Estudio de redes de regulación transcripcional mediante maximización de la expectación

Daniel Holgado Chicharro

Resumen—Descripción, desarrollo e implementación de nuevas funcionalidades para CGB. CGB es un software que se basa en el uso de métodos de genómica comparativa para analizar redes de regulación transcripcional. Las nuevas funciones introducidas son la implementación de la obtención automática de genomas para el input de CGB y introducción de algoritmos de alineamientos de secuencias para los motivos de referencia.

Palabras clave—Genómica comparativa, factor de transcripción, redes de regulación transcripcional, CGB, biopython, BLAST, NCBI, E-utilities, ete3, LASAGNA.

Abstract—Description, development and implementation of new functionalities for CGB. CGB is a software that is based on the use of comparative genomics methods to analyse transcriptional regulatory networks. The new functions introduced are the implementation of automatic genome retrieval for CGB input and introduction of sequence alignment algorithms for reference motifs.

Index Terms— Comparative genomics, transcription factor, transcriptional regulatory networks, CGB, biopython, BLAST, NCBI, E-utilities, ete3, LASAGNA.

1 Introducción

La bioinformática es una disciplina científica centrada en la utilización de técnicas computacionales para procesar e interpretar los altos volúmenes de información generados por las nuevas tecnologías en biología. Un aspecto fundamental de la bioinformática es el procesamiento de secuencias genéticas, como el genoma humano, a fin de mejorar nuestro conocimiento sobre los seres vivos y su evolución. En este contexto, un problema fundamental y todavía abierto en bioinformática es el uso de métodos de genómica comparativa para analizar redes de regulación transcripcional (ver apéndice A12). La genómica comparativa se basa en el análisis de múltiples genomas para identificar rasgos comunes que resultan de la conservación de elementos esenciales para la supervivencia. La regulación transcripcional (ver apéndice A3) es el mecanismo más extendido de control de expresión genética en todos los seres vivos. Sin embargo, se han desarrollado pocos métodos de genómica comparativa para el análisis automatizado de estos sistemas regulatorios. CGB [1] es una plataforma flexible para la genómica comparativa de regulones procarióticos. La plataforma automatiza la unificación de información experimental y utiliza un marco teórico bayesiano para generar e integrar resultados fácilmente interpretables. Este TFG tiene como meta implementar varias mejoras a CGB.

2 CGB

Dado algún conocimiento previo sobre la especificidad de unión de un factor de transcripción o TF (ver apéndice A10), los datos de la secuencia genómica se pueden aprovechar para identificar supuestos sitios de unión o TFBS (ver apéndice A11) y reconstruir la red transcripcional, bajo el control de un factor de transcripción dado. En teoría, esto proporciona los medios para dilucidar las redes reguladoras transcripcionales que codifican, lo que arroja información sobre los mecanismos moleculares utilizados por las bacterias para orquestar y coordinar diversos procesos fisiológicos. En la práctica, sin embargo, la naturaleza corta y degenerada de los patrones o motivos de unión de TF conduce a altas tasas de falsos positivos en las búsquedas de todo el genoma, lo que limita su aplicabilidad.

Los métodos de genómica comparativa para la recons-

- *E-mail de contacto: daniel-holgado@hotmail.es*
- *Mención realizada: Tecnologías de la información*
- *Trabajo tutorizado por: Joan Serra Sagrista (Departamento de ingeniería de información y comunicaciones)*
- *Curso 2021/22*

trucción de redes transcripcionales bacterianas explotan la noción de que sólo los sitios de unión a TF funcionales deben conservarse a lo largo de períodos evolutivos sustanciales. Por lo tanto, la identificación de un sitio de unión de TF conservado en la región promotora de dos o más operones ortólogos (genes que tienen la misma identidad en distintas especies) debería reforzar intuitivamente nuestra confianza en su predicción como elemento funcional.

Se procede a explicar con cierto nivel de abstracción el flujo de trabajo de la plataforma CGB (véase Figura 1). Si se quiere profundizar más en el entendimiento de la plataforma CGB consultar el artículo de CGB [1]

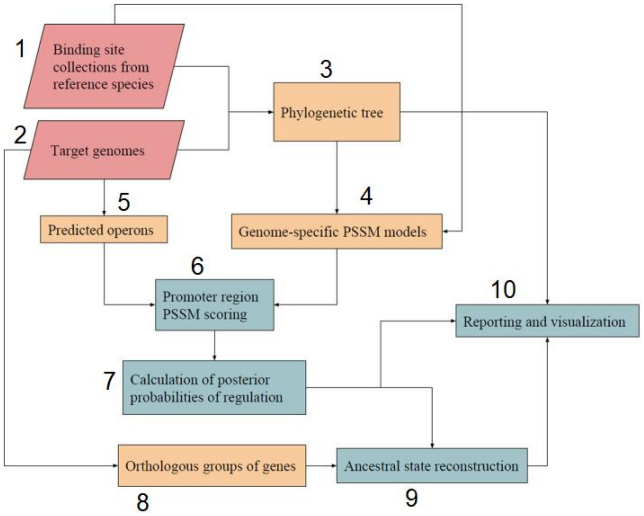

Fig 1. Flujo de trabajo de CGB.

Binding site collections from reference species (1).

La ejecución comienza con la lectura de un archivo de entrada con formato JSON. Este archivo contiene el número de acceso NCBI [2] a la proteína (TF) y la lista de sitios de unión alineados para al menos una instancia de factor de transcripción.

Target genomes (2).

Los genomas diana (target genomes) se representan con números de acceso para cromidos [3] o mapeo de cóntigos [4] de una o más especies objetivo. Con especies objetivo nos referimos a las especies de bacterias en las cuales queremos encontrar TFBS (transcription factor binding sites) con cierta probabilidad para el TF que estamos estudiando. Además de varios parámetros de configuración.

Estas dos secciones anteriores conforman el input de CGB.

Phylogenetic tree (3)

Las instancias TFBS de referencia se utilizan para detectar ortólogos en cada genoma objetivo y un árbol filogenético de instancias TFBS es generado.

Genome-specific PSSM models (4)

El árbol se utiliza para combinar información de TFBS disponible en una PSSM (position-specific scoring matrix) [5] para cada genoma objetivo (véase figura 2).

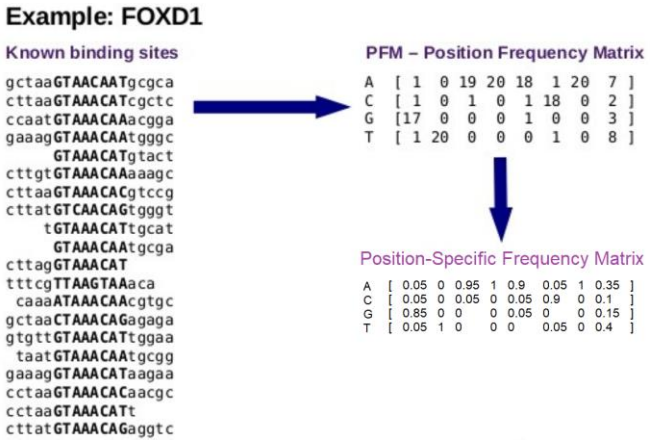

Fig 2. PFM y PSFM, los pasos previos a la creación de una PSSM.

Predicted operons (5)

La distancia intergénica es un predictor de operones eficaz y ampliamente utilizado. Se considera que genes pares en una misma dirección (adyacentes en la misma orientación sin genes intervinientes en la hebra opuesta) pertenecen a un operón si su distancia intergénica está por debajo de un umbral preestablecido. Debido a que diferentes genomas pueden tener diferentes densidades de codificación, CGB define este umbral de manera adaptativa como la distancia intergénica promedio en todas las direcciones dentro de un genoma dado.

Promoter regions PSSM scoring (6)

Aquí se adopta un marco probabilístico bayesiano. Este marco estima probabilidades posteriores de regulación que son fácilmente interpretables y directamente comparables entre especies. Para cada posición i de la región de un promotor, se combinan los puntajes PSSM obtenidos en ambas hebras.

Calculation of posterior probabilities of regulation (7)

Para estimar la probabilidad posterior de regulación de un promotor, definimos dos distribuciones de puntuaciones de PSSM dentro de una región promotora. En un promotor no regulado por el TF, esperamos una distribución determinada (distribución *background*) de las puntuaciones (B). En un promotor regulado por el TF, sin embargo, esperamos que la distribución de las puntuaciones del PSSM (R) sea una mezcla de tanto la distribución *background* (B) como de la distribución de puntuaciones en sitios funcionales (D). Para cualquier promotor dado, podemos definir la probabilidad posterior de la regulación cómo $P(R \mid D)$.

Orthologs groups of genes (8)

La detección de ortólogos sigue siendo un problema computacionalmente intensivo en bioinformática. Aquí presentamos la mayor dependencia de CGB, BLAST [6]. BLAST es un programa capaz de comparar una secuencia problema (denominada *query*) contra una gran cantidad de secuencias que se encuentren en una base de datos. Los aciertos BLAST por pares recíprocos se utilizan para generar un grafo donde los nodos corresponden a productos génicos y las aristas denotan las mejores relaciones recíprocas de aciertos BLAST, y los grupos de genes ortólogos se detectan como cliques en el grafo.

CGB limita la detección de ortólogos a aquellos genes presentes en operones con una probabilidad posterior de regulación superior a un límite especificado por el usuario en cualquiera de las especies objetivo. Esto reduce el tiempo de ejecución considerablemente.

Ancestral state reconstruction (9)

Una regla empírica común en muchos análisis genómicos comparativos es asumir que la detección de TFBS aparentes en la región del promotor en operones ortólogos de dos o más genomas suficientemente divergentes representa una fuerte evidencia de regulación. Más recientemente, la reconstrucción comparativa de regulones se ha reformulado formalmente como un problema de reconstrucción de estados ancestrales, en el que se busca inferir la probabilidad de regulación para un operón dado en un árbol filogenético.

CGB implementa este enfoque mediante el *bootstrapping* de la reconstrucción de estados ancestrales para cualquier gen dado en un árbol filogenético de instancias de TF.

Report and visualization (10)

En la última sección del flujo de trabajo se generan e imprimen las gráficas, tablas, ilustraciones, etc. con la información obtenida tras la ejecución. El usuario puede configurar previamente que tipo de resultados quiere visualizar.

3 OBJETIVOS

El objetivo de este TFG es mejorar el pipeline input/output de la plataforma CGB. Para lograrlo se han determinado dos subobjetivos:

3.1 Implementación de la obtención automática de genomas para el input de CGB.

Actualmente, CGB toma como entrada un archivo JSON. Además de los parámetros de análisis, el archivo JSON especifica dos tipos principales de información:

-TF conocidos y sus binding sites

-Genomas diana que se analizarán, enumerados con ID de RefSeq / GenBank para todos los crómidos constituyentes.

En un caso de uso típico, el usuario querrá analizar la red reguladora codificada por una transcripción de interés dentro de un clado bacteriano específico (por ejemplo, orden *Enterobacteriales*). Actualmente, esto requiere que el usuario identifique manualmente todas las especies bacterianas que se analizarán y compile una lista de sus ID de crómido (un crómido es un cromosoma, plásmido o registro contiguo). Para genomas completos, esto no es demasiado costoso (generalmente uno o dos cromosomas más algunos plásmidos), pero los "draft genomes" (ensamblajes Whole Genome Sequencing [7]) pueden tener cientos de cóntigos.

El objetivo es permitir al usuario, alternativamente, ingresar algunos parámetros para el análisis y hacer que CGB preprocese el input para obtener toda la información relevante sobre el ID del crómido. Para mantener la compatibilidad, el enfoque más simple parece ser leer el archivo JSON con parámetros alternativos (y sin ID de crómido) y volver a guardarlo una vez que se hayan compilado los ID de crómido. Es decir, pasamos de un archivo JSON con un campo "genomas" vacío (y algunos parámetros de configuración agregados) a un archivo JSON que contiene los "genomas".

3.2 Implementación de algoritmos de alineamientos para los motivos de referencia.

Actualmente CGB requiere que los motivos de referencia se proporcionen como listas de sitios de unión alineados. Si bien esto es conveniente desde el punto de vista del procesamiento, la plataforma se beneficiaría si pudiera permitir la realineación en caso de que los motivos entregados en el input no lo estuvieran, de modo que se puedan ingresar motivos sin estar necesariamente alineados.

4 METODOLOGÍA

La metodología utilizada para este TFG es AGILE (véase Figura 3). A lo largo de las semanas se han realizado reuniones en las que se discutía el diseño, los requisitos, las dependencias y en definitiva el plan de acción a seguir para la realización de cada subobjetivo. Se escogió la metodología AGILE debido a la flexibilidad que aporta. Cada vez que se completaba una tarea se discutía sobre cómo empezar la siguiente. Otra razón por la cual decidimos usar esta metodología es porque no teníamos planteados todos los objetivos del proyecto ni como llevarlos a cabo, en un principio. AGILE permite enfocarse en traba-

jar sobre un único objetivo aislándolo de los demás.

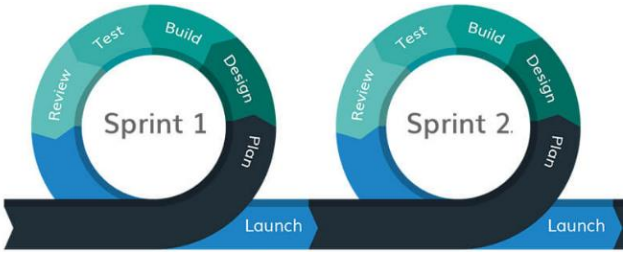

Fig 3. Metodología AGILE

Un ciclo AGILE estará compuesto por los siguientes pasos:

1. Definir un pseudocódigo no muy elaborado, lo suficiente como para que tener un hilo que seguir a lo hora de programar. Aquí se definen el diseño, los requisitos, las dependencias, etc.
2. Programar hasta que el software funcione, sin tener en cuenta memoria utilizada, tiempo de ejecución, redundancia en la codebase, etc.
3. Testear software para ver que realmente el código funciona.
4. Transformar el código para que tenga en cuenta los parámetros de eficiencia, es decir, intentar reducir el tiempo de ejecución, la memoria utilizada, la cantidad de código escrito, etc.
5. Volver a testear el nuevo código.
6. Introducir comentarios, cambiar el nombre a variables y funciones y cambiar el estilo para que pueda ser entendido y mantenido por otros programadores de software.

El lenguaje utilizado fue Python (3.8) en un entorno de programación creado mediante *Spyder*. La librería que nos permitirá conectarnos y trabajar con las bases de datos NCBI es biopython.

5 Planificación

El TFG se divide en 9 grandes tareas:

-Inicio del proyecto.

-Realización del informe inicial

-Realización informe de progreso 1

-Realización informe de progreso 2

-Realización informe final

-Realización del primer subobjetivo.

-Realización del segundo subobjetivo.

-Acoplamiento de código a GitHub

-Realización de la presentación

Para información más detallada sobre la planificación y las tareas véase el diagrama de Gantt del proyecto en el anexo A13.

6 Desarrollo

En este apartado se detallará la implementación de los dos subobjetivos.

6.1 Implementación de la obtención automática de genomas para el input de CGB

Antes de nada, comentaremos las 3 librerías que se han utilizado:

- **biopython** [8], en específico el módulo Entrez (E-utilities) y el módulo SeqIO.

Las E-utilities son un conjunto de ocho programas "server-side" que proporcionan una interfaz estable para comunicarse, mediante "querys", con las bases de datos del Centro Nacional de Información Biotecnológica (NCBI). E-utilities utilizan una sintaxis de URL fija que traduce un conjunto estándar de parámetros de entrada en los valores necesarios para que varios componentes software de NCBI busquen y recuperen los datos solicitados. Las E-utilities son la interfaz estructurada del sistema Entrez, que actualmente incluye 38 bases de datos que cubren una variedad de datos biomédicos, incluidas secuencias de nucleótidos y proteínas, registros de genes, estructuras moleculares tridimensionales y la literatura biomédica.

SeqIO proporciona una interfaz simple y uniforme para ingresar y retornar secuencias con formatos de archivo variados (FASTA, genbank, etc.), transformándolas en objetos SeqRecord con métodos y atributos útiles para tratarlos homogéneamente.

- **ete3** [9], librería que nos permite navegar eficientemente por el árbol taxonómico de Taxonomy.

La base de datos Taxonomy de NCBI contiene información taxonómica de una gran cantidad de especies, pero está implementada de forma que no ocupe demasiada memoria lo que posibilita la opción de descargarse la base de datos en local y acceder a los registros de esta mucho más rápido que mediante el uso de los servidores NCBI. Esto es lo que hace precisamente la librería ete3, montar una copia de la base de datos Taxonomy en local y brindar métodos para poder recorrerla y extraer información determinada.

- **json**, librería nativa de Python para procesar archivos JSON.

Podemos dividir el funcionamiento del código en cuatro partes.

1. Lectura del archivo JSON (input CGB) y posterior

escritura de los accession numbers de los genomas (o assemblies) de interés en el archivo JSON y en el diccionario de variables de CGB.

Leemos del archivo JSON, que utilizará CGB como input, el TaxID y el nivel taxonómico de búsqueda. Además, leemos los parámetros de configuración opcionales, los 3 boléanos relacionados con las entidades taxonómicas noRank que explicaremos más adelante. Una vez obtenemos los accession numbers (sección 2 y 3 a continuación), los escribimos dentro de un diccionario determinado del archivo JSON y en el diccionario de variables de CGB.

2. Obtención del árbol taxonómico a partir de un TaxID

Mediante ete3 creamos una copia de la base de datos Taxonomy. El uso de dos parámetros (input usuario) que son el TaxID de la entidad taxonómica de la cual se quieren extraer los genomas (ej: 562 el TaxID de *Escherichia*) y un nivel taxonómico determinado (ej: especie) permite acotar detalladamente la información que queremos extraer de la base de datos local.

Pongamos un ejemplo, si un usuario quiere estudiar los genomas de todas las especies pertenecientes al género *Escherichia* entonces el primer paso sería saber qué y cuántas especies son. Esto es lo que realiza esta primera parte del código, obtiene un conjunto de TaxID de especies determinadas que posteriormente serán usados para obtener sus genomas representativos.

En un primer momento se intentó no utilizar la librería ete3, lo que implicaría utilizar E-utilities para acceder y filtrar la información de Taxonomy. Esta opción se implementó y desechó debido a que aumentaba notablemente el tiempo de ejecución del código si el usuario quisiera estudiar múltiples entidades taxonómicas diferenciadas.

Hay que destacar un problema que deriva de la clasificación taxonómica que utiliza Taxonomy. Si la clasificación de entidades taxonómica (nodos del árbol filogénico) fuera perfecta los niveles del árbol serían únicamente los siguientes:

Nivel taxonómico	**Altura en el árbol taxonómico**
Raíz (root)	0
Superreino (Superkingdom)	1
Clado (clade)	2
Filo(phylum)	3
Clase (class)	4
Orden (order)	5
Familia (family)	6
Género (genus)	7
Especie (species)	8
Cepa (strain)	9

Sin embargo, como no todas las entidades taxonómicas descubiertas están clasificadas o demostradas, Taxonomy utiliza un añadido nivel taxonómico para intentar posicionar a dichos sujetos en el árbol de una forma relativamente organizada. Este nivel es el "noRank", que engloba las entidades taxonómicas no clasificadas (*unclassified*) y las muestras ambientales (*environmental samples*). El código tiene en cuenta esto, utiliza 3 boléanos provenientes del JSON para configurar como va a reaccionar el algoritmo ante este tipo de entidades taxonómicas. Por defecto, se asume que el usuario quiere procesar entidades taxonómicas noRank ya sean *unclassified* o *envirormental samples*.

También se ha implementado un método para detectar y avisar cuando el nivel taxonómico de búsqueda es menor (altura del árbol) al nivel taxonómico del TaxID, ya que si no se darían situaciones absurdas como buscar filos dentro de géneros y obviamente el programa no retornaría nada.

3. Acceso a las secuencias del genoma a partir de un TaxID (proveniente del árbol taxonómico)

Una vez tenemos todas las TaxID necesarias hay que obtener los accession number (identificadores únicos) de los genomas para poder escribirlos en el archivo input JSON de CGB. En teoría la operación debería ser directa, utilizar E-utilities para hacer una búsqueda del genoma representativo según un TaxID determinado. Sin embargo, aparecen varios problemas.

Antes de nada procedemos a explicar que hacen esearch(), efetch() y elink(), las funciones de la api E-utilities [10].

- esearch(), busca información en las bases de datos de NCBI utilizando como parámetros básicos el nombre de la base de datos y el término (query text) de búsqueda. Retorna una lista de identificadores únicos (UID list).

Ejemplo: Entrez.esearch(db=nuccore, term="txid956"). Este código realiza una búsqueda en la base de datos de nucleótidos (nuccore) y retorna una UID list de los archivos de nucleótidos de la entidad taxonómica "956" (TaxID).

- efetch(), busca registros específicos en las bases de datos de NCBI utilizando como parámetros básicos el nombre de la base de datos y el UID del registro que se quiere obtener. Retorna objetos de estructuras distintas según la base de datos a la cual quieras acceder.

Ejemplo: Entrez.efetch(db=genome, id=167). Este código realiza una búsqueda en la base de datos de genomas (genome) y retorna un objeto. Para más información sobre qué objeto retorna dependiendo de la base de datos accedida, consultar [11].

Sin obligar al lector a acceder a la referencia anterior voy a anunciar que conclusiones sacamos de la información publicada en dicho enlace. Las E-utilities se diseñaron orientadas al uso de una escasa cantidad de bases de datos: nuccore, nucest, nucgss, protein, popset y snp. Por lo tanto, se han diseñado objetos específicos para tratar con registros de dichas bases de datos, pero el resto de las bases de datos únicamente tienen a su disposición los objetos por defecto. Los objetos por defecto (docsum y uilist) representan vagamente los registros, además, no contienen atributos y métodos específicos que podrían brindar información útil. Las bases de datos a las que accedemos a lo largo del programa son taxonomy [12], assembly [13], genome [14] y nuccore [15], la única con objetos específicos. El hecho de usar los objetos por defecto limita la cantidad de información que podemos extraer de ellos por lo tanto complica el proceso que queremos implementar.

- elink(), realiza acciones varias pero la que utilizamos es la de retornar UIDs vinculados a un conjunto de UIDs (argumento) de la misma base de datos o de una diferente. Los argumentos obligatorios que usa son los nombres de las dos bases de datos y la lista de UIDs.

Ejemplo: Entrez.elink(db=nuccore, dbfrom=assembly, id=1755381). Este código realiza una búsqueda en la base de datos de nucleótidos y retorna una lista de UID de las secuencias de nucleótidos que estén vinculadas al assembly con UID 1755381.

Una vez que tenemos una idea de cómo podemos interactuar con las bases de datos NCBI podemos deducir que surge un problema y es que no hay una operación directa para obtener secuencias de genomas representativos mediante un TaxID. El proceso más eficiente encontrado es el siguiente: primero utilizamos esearch() en la base de datos de genomas con un TaxID determinado siendo este el término de la query, esto nos retorna un UID de genoma. Segundo, con este UID realizamos un efecth() en la base de datos de genoma, esto nos retorna un objeto docsum. Por desgracia este objeto, al ser por defecto, no contiene el archivo secuencia por lo tanto debemos seguir operando en las bases de datos NCBI, sin embargo, sí que nos retorna el UID de assembly del genoma. Tercero, con este UID de assembly realizamos un elink() en la base de datos de nucleótidos buscando registros de nucleótidos vinculados al assembly UID del genoma específico. Esto nos retorna una lista de UIDs de registros de nucleótidos. Uno pensaría que el proceso ya está terminado ya que tenemos el UID del registro de nucleótidos del genoma representativo del TaxID determinado, pero CGB no admite UIDs de registros de nucleótidos. CGB utiliza lo que se llama "*accession number*" que es el identificador único para un registro de secuencias. Para la sección que estamos implementando el accession number tiene funcionalidad idéntica al UID del registro de nucleótidos, pero para CGB no es así, por lo tanto, se requiere realizar un último paso. Cuarto, con los UIDs de los registros de nucleótidos realizamos un efetch() en la base de datos de nucleótidos (nuccore). Esto nos retorna un objeto específico con mucha información como por ejemplo la secuencia completa de bases nitrogenades, que guardaremos en caché, y el accession number de dicha secuencia que es lo que nos interesa.

Ahora comentaremos un problema que nos encontramos programando está parte y es que no todas las especies registradas en Taxonomy tienen un genoma representativo comprobado (RefSeq genome), incluso puede que ni siquiera tengan un genoma secuenciado, y lo que es peor, puede que ni siquiera tengan un registro assembly. Debemos tomar medidas si alguno de los casos mencionados aparece. El caso ideal es obtener un RefSeq representative genome, sino se da el caso se aceptaría cualquier otro tipo de genoma (Genbank u otras bases de datos). Si una especie no tiene un genoma registrado entonces pasamos a buscar assemblies, hemos dividido la idealidad de los registros assembly en 3 tiers:

Tier 1	Assembly RefSeq secuenciación completa
Tier 2	Assembly secuenciación completa
Tier 3	Assembly secuenciación parcial

Finalmente, si una especie no tiene registros assembly se ignora. En un principio se tenía planeado implementar un filtro de calidad de secuencias de nucleótidos para ser retornadas si ocurría este caso, pero desechamos la idea.

4. Main, ensamblaje de las 3 partes anteriores un sólo algoritmo.

El main es un sencillo código que orquesta el funcionamiento de las 3 partes anteriores del código teniendo en cuenta las posibilidades de búsqueda del usuario. El programa funciona de dos maneras. Si el usuario introduce "especie" como nivel taxonómico de búsqueda, entonces funciona como he explicado anteriormente. Sin embargo, si introduce cualquier otro nivel taxonómico (género, familia, etc.) entonces se modifica ligeramente la segunda parte del código ya que se realiza un filtro de las especies para seleccionar una única especie representativa para cada entidad taxonómica dentro de dicho nivel taxonómico y luego trabaja como lo haría normalmente.

Pongamos un ejemplo, si un usuario quiere estudiar los géneros dentro de un orden determinado entonces primero se obtienen todos los géneros pertenecientes a dicho orden y luego se realiza un filtro para seleccionar una especie representativa de dicho género. Sobre esta especie representativa trabajará el programa para obtener el ac-

cession number del genoma que representa al género.

6.2 Implementación de algoritmos de alineamiento para los motivos de referencia

La alineación se debe realizar en 2 casos:

-Alineación de sites dentro del motivo, es decir alineamiento de secuencias dentro del conjunto de secuencias pertenecientes a la misma especie (motivo). Se utiliza el algoritmo LASAGNA.

-Alineación de motivos de especies diferentes, es decir alinear un motivo con otro y por tanto alinear las secuencias que contienen ambos. Se utiliza el algoritmo IC (contenido de información) [16].

El primer algoritmo de alineamiento, LASAGNA (véase Figura 4 y 5), es relativamente complejo por lo que vamos a explicarlo con cierto nivel de abstracción.

El algoritmo toma un conjunto de binding sites no alineados, *U*, como entradas. Sea *A* el conjunto de binding sites alineados. Un binding site en *A* puede tener letras "hueco" (gap letters) agregadas en uno o ambos extremos como resultado de la alineación. El algoritmo funciona de la siguiente manera:

1. Inicializa *A* en {*s*}, donde *s* es el binding site más corto elegido arbitrariamente de *U*. Luego quitamos *s* de *U*.

2. (a) Construye la PSSM a partir de *A*. Sea *L* la longitud de esta PSSM.

   (b) Elimina el binding site más corto de *U*.

   (c) Crea *S*, la secuencia aumentada de *s*, agregando *L*−1 gap letters a ambos extremos de *s*.

   (d) Califica cada elemento de *S* mediante PSSM para encontrar el puntaje más alto.

   (e) Con el complemento inverso de *s* se repite c y d. Es decir, se considera la hebra opuesta.

3. Agrega *s* a *A* si el elemento de puntuación más alta reside en *s*. De lo contrario, agregue su complemento inverso a *A*. Las gap letters se agregan a uno o ambos extremos de las secuencias en *A*. Esto asegura que todas tengan la misma longitud y que cada columna de la alineación tenga al menos una letra que no sea gap.
4. Repite 2-3 hasta que *U* esté vacío.

En el paso 2b, puede haber varios binding sites igual de cortos en *U*. Para romper el empate, usamos PSSM para escanear cada uno de los binding sites más cortos. El binding site que contiene el elemento con la puntuación más alta se elimina de *U* para alinearse con las secuencias en *A*. En el caso poco probable de que dos o más binding sites con longitud mínima en *U* compartan la misma puntuación, se elige uno arbitrariamente.

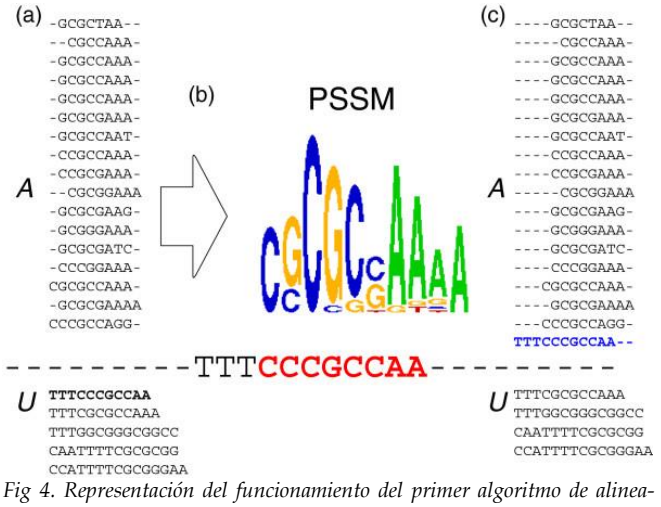

Fig 4. Representación del funcionamiento del primer algoritmo de alineamiento (LASAGNA [17]).

A la hora de implementar LASAGNA teníamos dos opciones, implementar el algoritmo desde cero o bien adaptar el código existente proveniente de esta web [18].

Adaptar el código existente para que pueda ser utilizado en CGB tiene 2 inconvenientes. Primero el código está escrito en Python 2 por lo tanto se debe convertir a Python 3. Segundo, sólo utiliza la librería numpy por lo tanto crea nuevas clases y métodos que ya están implementados por biopython aumentando la cantidad de código enormemente.

Debido a esto decidimos implementar una versión del algoritmo propia y adaptada a CGB. La implementación tenía que utilizar biopython y así aprovechar todas las clases y métodos que tiene para reducir al máximo la cantidad y complejidad del código. Las clases de biopython necesarias para la implementación son "Seq" y "Alphabet". La primera sirve para representar secuencias y la segunda para representar el alfabeto de caracteres posibles que utilizan la secuencias.

El primer obstáculo que debíamos superar era encontrar un método para introducir gaps (huecos) en las secuencias ya que LASAGNA añade gaps en las secuencias. Por defecto el constructor de un objeto "Seq" sólo admite 4 caracteres, las bases nitrogenadas del ADN (A, C, G y T).

No hay manera de introducir un hueco o "gap" en la secuencia. Aquí es donde entra la clase "Alphabet". Con dicha clase se puede introducir un nuevo carácter utilizable en las secuencias, el carácter "gap" (-). Tras leer el artículo [17] y entender el funcionamiento del algoritmo y las herramientas disponibles iniciamos la codificación.

La implementación fue satisfactoria, pero surgió un problema que no habíamos previsto. La clase "Alphabet" sería eliminada en las próximas actualizaciones de biopython y se le otorgaría el estatus de "deprecated". Tras mucho debatir decidimos que no merecía la pena realizar una nueva implementación sin utilizar la clase "Alphabet" debido a la complejidad y fragilidad de código. Una semana de trabajo fue desechada debido a que en un principio no leímos con detenimiento cual era la vida útil de una clase, lección aprendida.

Tras este suceso decidimos decantarnos por la otra opción, adaptar y convertir el código existente del algoritmo LASAGNA. Este proceso es relativamente trivial. Debíamos entender, filtrar, adaptar y convertir, aproximadamente 600 líneas de código (código fuente de [18]), además de añadir las que fuesen necesarias para que el algoritmo funcionase con el input/output de CGB.

Caso de uso

Fig 5. Caso de uso algoritmo 1

El segundo algoritmo de alineamiento (véase Figura 6) se basa en utilizar el contenido de información que otorga cada posible alineamiento entre las dos agrupaciones de secuencias y quedarse con el alineamiento que maximice el IC.

Para lograr esto el proceso que hemos seguido es el de crear motivos temporales donde cada uno contiene las secuencias superpuestas en una alineación determinada por un índice y se realiza el promedio de la PSSM para esa alineación determinada. Mediante el promedio de la PSSM se obtiene el IC.

La librería biopython contiene métodos para trabajar con objetos de la clase "Seq" (objetos que representan secuencias). Dentro de estos métodos está pssm() y mean() por lo tanto el proceso es bastante trivial. Tenemos que crear tantos motivos (objeto "Seq") como combinaciones de alineamientos posibles (determinados por índices) entre las secuencias y aplicar a cada uno de esos objetos los métodos pssm() y mean(), es decir motif.pssm().mean(), para obtener el IC. El motivo que maximice IC determinará el alineamiento óptimo.

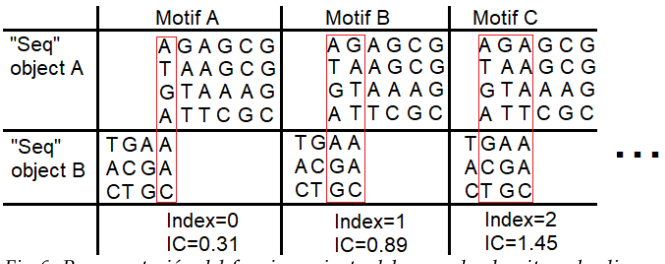

Fig 6. Representación del funcionamiento del segundo algoritmo de alineamiento.

Caso de uso

Partimos de 2 motivos de referencia de 2 especies diferentes (véase Figura 7):

Especie 1:	Especie 2:
CTTAATATTCTCTTAAAGTT	TAATATTTCGTTAAGGT
GTTAATATTTGCTTAAGTAT	TAAGGTTGCATTAATTT
CTTAATATAAGATTAATTTT	TAATATTTCCTTAAGGT
TTTAAGTTTTACTTAAGTTA	TAATATTCTCTTAAGCT
CTTAAGGTTAAATTAATATT	TAATATTTTCTTAAGGT
CTTATGGTTTTCTTAAGCGT	TAATACTGTCTTAAGGA
CTTAATACTTTCTTAATACT	TAATATTTCGTTAAGGT
CTTAATATTTCCTTAAAACT	

Fig 7. Input algoritmo 2

Como observamos las secuencias están alineadas, pero únicamente dentro de la propia especie. CGB requiere que las secuencias de motivos de todas las especies tengan la misma longitud. Por lo tanto, se debe realizar alineamiento y un recorte posterior de las secuencias siguiendo un determinado criterio (véase Figura 8). Este criterio como hemos comentado es la cantidad de información.

Fig 8. Alineamiento segundo algoritmo.

7 PRESENTACIÓN Y DISCUSIÓN DE RESULTADOS

A continuación, mostraremos unas graficas de rendimiento de la ejecución del código de ambos subobjetivos. Para realizar las pruebas de rendimiento se ha utilizado una máquina con un procesador Intel Core i7-4790 de 3.60GHz, 16 GB de RAM DDR3, 550 Mb/s de velocidad bajada y memoria HDD.

7.1 Ejecución primer subobjetivo

Se ha de destacar que en el caso de las ejecuciones del código del primer subobjetivo es difícil establecer una métrica indicando si es eficiente o no, ya que no hay unas instrucciones claras a la hora de volcar la información de secuencias en un archivo genbank. En concreto me refiero a que un genoma puede estar formado por 40 archivos genbank en donde cada archivo contiene 4 kB de secuencias y otro genoma de otra especie diferente puede estar representado en un único fichero genbank con un peso de 11 MB. Además, los archivos genbank contienen otro tipo de información que no son secuencias que varía en tamaño según el creador. En definitiva, podemos decir que la eficiencia de almacenaje de secuencias en archivos genbank es aleatoria y por tanto la ejecución del código del primer subobjetivo hasta cierto punto también lo es.

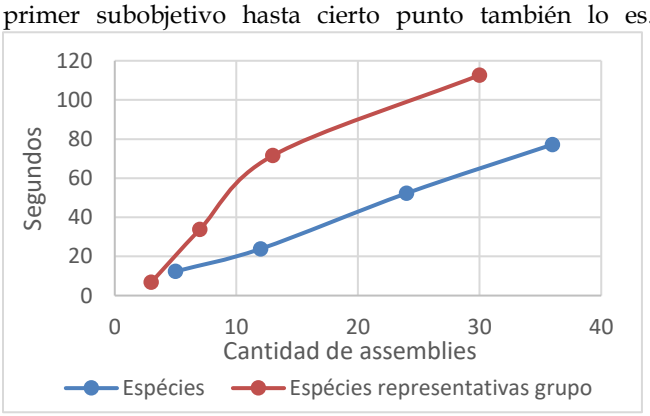

Fig 9. Gráfica rendimiento búsquedas assembly

Si observamos la figura 9 podemos ver 8 ejecuciones distintas del código del primer subobjetivo. La línea roja representa 4 ejecuciones de búsquedas de assemblies de especies representativas de 4 grupos taxonómicos distintos. La línea azul representa 4 ejecuciones de búsquedas de assemblies de especies dentro de un grupo taxonómico. Como es de esperar el algoritmo tarda más en encontrar assemblies de especies representativas de grupos taxonómicos que assemblies de cualquier especie debido a que no tiene que hacer el paso previo de elegir la especie representativa.

7.2 Ejecución segundo subobjetivo

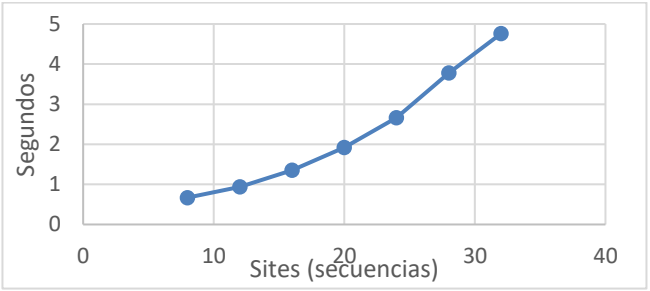

Fig 10. Gráfica rendimiento algoritmo de alineamiento

En el caso del código del segundo subobjetivo se han realizado 7 ejecuciones con 8, 12, 16, 20, 24, 28 y 32 sites de longitudes de 8 a 12 caracteres provenientes de 2, 3, 4, 5, 6, 7 y 8 motivos respectivamente.

Al combinar los dos algoritmos se busca primero alinear todos los sites de un motivo y luego alinear dicho motivo junto con otro motivo que ha pasado por el mismo proceso. Esto se va realizando iterativamente hasta que todos los motivos estén alienados (véase figura 11). Al realizar la siguiente iteración se utiliza el motivo resultante de la anterior iteración y otro nuevo.

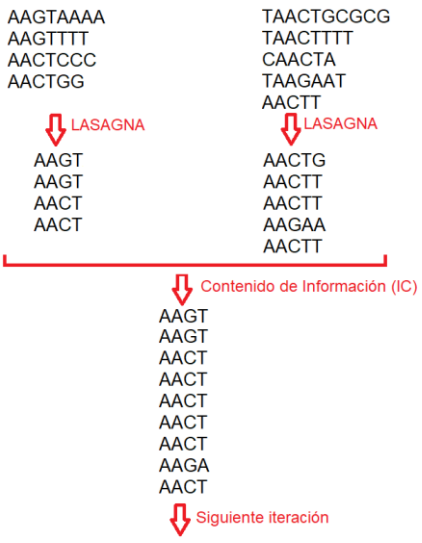

Fig 11. Combinación de ambos algoritmos de alineamiento

Al utilizar dos algoritmos deterministas, al contrario que en caso del primero subobjetivo, los resultados obtenidos en las pruebas de rendimiento del segundo subobjetivo son fiables.

7.3 Acoplamiento del código

El desarrollo de las funcionalidades desde un principio se realizó totalmente aislado de CGB. Por ello se introdujeron cambios debido al proceso de acoplamiento del código a la plataforma CGB y al GitHub en sí:

-Substitución de variables y métodos por los ya presentes en CGB.

-Modificación de métodos, atributos y clases de la codebase de CGB.

-Eliminación de métodos y variables destinadas al *debugging*.

-Homogeneización del estilo de programación para que estuviera en relativa sincronía con el de CGB. Además de la adición de comentarios competentes.

8 Conclusiones

La obtención automatizada de genomas para el input de CGB y la creación de algoritmos de alineamiento competentes han sido todo un éxito. Se ha ido alterando el código de ambos a lo largo del TFG, pero el funcionamiento que se tenía como objetivo se ha logrado. Acoplarlo a CGB dio más problemas de lo esperado debido a que se debían cambiar como se obtenían los parámetros de entrada y cómo se mostraban los resultados de la ejecución. También tuvimos que alterar algunas funciones de CGB para que trabajasen correctamente con el nuevo código.

Únicamente hubo dos hitos que no se lograron realizar y no fue por falta de tiempo o habilidad sino más bien por la viabilidad de su realización. En ambos casos el esfuerzo necesario para llevarlos a cabo no merece la pena.

El primero es la creación de un filtro de secuencias de nucleótidos cuando no hay genomas representativos o assemblies que retornar para un TaxID determinado. Cómo he comentado anteriormente en la explicación del desarrollo del primer subobjetivo, este filtro no era viable de realizar ya que expandiría mucho la codebase, además, se utilizaría en casos muy puntuales y en estos casos aumentaría enormemente el tiempo de ejecución.

El segundo es la implementación del algoritmo LASAGNA mediante biopython. Este algoritmo se llegó a implementar, pero debido a que utilizaba una clase que en un futuro cercano obtendría el estado de obsoleta se decidió desechar.

Finalmente, hablaré de las posibles vías de continuación para este TFG. CGB es un software complejo y en que han participado un considerable número de personas. Esto es algo que se puede ver reflejado en el código. Cada persona tiene una forma de programar distinta y una idea distinta del funcionamiento objetivo de CGB. En algunos casos el código parece estar optimizado para reducir el impacto en memoria o en el procesador, en otros casos parece tener como objetivo prioritario la comprensión y mantenimiento del código, etc. Además, cómo consecuencia directa de la heterogeneidad del código, no hay un estilo de programación definido y la ausencia de este es relativamente evidente a la hora de mirar los comentarios y nombres de variables. Por lo que sí, pienso que sería positivo intentar homogenizar el código.

Sin embargo, el siguiente paso a realizar es la transformación de CGB de un programa ejecutado en local a un software ejecutado en un servidor web. Instalar CGB, aunque sencillo, puede ser engorroso para los usuarios por lo que sería beneficioso ejecutarlo en plataformas cómo PythonAnywhere [19] que permiten al usuario utilizar el programa en cualquier lugar sin tener que realizar un montaje previo o sacrificar memoria de sus máquinas.

Agradecimientos

Agradezco la paciencia, ayuda y atención recibida por parte de Iván Erill y Joan Serra Sagristà a lo largo del desarrollo del TFG y la memoria. También doy gracias a la Escuela de Ingeniería de la UAB por brindarme la oportunidad de embarcarme en este proyecto y de otorgarme el conocimiento y habilitades necesarias para llevarlo a cabo.

Bibliografía

[1] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7739468/ Flexible comparative genomics of prokaryotic transcriptional regulatory networks. Sefa Kılıç, Miquel Sánchez-Osuna, Antonio Collado-Padilla, Jordi Barbé y Ivan Erill 2020
[2] https://www.ncbi.nlm.nih.gov/
[3] https://pubmed.ncbi.nlm.nih.gov/20080407/ Introducing the bacterial 'chromid': not a chromosome, not a plasmid. Peter W Harrison, Ryan P J Lower, Nayoung K D Kim, J Peter W Young 2010
[4] https://www.genome.gov/es/genetics-glossary/C%C3%B3ntigo
[5] https://en.wikipedia.org/wiki/Position_weight_matrix
[6] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[7] https://en.wikipedia.org/wiki/Whole_genome_sequencing
[8] http://biopython.org/DIST/docs/tutorial/Tutorial.html
[9] http://etetoolkit.org/docs/latest/tutorial/tutorial_ncbitaxonomy.html
[10] https://www.ncbi.nlm.nih.gov/books/NBK25499/
[11] https://www.ncbi.nlm.nih.gov/books/NBK25499/table/chapter4.T._valid_values_of__retmode_and/?report=objectonly
[12] https://www.ncbi.nlm.nih.gov/taxonomy/
[13] https://www.ncbi.nlm.nih.gov/assembly/
[14] https://www.ncbi.nlm.nih.gov/genome/
[15] https://www.ncbi.nlm.nih.gov/nuccore/
[16] https://en.wikipedia.org/wiki/Information_content
[17] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3747862/ LASAGNA: A novel algorithm for transcription factor binding site alignment. Chih Lee and Chun-Hsi Huang 2013
[18] https://biogrid-lasagna.engr.uconn.edu/lasagna_search/index.php
[19] https://www.pythonanywhere.com/

APÉNDICE

A1. Transcripción

Es el proceso mediante el cual una célula elabora una copia de ARN proveniente de una pieza de ADN. Esta copia de ARN, que se llama ARN mensajero (ARNm), transporta la información genética que se necesita para elaborar las proteínas en una célula

A2. Traducción

Es el segundo proceso de la síntesis proteica (siendo el primero la transcripción), parte del proceso general de la expresión génica, que ocurre en todos los seres vivos. En la traducción, el ARN mensajero se decodifica para generar una cadena específica de aminoácidos, llamada polipéptido (el producto de la traducción), de acuerdo con las reglas especificadas por el código genético. Es el proceso que convierte una secuencia de ARN mensajero en una cadena de aminoácidos para formar una proteína

A3. Regulación

Incluye una amplia gama de mecanismos que utilizan las células para aumentar o disminuir la producción de productos genéticos específicos (proteína o ARN). Vamos a poner un ejemplo. Todas las células del cuerpo contienen una copia exacta de todo el ADN del individuo, sin embargo, hay células que realizan funciones diferentes respecto a otras. Por ejemplo, las células que se encuentran en la pared del estómago producen HCl (ácido clorhídrico) que ayuda a digerir alimentos, esas mismas células contienen el mismo ADN que las células que forman la retina de tus ojos. Sin embargo, el humano no secreta HCl por los ojos y esto es debido a que existe regulación de material genético en las células para determinar qué productos genéticos se crean y son utilizados por la célula para realizar su función en el organismo. Si todas las células del cuerpo produjeran los mismos productos genéticos y no existiera la regulación de genes, todas las células del cuerpo realizarían la misma función lo que obviamente no es compatible con la vida pluricelular.

A4. Polimerasa

Es una enzima capaz de transcribir o replicar ácidos nucleicos, que resultan cruciales en la división celular (ADN polimerasa) y en la transcripción del ADN (ARN polimerasa).

A5. Gen Estructural

Un gen estructural es un tipo de gen que codifica una proteína o una cadena de ARN en particular. Se trata de genes cuya expresión está regulada.

A6. Gen Regulador

El gen regulador es un tipo de gen que se involucra en el control de la expresión (regulación) de uno o más genes.

A7. Operón

Es un complejo presente en los cromosomas de organismos procariotas. Dicho complejo está formado por genes que codifican para la síntesis productos genéticos específicos, cuya expresión generalmente está regulada por un único promotor y operadores.

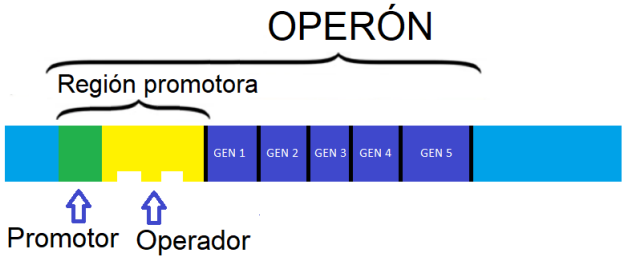

Fig 12. ***Partes de un Operón.***

A8. Promotor

Se trata de un elemento de control que es una región del ADN con una secuencia que es reconocida por la ARN polimerasa para comenzar la transcripción.

A9. Operador

Se trata de otro elemento de control que es una región del ADN con una secuencia que es reconocida por el factor de transcripción. El operador se sitúa en la región promotora.

A10. Factor Transcripcional o TF

Los factores de transcripción son proteínas que se unen a promotores de una manera específica y pueden dificultar o promover la transcripción de operones que contienen genes expresados a partir de un promotor compartido. El TF se une a la región promotora y actúa como puerta lógica impidiendo o no el paso de la polimerasa.

A11. Binding Site o TFBS

Los promotores, operadores y TFs interactúan entre ellos para realizar la transcripción selectiva de operones. En el control de la expresión de genes intervienen el promotor, una secuencia de bases nitrogenadas cerca del inicio de la secuencia del gen a transcribir. El promotor no es transcrito, pero actúa como lugar de unión (binding site) para la RNA polimerasa permitiendo el inicio de la transcripción. Sin embargo, para que esto suceda se requiere la presencia de los factores de transcripción, proteínas que se unen al promotor permitiendo o denegando la unión de la polimerasa y el promotor. A veces, varios TFs son necesarios y primero deben ser "activados" para poder unirse al promotor.

Hay que tener en cuenta que estos elementos no son los únicos que controlan la transcripción de los genes, inter-

vienen también otras proteínas reguladoras que actúan en otros niveles, además de otras moléculas que escapan al objetivo de esta explicación.

A12. Red Transcripcional

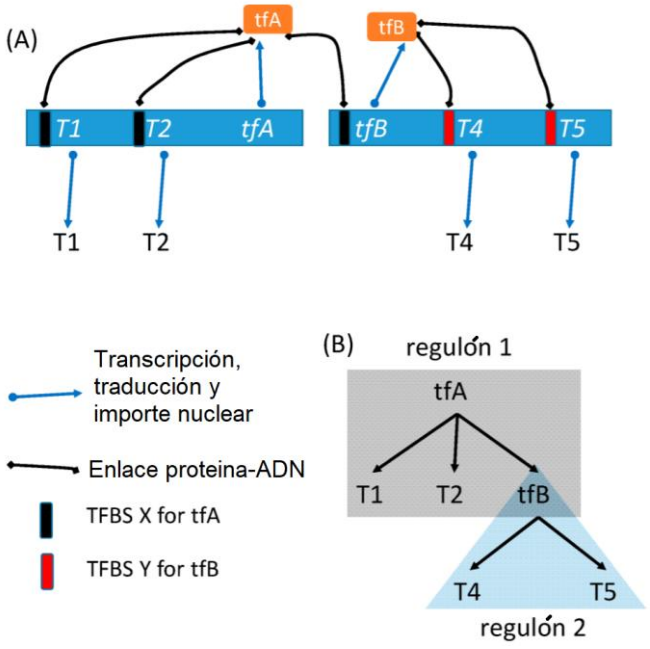

Fig 13. Red transcripcional.

T1, T2, T4 y T5 son genes pertenecientes a un operón determinado.

tfA y tfB son factores de transcripción (transcription factors).

TFBS X y TFBS Y son transcription factors binding site, es decir, promotores.

El importe nuclear es el transporte de la proteína hacia el lugar donde tiene que realizar su función.

En la figura 13 A se observan dos operones. Uno compuesto por el gen T1, T2 y el gen que produce el tfA, y otro operón, compuesto por el gen T4, T5 y el gen que produce el tfB. Además, podemos visualizar dos proteínas, el tfA y el tfB. Podemos observar como en la figura 13 B hay un árbol donde cada nodo padre (TF) y su descendencia más próxima (genes y TFs) forman un regulón. La figura 13 A y la figura 13 B están relacionadas de la siguiente forma:

El tfA controla la expresión de los genes T1, T2 y tfB, es decir que la existencia del tfB (proteína) depende de la presencia de tfA (proteína) que a la vez esta depende de la presencia de un determinado TF que promueva la transcripción del gen tfA. tfB a su vez controla la expresión de los genes T4 y T5. De esta forma se puede crear un árbol que describa las relaciones entre regulones, formando lo que se llama red transcripcional.

A13. Diagrama de Gantt

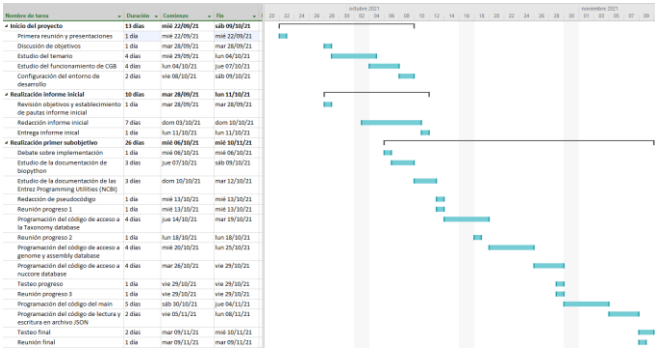

Fig 14. Diagrama de gantt. Tareas 1,2 y 3.

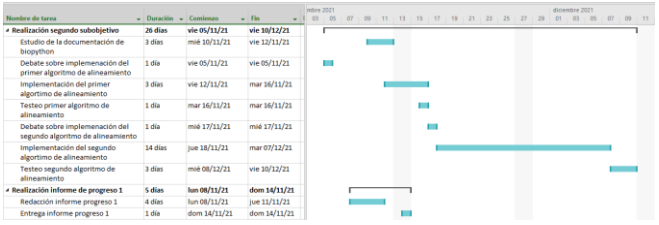

Fig 15. Diagrama de gantt. Tareas 4 y 5.

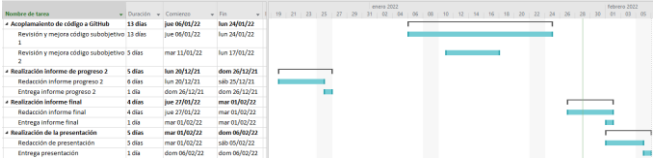

Fig 16. Diagrama de gantt. Tareas 6,7,8 y 9.