distancia entre sí no es grande (violeta, en el mapa de color). La separación es a causa de la distancia de cada una de ellas con las restantes familias. Finalmente, si bien no existe una separación determinante entre familias globulares y repetitivas, hay subramas que las agrupan. Este es el caso del grupo de familias globulares que van desde Kelch1 hasta Mitocarr, y el de las familias repetitivas que van desde NEWAnk hasta TPR1.

2.7.1. Discusión

La función de distancia propuesta mostró ciertas características satisfactorias: a) refleja la cercanía esperada entre algunas familias ya conocidas, como por ejemplo, NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb,y PD40-NEWWD40; b) permite diferenciar las familias control de las otras familias; c) dentro de las familias control, permite separar entre uniform y scrambled; y d) la familia mix se encuentra más cercana a scrambled que a uniform. En cuanto a la separación entre familias globulares y repetitivas, no logra separarlas totalmente, pero empieza a notarse un agrupamiento entre ciertas familias del mismo tipo.

Un punto que no mencionamos previamente, pero que es importante y que queda como trabajo a futuro a resolver, es qué sucede al momento de determinar el valor del nodo del trie cuando el prefijo de un patrón no pertenece al conjunto de entrada. Sería el caso, por ejemplo, de que en la Tabla 2.6 el patrón A no formara parte del conjunto de patrones, ¿qué

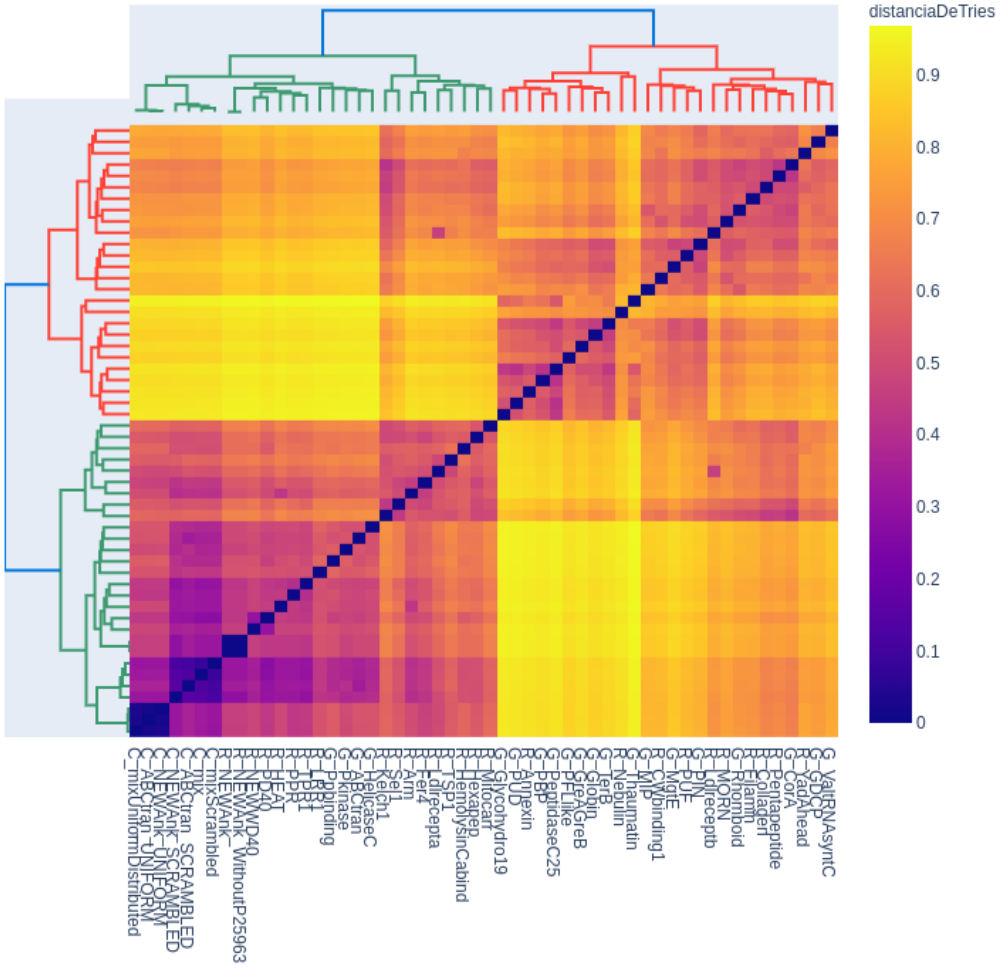


Fig. 2.57: Mapa de calor y dendograma en base a valores de *distanciaDeTriesInstancias* para todos los pares posibles de familias de nuestro caso de estudio.

valor deberíamos asignarle al primer nodo A del trie? Según nuestro método, deberíamos asignarle el valor $max(4, 6) = 6$, correspondiente al máximo de instancias entre los patrones AAA y AGT. Sin embargo, sabemos que la letra A ocurre más veces en nuestra secuencia (al menos 4+6=10 veces, seguro). De esta manera, como se vio en el ejemplo, el trie no estaría reflejando totalmente la estructura de conformación de cada familia, y por lo tanto, podría estar influyendo en la separación entre ellas.

2.8. distanciaDeTries - Proteínas afectadas

Continuando con nuestro objetivo de discriminar entre familias globulares y repetitivas, exploramos incorporar a la función de distancia la información correspondiente a la cantidad de proteínas, dentro de la misma familia, en las que aparece el patrón maximal

de repetición. A esta cantidad la denominaremos *cantidad de proteínas afectadas.*

La mecánica, para implementar este cambio en la función de la sección anterior, consiste simplemente en, al momento de construir el trie, trabajar con el valor de cantidad de proteínas afectadas en vez de cantidad de instancias del patrón. El resto de la metodología se mantiene exactamente igual. A esta nueva función la denominaremos *distanciaDeTries-ProteinasAfectadas*

Aplicamos la nueva función *distanciaDeTriesProteinasAfectadas* a las familias de proteínas de nuestro caso de estudio. En la figura 2.58 se muestran los resultados obtenidos utilizando la visualización de un mapa de calor. Como primer resultado, se puede observar que las familias se encuentran mucho más alejadas entre sí que lo visto en secciones anteriores (mapa de calor más amarillo). Otro punto a notar es que se pueden ver agrupadas las familias de control. También parecerían observarse a corta distancia NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb, PD40-NEWWD40 y Arm-HEAT. Además se destacan algunas nuevas relaciones como Kelch1-Filamin y HemolysinCabind-Filamin.

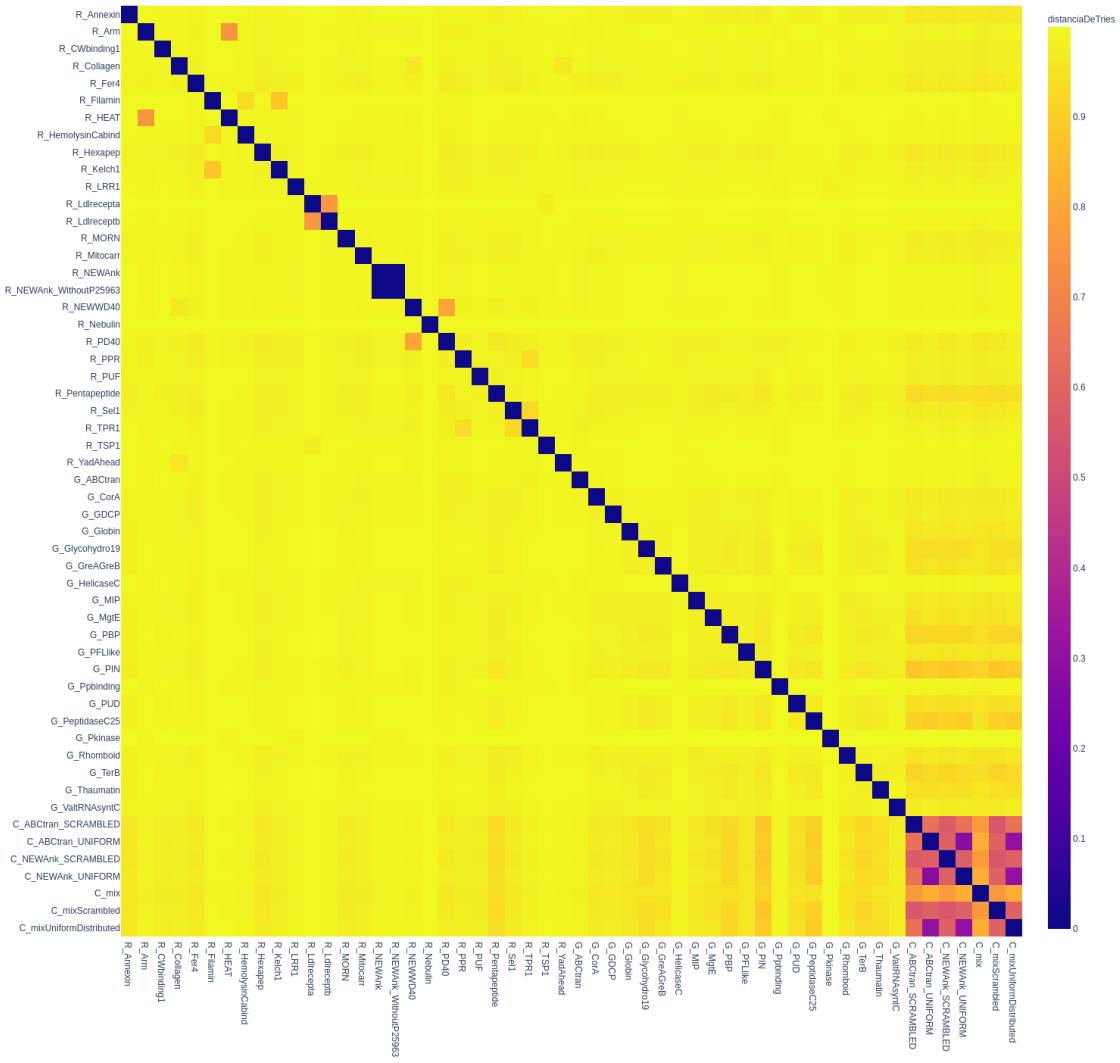


Fig. 2.58: Mapa de calor en base a la función *distanciaDeTriesProteinasAfectadas* aplicada entre todos los pares posibles de familias de nuestro caso de estudio..

En la figura 2.59 se puede observar el dendograma en base a la función *distanciaDeTriesProteinasAfectadas* y la función *complete* de la biblioteca *scipy.cluster.hierarchy.linkage* para la distancia entre grupos. En el mismo se destacan 4 grupos. Por un lado, se agrupan las familias de control con excepción de mix (color verde). Dentro de esta rama (verde) se observan dos subconjuntos, por un lado las uniform y por el otro las scrambled. Luego, se observa un segundo y pequeño grupo conformado por NEWAnk y NEWAnk_WithoutP5963 (color rojo). Siguiendo, notamos un tercer grupo (celeste), en donde encontramos a la familia de control mix, sólo dos familias repetitivas (Pentapeptide y Annexin) y 13 familias restantes de tipo globular. Por último, se observa un grupo púrpura, donde encontramos sólo 7 familias globulares (Pkinase, PPbinding, ABCtran, HelicaseC, ValtRNAsyntC, GDCP, y CorA) y 23 familias repetitivas.

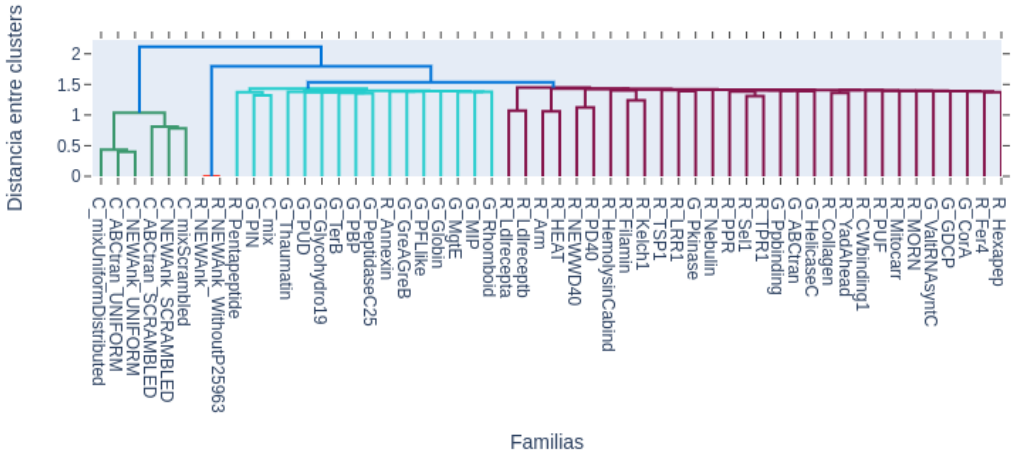


Fig. 2.59: Dendograma en base a valores de *distanciaDeTriesProteinasAfectadas* para todos los pares posibles de familias de nuestro caso de estudio

En la Figura 2.60 se muestra una combinación del mapa de calor con el dendograma. Como mencionamos previamente, se puede observar que, a diferencia de la función de distancia de la sección anterior (*distanciaDeTriesInstancias*), esta nueva función de distancia arroja valores muy altos (color amarillo) para la mayoría de las comparaciones entre familias. Esto no nos permite ser muy contundentes al momento de tener que tomar la decisión de separar entre familias, salvo para el caso de las familias control uniform y scrambles, donde la diferencia con las restantes familias es notoria (color violeta).

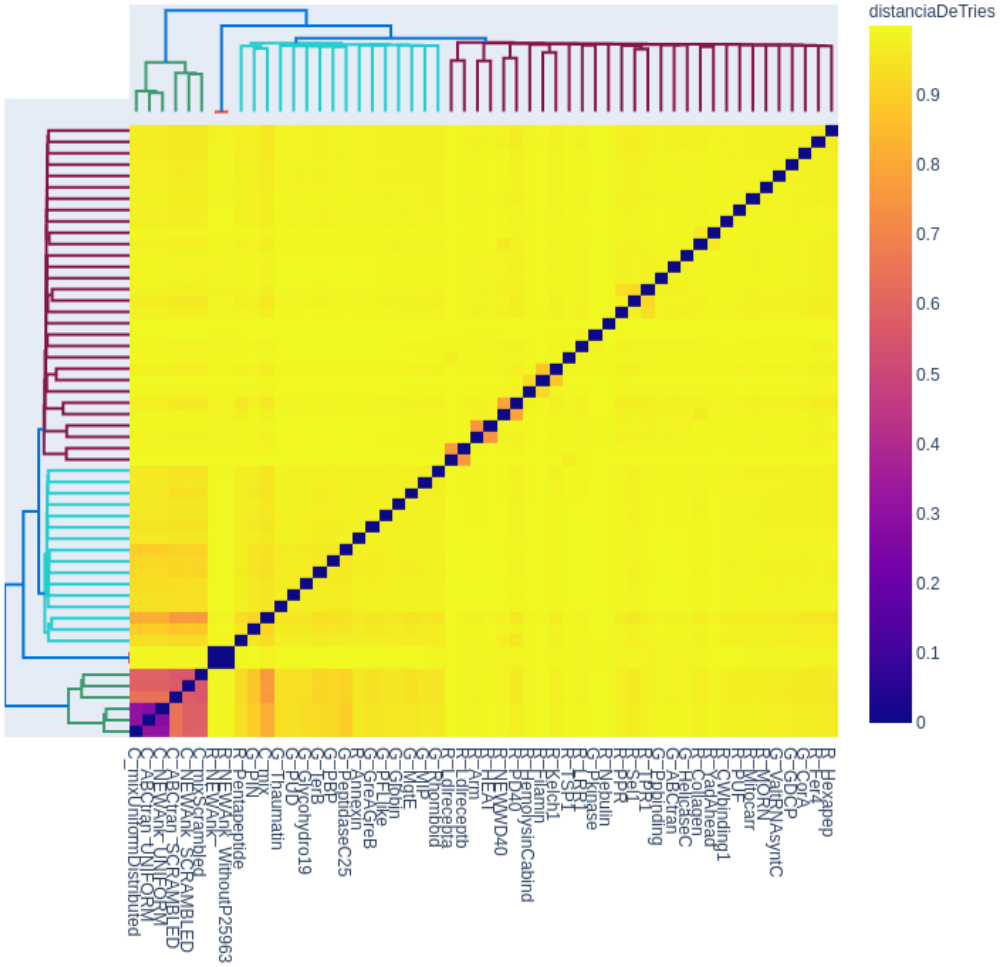


Fig. 2.60: Mapa de calor y dendograma en base a valores de *distanciaDeTriesProteinasAfectadas* para todos los pares posibles de familias de nuestro caso de estudio.

2.8.1. Discusión

La relación de distancia, considerando la cantidad de proteínas afectas por un patrón, mostró ciertas características satisfactorias: a) refleja la cercanía esperada entre algunas familias ya conocidas, como por ejemplo, NEWAnk-NEWAnk_WithoutP25963, Ldlrecepta-Ldlreceptb,y PD40-NEWWD40; b) permite diferenciar las familias control (uniform y scrambled) de las otras familias; c) dentro de las familias control, permite separar entre uniform y scrambled,

Sin embargo, mostró algunas características no esperadas. En primer lugar, con esta función, la familia mix se encuentra dentro de un grupo distinto al de las familias control (uniform y scrmabled). En segundo lugar, los valores arrojados son muy altos para la mayoría de las comparaciones entre familias, lo cual nos impide ser muy contundentes al momento de tener que tomar la decisión de separar entre familias, salvo para el caso de

las familias control uniform y scrambled, donde la diferencia con las restantes familias es notoria. Un posible trabajo a futuro podría ser estudiar qué pasa si en vez de utilizar patrones de repetición maximales nesteados (NE) se utilizan los no nesteados (NN). Esto podría cambiar los resultados ya que se espera que, dependiendo de la familia, estos últimos afecten a distintas cantidades de proteínas.

3. CONCLUSIONES

En el presente trabajo se tuvo como objetivo específico proponer una función de distancia entre familias de proteínas, en base a los prefijos de sus repeticiones maximales, que permita: a) discriminar entre familias naturales y las generadas/agrupadas al azar; y b) dentro de las familias naturales, nos permita agrupar las familias repetitivas por un lado y las globulares por otro.

Para alcanzar ese objetivo, decimos trabajar en base a los prefijos de las repeticiones maximales. Se buscó y se seleccionó una estructura de datos para contenerlos y procesarlos. La estructura elegida fue el Trie.

En una primera etapa, se buscó comparar dos estructuras de Trie con algoritmos ya existentes en el área de redes de computadoras. Para ello, se analizaron los algoritmos de *Deltacon*, *Gmatch4pyeg* y *Networkx*. Los resultados obtenidos no fueron satisfactorios. Los motivos fueron variados. En el caso de *Deltacon*, una proyección de los tiempos de cómputo requeridos para el tamaño de nuestros tries arrojó que no están dentro de lo aceptable. Adicionalmente, esta herramienta requiere que los tries a comparar posean la misma cantidad de nodos, lo cual no es algo que podamos garantizar. En el caso de *Gmatch4pyeg*, los valores de salida resultaron ser multidimensionales, cuando en realidad nosotros necesitamos una respuesta unidimensional. Finalmente, en el caso de *Networkx*, el consumo de memoria resultó ser excesivo para los tamaños de trie asociados a nuestras familias. De esta manera, decidimos seguir por otro camino.

Como segunda etapa de este trabajo, se optó por definir una función propia de distancia para comparar dos estructuras de Trie. Se exploraron varias alternativas:

- $distanciaDeTries$
- $distanciaDeTriesContadorPrefijos$
- $distanciaDeTriesContadorPrefijosNormalizado$
- $distanciaDeTriesInstancias$
- $distanciaDeTriesProteinasAfectadas$

El rango de valores definido para la imagen de estas funciones de distancia, en todos los casos se restringió al intervalo $[0, 1]$, salvo para la función *distanciaDeTriesContadorPrefijos* la cual si bien no se restringió a dicho intervalo, nos sirvió de puntapié para comenzar a estudiar los patrones en base a las repeticiones de sus prefijos.

En la mayoría de las funciones de distancias se logró el objetivo específico de *a) discriminar entre familias naturales y las generadas/agrupadas al azar*. En *distanciaDeTries*, *distanciaDeTriesContadorPrefijosNormalizado* y *distanciaDeTriesInstancias* se observaron valores de distancia más fuertemente marcados, permitiendo agrupar a la totalidad

de las familias control en una rama separada del dendograma. En el caso de *distanciaDeTriesProteinasAfectadas* las distancias fueron menos marcadas, generando que la familia de control mix se posicionara en una sub-rama mezclada entre las ramas de las familias naturales, quedando separada de las de tipo SCRAMBLED y UNIFORM. En el caso de *distanciaDeTriesContadorPrefijos* no fue posible discriminar entre familias control de naturales.

En lo que respecta al objetivo específico correspondiente a que *b) la función de distancia, dentro de las familias naturales, nos permita agrupar las familias repetitivas por un lado y las globulares por otro,* no pudo ser logrado. En el caso de *distanciaDeTries*, las familias naturales se separan en prácticamente dos bloques principales, sin embargo dentro de estos bloques aparecen tanto familias repetitivas como globulares. La función *distanciaDeTriesContadorPrefijos*, comienza con un problema previo y es que desde un inicio no separa en dos bloques las familias control de las naturales. La función *distanciaDeTriesContadorPrefijosNormalizado* divide a las familias naturales en tres bloques, agrupando en uno de ellos un número alto de familias repetitivas, lo cual es bastante prometedor. Sin embargo, en los otros dos bloques restantes se mezclan bastante las globulares con las repetitivas remanentes, lo cual nos aleja de nuestro objetivo buscado. En el caso de la función *distanciaDeTriesInstancias*, tampoco se observa una separación de las familias naturales en globulares y repetitivas. Por último, en el caso de la función *distanciaDeTriesProteinasAfectadas*, se separa en 3 grandes bloques y uno muy pequeño. En el primero de los 3 bloques, se encuentran las familias de control, con excepción de mix que se encuentra en el segundo de los 3 bloques. En el segundos de los 3 bloques se observa un alto porcentaje de familias globulares (81 %) y un bajo porcentaje de repetitivas (12 %). En el último de los 3 bloque, encontramos a la mayoría de las familias repetitivas (77 %) y un bajo porcentaje de familias globulares (23 %). En vistas de estos resultados, esta última función parecería acercarnos mejor a nuestro objetivo. Sin embargo, la poca diferencia que existe entre los valores de distancias no nos permite ser categóricos con esta elección.

Es importante mencionar que tanto para *distanciaDeTriesInstancias* como para *distanciaDeTriesProteinasAfectadas*, al momento de construir los tries, se tomó el máximo entre el valor contenido en el nodo del patrón existente y el valore del nuevo patrón agregado. Esto puede llegar a generar una variación en los valores absolutos de diferencia entre los tries de las familias. Como trabajo futuro, se pueden analizar otras opciones, como por ejemplo tomar la suma en vez del valor máximo.

En vista de los resultados obtenidos se puede considerar la posibilidad de que para lograr el objetivo de separar las familias de proteínas globulares de las repetitivas, no alcance sólo con los patrones ya que los mismos quizás no posean suficiente información. Quizás, para poder distinguirlas falte, a modo de ejemplo, información acerca del *código de plegado*; o quizás sea necesario utilizar otra representación del alfabeto. Queda como trabajo futuro pensar alternativas para poder lograr incorporar nueva información.

4. TRABAJOS FUTUROS

Durante el desarrollo del presente trabajo surgieron varias ideas y caminos alternativos que, por cuestiones de tiempo, no llegaron a ser explorados. A continuación mencionaremos los más relevantes, con el objeto de que puedan ser explorados en trabajos futuros.

4.1. Función de distancia entre una proteína y una familia de proteínas

En el presente trabajo nos abocamos a realizar la comparación entre familias de proteínas. Queda como trabajo a futuro, una vez extraídas las características de una familia, proponer una función de distancia que permita determinar qué tan cerca está una única proteína de una determinada familia. El objetivo final es que esta función permita descartar secuencias aleatorias de aminoácidos de proteínas funcionales.

4.2. Análisis de niveles de patrones completos

En una parte del trabajo se propuso como alternativa representar Tries contemplando la marca de nodos finales (ver figura 2.4). Al momento de realizar un análisis por niveles, se puede introducir la variante de considerar solamente nodos correspondientes a patrones completos, es decir, que contengan el nodo final dentro del rango del nivel especificado.

A modo de ejemplo, volviendo al Trie de la figura 2.4, se puede observar que si se recorta el Trie a nivel 2, hay una única rama que posee un nodo final en dicho nivel (patrón *EA*), por lo tanto, en este caso se puede contemplar sólo esta rama para el cálculo de distancia.

4.3. Estudio de la función de distancia en base al tamaño del conjunto de patrones

En nuestro caso de estudio, el tamaño del conjunto de patrones maximales difiere de familia en familia. La cantidad de patrones dentro de una familia podría llegar a influenciar al valor de distancia obtenido. ¿Qué ocurre en el caso de comparar 2 familias donde, una posee un conjunto de patrones muy grande, y la otra uno muy pequeño? ¿Está más desfavorecida en el cálculo de distancia una familia con un conjunto pequeño de patrones frente a otra cuyo conjunto es mucho más grande? Este es un tema que queda pendiente de estudio.

4.4. Asignación de valores a nodos del Trie

En la construcción del Trie, al asignar el valor de un nodo compartido entre dos patrones, tomamos la decisión de asignarle el máximo valor de ambos. Esta determinación puede ser modificada utilizando otra función, como por ejemplo, la suma de ambos valores. Esto, de seguro, impactará en los valores de distancia obtenidos. Queda como trabajo futuro analizar su impacto.

4.5. Definición de Métrica de distancia de familias de Proteínas

Finalmente, nos quedó pendiente, trabajar sobre las funciones de distancia propuestas para que cumplan las propiedades propias de ser una métrica.

5. ANEXO

5.1. Algoritmo para la identificación y clasificación de MR (SMR, NN, NE)

A continuación se transcribe el algoritmo utilizado para la obtención de los patrones maximales de las familias de proteínas. Dicho algoritmo se obtuvo del trabajo [9].

Require: s , SA , LCP
procedure PROCESS INTERVAL(l, i, j)
$\sigma \leftarrow$ s[SA[i]−1] //Any pattern left context **to** next
//check **if** they are all the same
is_MR $\leftarrow$ (σ = separatorCharacter) // is_MR flag initialization.
// If σ is a separator then
// it's a MR
// Survey of left & right contex
LCCT, RCCT $\leftarrow$ empty dictionaries
for k **from** i **to** j
σ L $\leftarrow$ s[SA[k]−1]; //left context
σ R $\leftarrow$ s[SA[k]+l]; //right context
LCTT $\leftarrow$ **addOccurrence**(LCTT, σ_L);
RCTT $\leftarrow$ **addOccurrence**(RCTT, σ_R);
//l−interval always contains at least one element
//that is not−right extensible (by definition).
//We only need **to** check **if** it's not left−extensible
if(σ != σ_L)
is_MR $\leftarrow$ true; //It is not left−extensible,
//then it is MR
end if
end for

if (is_MR) //is it SMR, NN o r NE?
//MR is a SMR?
if (∃! any element **from** LCTT or RCTT > 1)
// There is not pair of occurrences that has the same same
// left or right−context
// Then, they can't be extended
PROCESS_MR_SMR // It's SMR
else
// is it NN o r NE?
// Find an occurrence that is not right And left extensible
// If it exist then it is a NN
isNested $\leftarrow$ true ;
for k **from** i **to** j //For each occurrence ...
$\sigma_L \leftarrow$ s[SA[k]−1]; //left context

```
                                σR ← s[SA[k]+l]; //right context
                                if (LCTT[σL] ≤ 1 and RCTT[σR] ≤ 1)
                                        // This occurrence is not right and left extensible
                                        isNested ← false;
                                end if
                        end for
                        if (is_Nested)
                                PROCESS_MR_NE; //It is a NE
                                                // ( all pattern ocurrences are nested )
                        else
                                PROCESS_MR_NN; //It is a NN
                                                //(at least one pattern occurrence
                                                //is non−nested)
                        end if
                end if
end if
end procedure
```

Listing 5.1: Algoritmo para la identificación y clasificación de MR (SMR, NN y NE) de un intervalo de LCP del conjunto de secuencias concatenadas S.

```
function addOccurrence (CTT, σ )
        if ( σ ∈ keys (CTT) )
                if ( σ != separatorCharacter )
                        CTT[σ] ← CTT[σ]+1 //separatorCharacter can only
                end if // have one instance
        else
                CTT[σ] ← 1
        end if
        return CTT
end function
```

Listing 5.2: Función para mantener la cantidad de elementos de la estructura de datos de diccionario.

5.2. Conjunto de Datos

Para este trabajo se utilizó el conjunto de datos del trabajo [9], el cual se compone de 47 familias de proteínas correspondientes tanto a familias repetitivas como a familias globulares.

La familia *NEWAnk_WithoutP25963*, es una familia de control basada en la familia *NEWAnk*, quitando sólamente la proteína *P25963*. Las restantes familias de control se componen de *ABCtran_SCRAMBLED*, *NEWAnk_SCRAMBLED*, *mixSCRAMBLED*, *ABCtran_UNIFORM*, *NEWAnk_UNIFORM*, *mixUniformDistributed* y *mix*. Las SCRAMBLED fueron generadas desordenando, dentro de las proteínas de las familias de referencia, sus aminoácidos. Las UNIFORM fueron generadas reemplazando, dentro de las proteínas, los aminoácidos por una selección al azar (uniforme) de los 20 aminoácidos más comunes. La familia mix se corresponde con una selección arbitraria de proteínas de distintas familias.

Finalmente, nuestro conjunto de datos cuenta con un total de 54 familias. Una descripción de cada una de ellas se puede observar en las tablas 5.1, 5.2, y 5.3

Tipo de Familia	Familia	# secuencias	#aminoácidos	Pfam ID
Familias de proteínas Repetitivas	Annexin	3,424	1,290,719	PF00191
	Arm	19,821	13,749,326	PF00514
	Collagen	14,747	6,014,509	PF01391
	CWbinding1	5,403	3,046,292	PF01473
	Fer4	30,523	13,273,813	PF00037
	Filamin	4,137	5,001,345	PF00630
	HEAT	25,339	23,309,958	PF02985
	HemolysinCabind	12,515	11,975,885	PF00353
	Hexapep	37,258	1,925,761	PF00132
	Kelch1	12,700	7,685,323	PF01344
	Ldlrecepta	12,312	12,331,937	PF00057
	Ldlreceptb	4,233	5,206,338	PF00058
	LRR1	33,060	23,086,490	PF00560
	Mitocarr	26,363	9,053,755	PF00153
	MORN	8,079	4,149,383	PF02493
	Nebulin	1,062	1,529,165	PF00880
	NEWAnk	32,169	23,086,941	PF00023
	NEWWD40	30,898	23,454,445	PF00400
	PD40	32,370	22,089,636	PF07676
	Pentapeptide	15,325	5,580,452	PF00805
	PPR	38,342	23,191,631	PF01535
	PUF	3,956	2,903,554	PF00806
	Sel1	21,424	9,599,210	PF08238
	TPR1	36,389	23,631,180	PF00515
	TSP1	11,296	10,503,271	PF00090
	YadAhead	3,933	4,386,128	PF05658

Tab. 5.1: Resumen de familias de proteínas repetitivas utilizadas en este trabajo.

Tipo de Familia	Familia	# secuencias	#aminoácidos	Pfam ID
Familias de proteínas Globulares	ABCtran	27,442	23,427,524	PF00005
	CorA	13,398	5,792,561	PF01544
	GDCP	8,371	4,872,335	PF02347
	Globin	6,606	1,956,835	PF00042
	Glycohydro19	3,363	1,079,071	PF00182
	GreAGreB	8,643	1,556,658	PF01272
	HelicaseC	37,928	23,670,587	PF00271
	MgtE	5,518	2,291,740	PF01769
	MIP	11,824	3,179,802	PF00230
	PBP	6,734	1,456,841	PF01161
	PeptidaseC25	1,392	1,242,874	PF01364
	PFLlike	3,068	1,932,072	PF02901
	PIN	21,705	3,147,108	PF01850
	Pkinase	27,797	22,788,340	PF00069
	PPbinding	4,230	23,564,300	PF00550
	PUD	924	1,155,557	PF03714
	Rhomboid	15,532	4,732,255	PF01694
	TerB	6,952	1,668,805	PF05099
	Thaumatin	2,827	805,684	PF00314
	ValtRNAsyntC	4,782	4,070,462	PF10458

Tab. 5.2: Resumen de familias de proteínas globulares utilizadas en este trabajo.

Tipo de Familia	Familia	# secuencias	#aminoácidos	Pfam ID
Familias de control	NEWAnk_WithoutP25963	32,168	23,086,624	PF00023
	ABCtran_SCRAMBLED	27,442	23,427,524	
	NEWAnk_SCRAMBLED	32,169	23,086,941	
	mixSCRAMBLED	70,859	24,028,359	
	ABCtran_UNIFORM	27,442	23,427,524	
	NEWAnk_UNIFORM	32,169	23,086,941	
	mixUniformDistributed	70,866	24,031,001	
	mix	70,859	24,028,359	

Tab. 5.3: Resumen de familias de proteínas de control utilizadas en este trabajo.

5.3. Manual de usuario del código provisto

Los scripts y las fuentes correspondiente al código generado para el presente trabajo se encuentra disponible a través de la herramienta [25], dentro del repositorio `https://gitlab.com/agusciraco/tesis-grado`, con acceso público.

5.3.1. Código principal

El código principal desarrollado para este trabajo consta de un programa en lenguaje C++. Para su compilación, contiene un archivo MAKE que se ejecuta con la siguiente línea de comando:

```
make main
```

Luego, para ejecutarlo, es suficiente con correr el ejecutable (resultado del paso previo del make), de la siguiente manera:

```
./main
```

Descripción: Código completo del presente trabajo. Contiene tanto las funciones de identificación y clasificación de MR, como las funciones de distancia. Para la ejecución de cada una de estas últimas, requiere agregar el *main* correspondiente a la función main. Para facilitar dicho trabajo, se provee de distintas carpetas fijando el main a correr. A modo de ejemplo, para el estudio en niveles, se encuentra disponible el main correspondiente dentro de la carpeta *By level*
Entrada: No toma parámetros de entrada, pero requiere que se especifique la carpeta con el conjunto de datos a utilizar.
Salida: El programa escribe los resultados de la ejecución en archivos de formato .csv .

5.3.2. Scripts

Para generar los gráficos presentados de este trabajo, es decir, mapas de calor, dendogramas, funciones de ajuste, entre otros, desarrollamos ciertos scripts. Dejamos dichos scripts a disposición.
En el caso de los mapas de calor y dendogramas, se provee la versión paramétrica del script, la cual cuenta con los siguientes parámetros:
Entrada: Toma como parámetro el nombre del archivo en formato .csv, con los datos a graficar.
Salida: El programa genera un archivo .html con el gráfico, el cual se puede descargar.

Los scripts disponibles son: **heatmap.py, heatmapByLevel.py, dendogramPlotterParametric.py y dendoAndheatmapPlotterParametric.py**

En el caso de las funciones de ajuste, se provee un archivo **main.py** con las funciones utilizadas y, dentro de la función main, ejemplos de su ejecución.

5.3.3. Estudio de Redes de Computadora

Dejamos a disposición los códigos fuente descargados de las distintas herramientas utilizadas para el análisis de la sección 2.2 y los códigos fuente utilizados para el análisis de tiempos de ejecución.

Entrada: Toma como parámetro dos archivos con grafos a comparar.
Salida: El resultado de la comparación de los grafos según la herramienta.

BIBLIOGRAFíA

[1] D.L. Nelson, A.L. Lehninger y M.M. Cox. *Lehninger Principles of Biochemistry*. Lehninger Principles of Biochemistry. W. H. Freeman, 2008. ISBN: 9780716771081. URL: https://books.google.com.ar/books?id=5Ek9J4p3NfkC.

[2] A. Schnek. *Curtis. Biologıa*. 2008. URL: https://books.google.com.ar/books?id=JCXVvQEACAAJ.

[3] *Biological Proteins*. https://www.visionlearning.com/en/library/Biology/2/Biological-Proteins/243. Accessed: 2020-11-21.

[4] *Fasta Substitution codes*. https://web.cas.org/help/BLAST/topics/codes.htm. Accessed: 2020-11-21.

[5] *Single letter amino acid codes*. https://teaching.ncl.ac.uk/bms/wiki/index.php/Single_letter_amino_acid_codes. Accessed: 2020-11-21.

[6] Andrey V. Kajava. «Tandem repeats in proteins: From sequence to structure». En: *Journal of Structural Biology* 179.3 (2012). Structural Bioinformatics, págs. 279-288. ISSN: 1047-8477. DOI: https://doi.org/10.1016/j.jsb.2011.08.009. URL: http://www.sciencedirect.com/science/article/pii/S1047847711002371.

[7] Vikram Alva, Michael Remmert, Andreas Biegert, Andrei N. Lupas y Johannes Söding. «A galaxy of folds». eng. En: *Protein science : a publication of the Protein Society* 19.1 (ene. de 2010). PMC2817847[pmcid], págs. 124-130. ISSN: 1469-896X. DOI: 10.1002/pro.297. URL: https://doi.org/10.1002/pro.297.

[8] Pablo Turjanski, R. Gonzalo Parra, Rocío Espada, Verónica Becher y Diego U. Ferreiro. «Protein Repeats from First Principles». En: *Scientific Reports* 6.1 (abr. de 2016), pág. 23959. ISSN: 2045-2322. DOI: 10.1038/srep23959. URL: https://doi.org/10.1038/srep23959.

[9] Pablo Turjanski y Diego U. Ferreiro. «On the Natural Structure of Amino Acid Patterns in Families of Protein Sequences». En: *The Journal of Physical Chemistry B* 122.49 (2018). PMID: 30239207, págs. 11295-11301. DOI: 10.1021/acs.jpcb.8b07206. eprint: https://doi.org/10.1021/acs.jpcb.8b07206. URL: https://doi.org/10.1021/acs.jpcb.8b07206.

[10] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. DOI: 10.1017/CBO9780511574931.

[11] Eddy Taillefer y Jonathan Miller. «Exhaustive Computation of Exact Duplications via Super and Non-Nested Local Maximal Repeats». En: *Journal of Bioinformatics and Computational Biology* 12.01 (2014). PMID: 24467757, pág. 1350018. DOI: 10.1142/S0219720013500182. eprint: https://doi.org/10.1142/S0219720013500182. URL: https://doi.org/10.1142/S0219720013500182.

[12] OLAF WEISS, MIGUEL A JIMÉNEZ-MONTAÑO y HANSPETER HERZEL. «Information Content of Protein Sequences». En: *Journal of Theoretical Biology* 206.3 (2000), págs. 379-386. ISSN: 0022-5193. DOI: https://doi.org/10.1006/jtbi.2000.2138. URL: http://www.sciencedirect.com/science/article/pii/S0022519300921386.

[13] Peter G. Wolynes, William A. Eaton y Alan R. Fersht. «Chemical physics of protein folding». En: *Proceedings of the National Academy of Sciences* 109.44 (2012), págs. 17770-17771. ISSN: 0027-8424. DOI: 10.1073/pnas.1215733109. eprint: https://www.pnas.org/content/109/44/17770.full.pdf. URL: https://www.pnas.org/content/109/44/17770.

[14] William A. Eaton y Peter G. Wolynes. «Theory, simulations, and experiments show that proteins fold by multiple pathways». En: *Proceedings of the National Academy of Sciences* 114.46 (2017), E9759-E9760. ISSN: 0027-8424. DOI: 10.1073/pnas.1716444114. eprint: https://www.pnas.org/content/114/46/E9759.full.pdf. URL: https://www.pnas.org/content/114/46/E9759.

[15] Thomson Andrew R Dryden David T.F y White John H. «How much of protein sequence space has been explored by life on Earth?» En: (2008), págs. 5953-956. DOI: http://doi.org/10.1098/rsif.2008.0085.

[16] Doolittle R.F. «The Roots of Bioinformatics in Protein Evolution.» En: (2010). DOI: https://doi.org/10.1371/journal.pcbi.1000875.

[17] Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S. Marks, Chris Sander, Riccardo Zecchina, José N. Onuchic, Terence Hwa y Martin Weigt. «Direct-coupling analysis of residue coevolution captures native contacts across many protein families». En: *Proceedings of the National Academy of Sciences* 108.49 (2011), E1293-E1301. ISSN: 0027-8424. DOI: 10.1073/pnas.1111471108. eprint: https://www.pnas.org/content/108/49/E1293.full.pdf. URL: https://www.pnas.org/content/108/49/E1293.

[18] Nicholas P. Schafer, Bobby L. Kim, Weihua Zheng y Peter G. Wolynes. «Learning To Fold Proteins Using Energy Landscape Theory». En: *Israel Journal of Chemistry* 54.8-9 (2014), págs. 1311-1337. DOI: https://doi.org/10.1002/ijch.201300145. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ijch.201300145. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ijch.201300145.

[19] Faruck Morcos, Nicholas P. Schafer, Ryan R. Cheng, José N. Onuchic y Peter G. Wolynes. «Coevolutionary information, protein folding landscapes, and the thermodynamics of natural selection». En: *Proceedings of the National Academy of Sciences* 111.34 (2014), págs. 12408-12413. ISSN: 0027-8424. DOI: 10.1073/pnas.1413575111. eprint: https://www.pnas.org/content/111/34/12408.full.pdf. URL: https://www.pnas.org/content/111/34/12408.

[20] Russell J. Dickson, Lindi M. Wahl, Andrew D. Fernandes y Gregory B. Gloor. «Identifying and Seeing beyond Multiple Sequence Alignment Errors Using Intra-Molecular Protein Covariation». En: *PLOS ONE* 5.6 (jun. de 2010), págs. 1-11. DOI: 10.1371/journal.pone.0011082. URL: https://doi.org/10.1371/journal.pone.0011082.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein. *Introduction to Algorithms*. 2.ª ed. The MIT Press, 2001.

[22] D. Koutra, J.T. Vogelstein y C. Faloutsos. «DELTACON: A Principled Massive-Graph Similarity Function». En: (2013). URL: http://scholar.google.de/scholar.bib?q=info:3SNgHa8eBEUJ:scholar.google.com/&output=citation&hl=de&as_sdt=0&ct=citation&cd=1.

[23] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel y Patrick Martineau. «An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems». En: vol. 1. Ene. de 2015. DOI: 10.5220/0005209202710278.

[24] Scott Fortin. *The Graph Isomorphism Problem*. Inf. téc. 1996.

[25] *Gitlab*. `https://about.gitlab.com/`. Accessed: 2020-12-03.