UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Búsqueda de Regiones Únicas en Genomas

Informe de Proyecto de Grado presentado por

Nicolas Blumetto

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Alvaro Martin
Guillermo Dufort y Álvarez

Montevideo, 7 de agosto de 2025

Agradecimientos

Han sido varios años de formación que no solo ampliaron mis conocimientos, sino que transformaron profundamente mi forma de ver el mundo. Este proyecto representa el cierre de una etapa intensa y formativa, y al mismo tiempo, el comienzo de una nueva etapa para la que me siento preparado. No podría dar este paso sin antes expresar mi más sincero agradecimiento a quienes me acompañaron en el camino.

Gracias a mis tutores Álvaro y a Guillermo por su conocimiento y su paciencia.

Gracias a cada profesor y mentor por enseñarme a ver más allá de lo evidente.

Gracias a mis padres y mis hermanas por su comprensión durante mis horas de estudio, su apoyo incondicional y el constante aliento. Gracias por creer en mí, incluso cuando yo mismo dudaba.

Gracias a todos mis amigos por cada risa compartida, por cada hombro ofrecido, siempre recordándome la importancia del equilibrio y celebrar cada paso.

Hoy, al mirar atrás a todos estos años, comprendo que este camino tan desafiante nunca lo recorrí solo y las palabras se quedan cortas para expresar el profundo amor y respeto que siento por cada persona que me acompañó.

Este logro es, en gran parte, de ustedes.

Resumen

La identificación de regiones únicas en genomas es una tarea clave en bioinformática, con aplicaciones relevantes en el diagnóstico molecular y la detección de patógenos. Este trabajo presenta un análisis comparativo de diversos algoritmos diseñados para esta tarea, evaluando su rendimiento en términos de eficiencia computacional y calidad biológica de los resultados. Se comparan las herramientas `fur`, `KEC` y GenMap, analizando sus ventajas y limitaciones mediante experimentos con datos simulados y reales.

Los resultados muestran que `fur` es especialmente eficaz para la búsqueda de marcadores diagnósticos de alta especificidad, mientras que `KEC` ofrece una solución rápida y eficiente, útil en escenarios donde `fur` no encuentra resultados. A partir de estos análisis, se desarrolla `KEC-FM`, una nueva herramienta que combina el enfoque basado en k-meros de `KEC` con técnicas inspiradas en `fur`, optimizando tanto la eficiencia como la calidad de los marcadores identificados.

Las evaluaciones experimentales demuestran que `KEC-FM` alcanza un rendimiento competitivo frente a las herramientas del estado del arte, posicionándose como una alternativa robusta para la identificación de regiones únicas en genomas.

Palabras clave: Regiones únicas, genomas, algoritmos, bioinformática.

Índice general

Capítulo 1

Introducción

El ADN es la molécula que alberga la información genética de los seres vivos. Su estructura consiste en largas cadenas de nucleótidos, cada uno con una de cuatro bases nitrogenadas: Adenina (A), Guanina (G), Citosina (C) y Timina (T). La secuencia específica de estas bases constituye el código genético que dirige el desarrollo y la función de todo organismo.

Un punto de inflexión para el estudio de este código fue el desarrollo de los primeros métodos de secuenciación en 1977 [19], los que permiten leer el orden exacto de los nucleótidos en el ADN. Desde entonces, la mejora continua de la secuenciación ha generado un volumen de datos genómicos sin precedentes, impulsando nuevas disciplinas basadas en el estudio y comparación de estos datos.

Desde un punto de vista computacional, los datos genómicos suelen ser archivos de texto que contienen largas cadenas de caracteres construidas a partir de un alfabeto de cuatro letras: A,G,C,T, que representan las mencionadas bases nitrogenadas del ADN. En la actualidad existen múltiples bases de datos con millones de secuencias para prácticamente todos los organismos conocidos. Esta disposición masiva de datos abre un sinfín de posibilidades para su análisis, a la par que genera la necesidad de desarrollar herramientas computacionales eficientes que permitan su procesamiento e interpretación.

Una de las aplicaciones computacionales más exitosas en la biología molecular es la búsqueda de similitud entre secuencias de ADN. Herramientas como `BLAST` [1] permiten a los investigadores tomar una secuencia y encontrar coincidencias en extensas bases de datos, siendo esto fundamental para, por ejemplo, inferir relaciones evolutivas.

Sin embargo, existe un problema conceptualmente adyacente (pues se basa en buscar diferencias), que posee una notable relevancia práctica: la búsqueda de regiones genómicas únicas.

Para un conjunto de genomas objetivo, una región única se define como un segmento de ADN que cumple dos condiciones: es **conservado** entre los miembros del grupo y está ausente en genomas de organismos externos a él.

Quizás el valor más evidente de una región única es su uso como *marcador de diagnóstico*, que son la base para el desarrollo de pruebas PCR. Estas pruebas son el estándar para la detección de organismos en muestras, y un ejemplo memorable de su aplicación fue durante la pandemia de SARS-CoV-2, donde se realizaban estas pruebas para identificar portadores del virus a partir de hisopados.

A primera vista, encontrar una región única podría parecer trivial, pero la aparente simplicidad de esta definición oculta una complejidad fundamental inherente a la biología evolutiva. El concepto de conservación no puede implicar una identidad exacta, puesto que, incluso entre individuos de una misma especie pueden existir variaciones. Si comparamos 2 genomas de dos individuos pertenecientes a la misma especie o cepa, encontraremos mutaciones que se dan naturalmente (estas mutaciones son la base de la evolución y el proceso de selección natural). Más aún, los métodos de secuenciación no son perfectos y pueden introducir mutaciones artificiales en la secuencia leída.

Esto da pie a introducir el concepto de **homología**: desde un punto de vista biológico, dos secuencias de ADN se consideran homólogas si comparten un origen evolutivo común. Computacionalmente, podemos inferir la homología basándonos en una coincidencia aproximada, con un umbral para la cantidad de mutaciones o errores que se consideran naturales o tolerables. Dicho umbral no puede ser arbitrario; idealmente, debe obtenerse a través de un análisis estadístico que distinga la similitud biológicamente significativa del ruido aleatorio.

Podemos entonces refinar la definición de región única que buscamos mediante el siguiente criterio dual:

1. Debe estar presente y conservada por homología en todos los genomas del conjunto de interés, de

aquí en adelante llamado conjunto *target*

2. No debe tener homólogos significativos, en ningún genoma que no sea parte del conjunto target.

Una observación inmediata que surge de esta definición es que no es razonable validar nuestra región con todos los genomas secuenciados que existan, pero afortunadamente tampoco es necesario. Las secuencias de ADN no son aleatorias, sino el resultado del proceso evolutivo de los organismos. Las especies no aparecen espontáneamente, sino que son el resultado de la acumulación de mutaciones generación tras generación, lo cual permite definir una estructura de árbol evolutivo que agrupa a los organismos según su grado de parentesco.

Para asegurar entonces que una región sea realmente específica del conjunto *target*, no es necesario comparar contra todos los genomas secuenciados disponibles, sino que basta con analizar aquellos organismos más cercanos evolutivamente. Si una región no presenta homólogos significativos en estos parientes cercanos, es razonable asumir que su presencia en otros linajes más distantes (que acumulan aún más diferencias) será improbable. De aquí en adelante, llamaremos a este conjunto de vecinos evolutivos cercanos conjunto *neighbor*.

Finalmente, dado un conjunto *target* de individuos de interés y un conjunto *neighbors* de vecinos evolutivamente cercanos, consideramos que una región es única si:

1. Está presente por homología en todos los genomas del conjunto *target*.

2. Está ausente por homología en todos los genomas del conjunto *neighbor*.

La reacción en cadena de la polimerasa (PCR, por sus siglas en inglés) es una técnica fundamental en biología molecular que permite amplificar de manera exponencial fragmentos específicos de ADN. A partir de una región seleccionada del genoma —que denominamos marcador—, la PCR genera millones de copias, facilitando su detección incluso cuando la cantidad original de ADN es mínima. Por esta razón, la PCR es ampliamente utilizada en diagnósticos clínicos, ambientales y de investigación, donde el objetivo es detectar la presencia de un organismo específico dentro de una muestra biológica.

Para que la PCR sea efectiva como prueba diagnóstica, el marcador elegido debe cumplir un criterio esencial: debe estar presente únicamente en el organismo que se desea detectar y ausente en otros organismos potencialmente presentes en la muestra. En este contexto, las regiones únicas, tal como las definimos en nuestro trabajo, representan candidatos ideales para ser utilizadas como marcadores diagnósticos.

Es importante señalar que, en la naturaleza, el ADN no existe como una cadena lineal aislada, sino que adopta una estructura tridimensional establecida por la complementariedad de bases: la adenina (A) se aparea con la timina (T), y la citosina (C) con la guanina (G). Esta complementariedad molecular no solo da origen a la doble hélice del ADN, sino que constituye el principio que permite procesos fundamentales como la replicación celular y, en el caso que nos ocupa, la amplificación específica mediante PCR.

La selección precisa del marcador durante la PCR se logra mediante el diseño de pequeños fragmentos sintéticos de ADN denominados *primers*. Estos primers deben ser complementarios a los extremos del marcador y permiten que la enzima polimerasa reconozca y amplifique específicamente la región comprendida entre ellos. Si el organismo objetivo está presente en la muestra, los primers se unirán correctamente y la reacción producirá millones de copias del marcador, generando así una señal detectable. En ausencia del organismo, los primers no encontrarán el marcador y no se producirá amplificación.

El diseño de primers constituye un desafío adicional y crítico, ya que requiere optimizar múltiples parámetros bioquímicos y termodinámicos. Sin embargo, más allá de su diseño, la especificidad de la prueba depende fundamentalmente de la secuencia del marcador sobre la que se diseñan los primers. En otras palabras, una secuencia única garantiza que la amplificación solo ocurrirá si el organismo objetivo está presente, minimizando los falsos positivos.

Para evaluar experimentalmente la utilidad diagnóstica de los marcadores que encontramos mediante algoritmos **en datos reales**, sería necesario realizar reacciones PCR reales con muestras biológicas. Sin embargo, esto no siempre es viable en el contexto de este proyecto. Como alternativa, recurrimos a simulaciones *in silico* que replican el comportamiento esperado de una PCR: dados los genomas conocidos de distintas especies, simulamos el proceso de unión de primers y la posterior amplificación, permitiendo estimar si un marcador sería amplificado correctamente en presencia del organismo a detectar y si permanecería ausente en otras especies.

Estas simulaciones constituyen una aproximación práctica para validar los marcadores sobre datos reales, ya que permiten evaluar su especificidad y cobertura antes de su aplicación experimental. Así, podemos anticipar si las regiones únicas detectadas por los algoritmos cumplen su propósito diagnóstico dentro de un contexto biológico complejo.

A pesar de la relevancia de encontrar marcadores, el desarrollo de herramientas computacionales especializadas en su búsqueda no ha progresado al mismo ritmo que las herramientas generales para la búsqueda de similitud. Esta brecha motiva el presente trabajo, cuyo objetivo es realizar un estudio sistemático del problema de la búsqueda de regiones únicas, con un énfasis particular en su aplicación como marcadores para PCR. Para ello, se propone relevar las herramientas computacionales existentes, definir una metodología para su análisis comparativo y, a partir de este análisis, identificar sus limitaciones y posibles vías de mejora.

Existen pocas herramientas específicas para el problema planteado, actualmente el proceso de la búsqueda de marcadores implica generalmente el uso de múltiples herramientas de propósito general. En el panorama actual se destacan tres herramientas representativas que pueden ser aplicadas al problema `KEC`, `fur` y `GenMap` [3, 6, 5]. Las 3 emplean filosofías algorítmicas distintas, presentando un compromiso fundamental entre velocidad, consumo de recursos y robustez biológica.

`KEC` (K-mer Exclusion by Cross-reference) representa el enfoque basado en la descomposición de los genomas en k-meros (sub-secuencias de longitud k) y el uso de tablas hash para identificar y excluir aquellos k-meros del conjunto *target* que también están presentes en el conjunto *neighbor*. Su principal ventaja es su simplicidad y eficiencia computacional, permitiendo comparaciones de genomas completos en segundos.

`fur` (find unique regions), es una herramienta que plantea un enfoque que prioriza la robustez biológica, implementando un flujo de trabajo de múltiples fases que combina la eliminación de secuencias homólogas en el conjunto *neighbor* a través de un análisis heurístico y luego una validación final mediante el uso de `BLAST`. Este último paso, explota la capacidad de `BLAST` para detectar homología y garantiza que se cumplan las condiciones de unicidad.

Finalmente, `GenMap` se trata de una herramienta cuyo propósito principal no es la búsqueda de regiones únicas, sino calcular la métrica de (k, e)-mapeabilidad, que cuantifica cuán única es cada sub-secuencia de longitud k permitiendo hasta e errores (mutaciones) pero dentro de una misma secuencia. Sin embargo, es posible aplicar `GenMap` a nuestro problema interpretando la salida de manera adecuada.

El presente trabajo aborda de manera sistemática el problema de la búsqueda de regiones únicas, poniendo especial énfasis en su aplicación como marcadores para reacciones PCR. El objetivo principal es superar las limitaciones de las soluciones existentes y cubrir brechas de conocimiento previamente identificadas en este campo. En este contexto, se destacan las siguientes contribuciones principales:

1. **Desarrollo de una metodología comparativa sistemática:** Se diseñó y ejecutó un marco integral de evaluación para comparar el rendimiento de las herramientas `fur`, `KEC` y `GenMap`. Esta metodología contempla tanto datos genómicos simulados —donde las regiones únicas son conocidas— como datos biológicos reales provenientes de patógenos de relevancia clínica. La evaluación incluye tanto aspectos de eficiencia computacional (tiempo y memoria) como la calidad biológica de los marcadores identificados, medida mediante métricas de exactitud y simulaciones de reacciones PCR *in silico*.

2. **Diseño e implementación de un algoritmo mejorado:** Como principal aporte técnico, se desarrolló y evaluó `KEC-FM`, una re-implementación optimizada del algoritmo `KEC`. Esta nueva versión sustituye la estructura de datos original basada en tablas hash por un índice FM (FM-index)[2], lo que mejora significativamente la escalabilidad y permite una exploración más flexible y eficiente de diferentes longitudes de k-meros sin necesidad de reconstruir el índice. De este modo, se aborda una de las principales limitaciones operativas presentes en la implementación original de `KEC`.

3. **Desarrollo de herramientas para filtrar y refinar marcadores:** Se implementaron tres herramientas complementarias para mejorar la calidad de los marcadores. Una permite recortar las regiones no únicas de cada secuencia utilizando `BLAST` contra genomas vecinos, conservando únicamente el núcleo único. Otra realiza un refinamiento por etapas, filtrando los marcadores presentes en genomas *neighbor* y confirmando su presencia en los genomas target, generando un informe con métricas de unicidad (presentadas en la sección 3.2). La tercera elimina redundancias entre marcadores candidatos, agrupando secuencias homólogas y seleccionando un representante por grupo para obtener un conjunto final no redundante.

El resto del documento se organiza de la siguiente manera: en el capítulo 2 se presentan los resultados de un relevamiento de la literatura en la búsqueda de regiones únicas y se presentan 3 herramientas representativas `fur`, `KEC` y `GenMap`.

El capítulo 3 detalla la metodología diseñada para realizar una comparación sistemática de las herramientas. Se definen métodos para simulación de datos, recolección de datos reales y se plantean las métricas en las que se basa la comparación.

El capítulo 4 expone y analiza los resultados obtenidos de los experimentos sobre datos simulados, identificando fortalezas y debilidades de cada algoritmo. Seguido a esto, evaluamos los algoritmos sobre un conjunto de datos reales y ponemos a prueba los marcadores mediante una simulación de PCR.

Con base en los resultados experimentales, en el capítulo 5, proponemos y evaluamos una versión mejorada de `KEC`, comparándola con su versión original.

Finalmente, el capítulo 6 resume las conclusiones del trabajo y plantea las líneas de trabajo futuras que se desprenden de este estudio.

Capítulo 2

Antecedentes en la búsqueda de marcadores

En este capítulo, presentamos el resultado de una revisión de la literatura existente. Detallamos los métodos computacionales existentes para abordar el problema de la identificación de marcadores genéticos de diagnóstico. Para cada algoritmo y herramienta relevante, se describen sus aspectos operativos más importantes, las estructuras de datos fundamentales que utiliza, sus aplicaciones específicas y las posibles modificaciones o consideraciones necesarias para su aplicación al problema planteado. El objetivo es proporcionar una base sólida para comprender el panorama actual de las metodologías disponibles para la búsqueda de marcadores genéticos en conjuntos de genomas.

2.1. Estructuras de datos fundamentales

Antes de profundizar en los algoritmos comparados, es importante destacar que el núcleo del problema consiste en la búsqueda eficiente de subcadenas dentro de textos masivos, como los genomas. Dado el tamaño de estos datos, los métodos de fuerza bruta resultan computacionalmente inviables. Por ello, se utilizan estructuras de datos tipo índice, que permiten realizar búsquedas rápidas tras un procesamiento previo del texto.

Cada uno de los tres algoritmos evaluados emplea una estructura de datos diferente. A continuación, se describen sus características clave y se comparan en el contexto de la búsqueda genómica.

2.1.1. Arreglo de Sufijos Mejorado

Históricamente, el Árbol de Sufijos (Suffix Tree) [24] ha sido la estructura de datos clásica para resolver problemas de procesamiento de cadenas. Permite buscar un patrón de longitud m en tiempo $O(m)$, independientemente del tamaño del texto n. Sin embargo, su elevado consumo de memoria lo hace poco práctico para genomas completos.

Para superar esta limitación, se desarrolló el Arreglo de Sufijos (Suffix Array, SA) [15], que almacena las posiciones iniciales de los sufijos del texto ordenadas lexicográficamente. El SA reduce significativamente el consumo de memoria. La búsqueda básica se realiza mediante búsqueda binaria, con complejidad $O(m \log n)$.

Esta eficiencia se mejora al incorporar el Arreglo de Prefijos Comunes Más Largos (LCP), que almacena la longitud del prefijo común entre sufijos consecutivos en el SA. Esta información permite reducir la complejidad de búsqueda a $O(m + \log n)$ en la práctica.

El conjunto formado por el SA, el LCP y otras tablas auxiliares constituye el Arreglo de Sufijos Mejorado (ESA por sus siglas en inglés), similar a un Árbol de Sufijos, pero con menor consumo de memoria y mejor desempeño en caché, lo cual resulta ideal para aplicaciones a gran escala.

La construcción de un ESA completo tiene un costo temporal de $O(n \log n)$ y requiere recorrer el texto y construir tanto el SA como el LCP, siendo la etapa de ordenamiento la más costosa.

2.1.2. La transformada de Burrows-Wheeler (BWT)

La Transformada de Burrows-Wheeler (BWT) es una permutación reversible de un texto, inicialmente diseñada como etapa previa a la compresión sin pérdidas. La BWT no comprime el texto por sí misma,

sino que lo reorganiza, agrupando caracteres similares para facilitar la compresión posterior.

Su construcción consiste en:

1. Añadir un símbolo centinela (`$`), lexicográficamente menor que cualquier otro carácter.
2. Generar todas las rotaciones cíclicas del texto con el centinela.
3. Ordenar estas rotaciones lexicográficamente.
4. Tomar el último carácter de cada rotación ordenada para formar la BWT.

Computacionalmente, la BWT puede obtenerse de forma eficiente a partir del Suffix Array en tiempo $O(n)$.

2.1.3. Índice FM

La BWT sirve de base para construir el **Índice FM**, una estructura de datos comprimida que permite realizar búsquedas eficientes. El Índice FM está compuesto por:

- La BWT del texto.
- El Suffix Array (SA), completo o muestreado.
- Una tabla de conteos C, que indica cuántos caracteres menores que un símbolo dado aparecen en el texto.
- Estructuras auxiliares (milestones), que permiten acelerar la reconstrucción de posiciones y el cómputo de rangos.

El Índice FM soporta las siguientes operaciones clave:

- `Count(P)`: Cuenta las ocurrencias exactas de un patrón `P` en tiempo $O(|P|)$, que representa una mejora con respecto al $O(|P| \log n)$ del SA.
- `Locate()`: Recupera las posiciones de las ocurrencias en tiempo esperado $O(|P| + k)$, donde k es el número de ocurrencias.

La construcción completa del Índice FM, incluyendo la BWT, el SA y las estructuras auxiliares, suele realizarse en tiempo $O(n \log n)$, similar al ESA, dado que ambas estructuras requieren ordenar sufijos.

Existen variantes más avanzadas, como el Índice FM bidireccional, utilizado en herramientas como `GenMap`. Esta variante permite realizar búsquedas extendidas en ambas direcciones desde cualquier punto del patrón, una capacidad esencial para búsquedas aproximadas, ya que permite dividir el patrón y buscar sus partes de forma independiente y eficiente.

2.1.4. Tablas Hash y mapas de k-meros

Una estrategia alternativa es la utilización de tablas hash para almacenar k-meros, que son sub-cadenas de longitud fija k. Cada k-mero actúa como clave asociada a valores como su frecuencia o sus posiciones en el genoma.

La principal ventaja de este enfoque es su rapidez: tanto las inserciones como las consultas tienen una complejidad promedio de $O(1)$. Sin embargo, presenta limitaciones:

- Solo permite buscar coincidencias exactas de longitud k.
- El consumo de memoria puede ser elevado, especialmente si hay muchos k-meros únicos, dado que cada uno debe almacenarse explícitamente.

La construcción de una tabla hash suele ser rápida ($O(n)$ si se recorren los n caracteres del texto una sola vez para extraer todos los k-meros), pero el costo exacto depende del tamaño de k y de la cantidad de k-meros únicos que se generen.

2.1.5. Comparación de estructuras de Datos

La tabla 2.1 resume las características comparativas de estas estructuras.

Tabla 2.1: Tabla Comparativa de Estructuras de Datos

Estructura	Características
Árbol de Sufijos	
Espacio	Alto ($O(n)$, típicamente $\geq 20n$ bytes)
Ventajas	Muy versátil, soporta muchas operaciones complejas.
Desventajas	Muy demandante de memoria.
ESA	
Espacio	Moderado ($O(n)$, típicamente $8n$ bytes)
Ventajas	Gran balance entre espacio y velocidad, cache-friendly.
Desventajas	Más complejo de implementar que un SA básico.
Tabla Hash	
Espacio	Variable, puede ser alto ($O(\text{num\_k-meros\_unicos})$)
Ventajas	Extremadamente rápido para búsquedas exactas de k-meros.
Desventajas	Limitado a k-meros de longitud fija, solo búsquedas exactas.
Índice FM	
Espacio	Bajo (comprimido, cercano al texto)
Ventajas	Altamente comprimido, ideal para genomas muy grandes.
Desventajas	Más lento que otras estructuras para ciertas operaciones.

2.2. Métodos

En esta sección se describen en detalle las soluciones más relevantes del estado del arte. Para garantizar la claridad y consistencia a lo largo de este documento, adoptaremos una nomenclatura y una entrada comunes. La entrada utilizada por todos los algoritmos debe ser los dos conjuntos definidos en la introducción:

- **Target**: Se refiere al conjunto de uno o más genomas en los que se desea encontrar las secuencias conservadas.
- **Neighbors**: Se refiere al conjunto de uno o más genomas de los que se desea excluir secuencias.

Además, es importante aclarar que todas las herramientas analizadas en este contexto requieren que estas secuencias sean ingresadas en el formato `FASTA`. Cada secuencia en un archivo FASTA comienza con una línea de descripción única, también conocida como línea de cabecera. Esta línea se distingue de las líneas de datos de secuencia por un símbolo de mayor que (>) en la primera posición. La palabra que sigue inmediatamente a este símbolo se interpreta como el identificador único de la secuencia. El resto de la línea de cabecera puede contener una descripción opcional de la secuencia.

Debajo de la línea de cabecera, se encuentran una o más líneas que contienen las letras que representan la secuencia biológica (ADN, ARN o proteínas). A continuación, se presenta un ejemplo simple de un archivo FASTA con dos secuencias de ADN:

```
>seq1 Descripción opcional de la secuencia
ATGCGTACGTAGCTAGCTAGCTAGCTAGCTA
CGTAGCTAGCTAGCTAGCTGACT
>seq2
GCTATGCGTACGTAGCTAGCTAGCCCCCTT
```

2.2.1. fur: find unique regions

Descripción general del algoritmo

La herramienta `fur` [6] no es un único algoritmo, sino un pipeline compuesto por varios pasos que se ejecutan de manera secuencial. Su enfoque es riguroso y está diseñado para producir regiones únicas de alta confianza. El proceso se puede describir en tres pasos principales:

1. **Sustracción:** En el primer paso, `fur` busca todas las sub-secuencias de un único genoma *target* representante que no están presentes en el conjunto de genomas *neighbors*. Durante este paso se elimina cualquier porción del *target* que tenga una contraparte idéntica en cualquiera de los *neighbors*.

El resultado es un conjunto de secuencias candidatas que son, en principio, únicas para el *target* representante. Esta sustracción se realiza realizando búsquedas sobre ESAs construidos a partir de los genomas neighbor.

2. **Intersección:** A continuación, se toman las secuencias candidatas de la fase anterior y se verifica si también están presentes en el resto de los genomas del conjunto *target* (si es que hay más de uno). Solo aquellas secuencias que se encuentran en todos los genomas *target* superan esta fase. Este paso asegura la conservación del marcador dentro del grupo de interés.

3. **Validación Final:** El último paso es una validación de homología mediante `BLAST`. Las regiones candidatas que sobrevivieron a las fases anteriores se comparan nuevamente con los genomas *neighbors*, pero esta vez se utiliza una búsqueda de similitud en lugar de identidad exacta. Esto es crucial para descartar candidatos que, aunque no sean 100 % idénticos a una secuencia *neighbor*, sean lo suficientemente parecidos como para generar una reacción cruzada en un ensayo de diagnóstico.

Construcción de indices y preparación de los datos

La ejecución de los pasos anteriores no se lleva a cabo sobre los datos crudos, sino que se basan en una serie de insumos preparados por un programa distinto llamado `makeFurDb`. La separación de esta preparación y la ejecución es una estrategia de optimización que amortigua el coste computacional del indexado, permitiendo ejecutar `fur` múltiples veces con diferentes parámetros sin repetir el trabajo pesado. La principal tarea de `makeFurDb` es calcular las "estadísticas de coincidencias"o "matching statistics", la cual es el insumo principal para el primer paso de fur.

Para esto se compara una secuencia *target* representante r (por defecto, la más corta de los *targets*) con cada una de las secuencias *neighbors* para identificar todas las regiones de coincidencia. El resultado es un "arreglo de coincidencias"(`en`), este arreglo es un vector binario diseñado para marcar las posiciones donde terminan las coincidencias entre r y las secuencias *neighbors*.

Su cálculo se realiza de la siguiente forma:

1. Se identifican todas las subcadenas ("match factors") de la secuencia de referencia que existen en cualquiera de las secuencias *neighbor*.

2. Para cada posición de inicio en la secuencia de referencia, se determina la longitud de la coincidencia más larga encontrada.

3. El arreglo en se construye marcando con un 1 cada posición que corresponde al final de una de estas coincidencias. Todas las demás posiciones se marcan con un 0.

Durante este cálculo es necesario realizar muchas búsquedas sobre el conjunto *neighbor* y para ello se utiliza un Enhanced Suffix Array (ESA).

`makeFurDb` también crea una base de datos `BLAST` a partir de todas las secuencias *neighbor* y separa la secuencia representativa r del resto de los *targets* preparando todo para el paso final de `fur`.

El funcionamiento de `makeFurDb` y `fur` es complejo, por lo que en el anexo A.1 se detalla con un ejemplo el cálculo del arreglo `en` y su posterior uso.

Implementación del pipeline

Una vez que `makeFurDb` ha preparado la base de datos, el programa `fur` ejecuta el algoritmo central descrito en la sección 2.2.1.

Paso I - Sustracción Inicial Este paso se realiza mediante un análisis de ventana deslizante sobre el arreglo `en`. El propósito explícito del análisis de ventana deslizante es encontrar intervalos donde el número de coincidencias (marcados con 1 en el vector) es mayor que un umbral t (modificable por parámetro); sencillamente se desliza la ventana y se verifica si la suma de los valores dentro de la ventana (nm) cumple la condición $nm >= t$. Si es así, se considera a esa ventana como de "alta complejidad". Estas regiones solapadas se convierten en las candidatas que pasan a los siguientes pasos del pipeline.

Una región "simple" sería aquella que tiene una coincidencia muy larga y continua con un vecino. Esa zona es claramente no-única y de poco interés. Mientras que una región de "alta complejidad", en este contexto, es una zona del genoma de referencia que está cubierta por muchas coincidencias cortas y diferentes con los genomas vecinos. Esta falta de coincidencias muy largas hace que sea candidata a ser única.

Por supuesto, el análisis de ventana requiere elegir el tamaño de la misma; este es el parámetro principal de `fur`.

El enfoque descrito es una innovación algorítmica reciente y es altamente escalable, ya que el requisito de memoria depende del vecino más grande, no de la suma de todos ellos. En la versión original de `fur` se producía un cuello de botella, ya que usaba una ejecución del programa externo `Macle` [17] y requería construir un único índice masivo con un alto consumo de RAM, limitando su aplicación a gran escala [16].

Paso II - Intersección Los marcadores tentativos m, derivados únicamente de r, se intersectan con el resto de las secuencias objetivo $(T \setminus \{r\})$ para asegurar que solo se conserven las regiones presentes en todos los objetivos. Formalmente, $m \leftarrow m \cap (T \setminus \{r\})$.

Esta tarea se realiza con el programa `Phylonium` [14], que realiza un alineamiento múltiple aproximado y eficiente. En lugar de un alineamiento completo, `Phylonium` identifica regiones "ancla"conservadas, una estrategia que prioriza la velocidad y la escalabilidad, ideal para genomas completos.

Paso III - Sustracción Final Para mejorar la especificidad, las regiones candidatas que sobreviven a la intersección se someten a una segunda y más rigurosa sustracción. Se utiliza `blastn` [1] para comparar cada marcador candidato contra la base de datos de vecinos pre-calculada por `makeFurDb`. Cualquier marcador que muestre una similitud significativa es eliminado. Este paso final de validación a nivel de secuencia es crucial para eliminar falsos positivos y refinar el conjunto final de marcadores únicos.

Fortalezas y debilidades de `fur`

Fortalezas

- **Alta especificidad:** El enfoque de tres pasos, que culmina con una validación rigurosa mediante `BLAST`, asegura una alta confianza en los marcadores resultantes, minimizando la probabilidad de falsos positivos.
- **Escalabilidad y eficiencia de memoria (versión nueva):** La versión más reciente de fur ha sido optimizada para que su consumo de memoria escale con el tamaño del genoma vecino más largo, en lugar del tamaño total del vecindario. Esto la hace altamente escalable y apta para análisis a gran escala en hardware convencional.
- **Velocidad:** A pesar de su enfoque riguroso, el uso de un ESA para la sustracción inicial y la separación de la construcción del índice (`makeFurDb`) del pipeline principal (fur) hacen que el proceso sea rápido y eficiente.

Debilidades

- **Menor sensibilidad:** `fur` tiende a ser conservador y a producir un número menor de marcadores candidatos evitando producir marcadores que puedan llevar a falsos positivos. Su alta especificidad se logra a costa de una menor sensibilidad.
- **Complejidad del pipeline:** Al ser un flujo de trabajo que integra herramientas externas como `Phylonium` y `blastn`, su funcionamiento depende de la correcta interacción de múltiples componentes, lo que puede introducir puntos de fallo.
- **Dependencia de un genoma representativo:** El paso inicial de sustracción se basa en un único genoma representativo del grupo *target*. Aunque esta es una heurística efectiva, una elección inadecuada del representante podría teóricamente sesgar los resultados.

2.3. `KEC`: Búsqueda rápida de secuencias únicas por exclusión de k-meros

K-mer Exclusion by Cross-reference (`KEC`) es una aplicación de software diseñada para permitir a los usuarios encontrar de manera rápida y sencilla secuencias únicas (de aminoácidos o ácidos nucleicos) mediante la comparación de los K-meros de las secuencias *targets* y *neighbors* [3]. La motivación original declarada para `KEC` es la búsqueda de secuencias de ADN suficientemente únicas dentro de genomas bacterianos para el diseño de primers de PCR para diagnóstico.

2.3.1. Metodología y Funcionamiento

`KEC` tiene dos modos de ejecución, inclusión y exclusión. El modo de exclusión es el que mejor se adapta a encontrar regiones únicas y sigue los pasos que se describen a continuación:

1. **Creación de *k-meros*:** En la primera fase, el programa genera *k-meros* tanto a partir de las secuencias *target* como de las *neighbor*.
2. **Referencia Cruzada:** Los *k-meros* de ambos conjuntos se almacenan en hash maps (mapas nativos de Go), un mapa para cada conjunto. Posteriormente, estos dos conjuntos de *k-meros* son comparados o "referencias cruzadas". En este paso, todos los *k-meros* del conjunto *target* se buscan en el mapa de neighbors y son eliminados si efectivamente se encuentran.
3. **Reconstrucción de Secuencias:** Los *k-meros* del conjunto *target* que sobreviven al filtro son reubicados en su posición original dentro de la secuencia de origen. Los *k-meros* que se superponen (aquellos que comparten al menos una posición en común) son fusionados para crear secuencias únicas de mayor longitud.
4. **Devolución del Resultado Final:** El resultado es un archivo que contiene las secuencias recuperadas del paso anterior, las cuales cumplen el requisito de no estar presentes en el conjunto de secuencias *neighbor*. El usuario puede especificar la longitud mínima que deben tener estas secuencias únicas resultantes para ser incluidas en el archivo.

De forma alternativa, el modo inclusión permite invertir la lógica de la referencia cruzada para, en lugar de excluir, conservar los *k-meros* que están presentes tanto en el conjunto *target* como en el *non-target*. En este modo, el resultado son las secuencias compartidas entre ambos conjuntos de datos.

Aplicación a la búsqueda de regiones únicas

La principal fortaleza de `KEC` es su extraordinaria velocidad, gracias al uso de los mapas (hash tables) de Go que permiten inserción y búsqueda en orden constante.

`KEC` no pretende ser una solución integral para una tarea específica, sino una alternativa conveniente, simple y rápida dentro de flujos de trabajo ya establecidos. Permite que investigadores sin una alta especialización en bioinformática puedan analizar secuencias genómicas fácilmente, sin la necesidad de aprender a manejar sistemas operativos o herramientas complejas. El resultado del programa es útil como punto de partida para análisis posteriores.

2.3.2. `GenMap`: Calculo exacto de la mapeabilidad

`GenMap` [5] fue diseñado para calcular de manera exacta la métrica denominada **(k, e)-mapeabilidad**, la cual mide qué tan única es cada sub-secuencia de longitud k (*k-mero*) dentro de un genoma, permitiendo hasta e errores (intercambios de una base por otra, inserciones o eliminaciones). Un valor de mapeabilidad alto indica que una región es poco frecuente, mientras que un valor bajo señala una región repetitiva dentro de un mismo genoma. Esta medida es útil en otros problemas de biología que involucran la evaluación de repetitividad genómica. `GenMap` es un algoritmo diseñado específicamente para calcular esta métrica de manera exacta y eficiente.

Metodología de Cálculo de `GenMap`

Para calcular la mapeabilidad, `GenMap` sigue un proceso de dos fases similar al de fur: primero, construye un índice de la secuencia de entrada y, segundo, utiliza este índice para encontrar eficientemente todas las ocurrencias aproximadas de cada k-mero.

Definición Formal La mapeabilidad se define en base a la noción de frecuencia. La **(k,e)-frecuencia** de un k-mero T_i (que inicia en la posición i de una secuencia T) es el número de k-meros T_j que coinciden con T_i con hasta e errores. Esta frecuencia se almacena en un vector F:

$$F[i] = |\{j | D(T_i, T_j) \leq e, 1 \leq j \leq n - k + 1\}|$$

donde $D(T_i, T_j)$ es la distancia (p. ej., de Hamming) entre los dos k-meros. La **(k,e)-mapeabilidad**, M, es simplemente la inversa de la frecuencia:

$$M[i] = \frac{1}{F[i]}$$

Un valor de $M[i] = 1$ implica que el k-mero es perfectamente único bajo los parámetros definidos dentro de la secuencia analizada.

Optimizaciones Algorítmicas Realizar esta búsqueda para cada k-mero del genoma es computacionalmente intensivo. La eficiencia de `GenMap` proviene de varias innovaciones clave:

- **Índice FM Bidireccional:** Utiliza como base un índice FM bidireccional, que permite búsquedas de patrones extremadamente rápidas hacia adelante y hacia atrás en la secuencia [7].
- **Esquemas de Búsqueda Óptimos:** Emplea algoritmos avanzados que dividen el k-mero y los errores permitidos en fragmentos más pequeños. Estos se buscan de manera estratégica en el índice para minimizar el espacio de búsqueda y acelerar el hallazgo de coincidencias aproximadas [8].
- **Reutilización de Cómputos:** Aprovecha el solapamiento entre k-meros adyacentes (T_i y T_{i+1}) para reutilizar estados de búsqueda, evitando re-calcular la información compartida. Además, cada secuencia de k-mero distinta se busca en el índice una sola vez, y los resultados se aplican a todas sus copias en el genoma.

Aplicación a la búsqueda de regiones únicas

`GenMap` no identifica directamente regiones únicas, es importante destacar que las medidas de frecuencia y mapeabilidad son calculadas para una única secuencia. Es posible usar `GenMap` sobre un conjunto de ellas, pero estas son concatenadas y tratadas como una sola.

De todas formas, para calcular estas medidas, `GenMap` debe buscar, contar e identificar la posición y la secuencia de origen en la que se halla cada k-mero. El programa, además, admite varios formatos de salida, siendo uno de ellos una tabla (en formato CSV) que contiene, para cada k-mero de entrada, una lista de las coordenadas exactas de todas sus ocurrencias (con hasta e errores).

Este resultado nos sirve como la base para un post-procesamiento para seleccionar los k-meros que cumplen el criterio de unicidad (estén presentes en todos los *target* y ausentes de todos los neighbors) y, más aún, sus coordenadas pueden ser agrupadas o fusionadas para definir las regiones contiguas únicas de interés.

Por defecto `GenMap` genera un archivo csv para cada uno de los archivos fasta indexados originalmente, esto es un gran problema en nuestro caso porque el proceso de crearlos es muy costoso. Para que la comparación sea justa, en el marco de este proyecto agregamos al programa una nueva opción que nos permite solo calcular un único archivo csv para un genoma original, esto reduce drásticamente el tiempo de cálculo ya que nos evita cómputos innecesarios para nuestro problema.

Es importante destacar:

1. Es posible utilizar valores de K más grandes que los que se utilizarían por ejemplo en `KEC` y sencillamente utilizar esos k-meros, sin fusionar, como candidatos a marcadores. Es por esto que mantenemos el procesamiento de la salida como una herramienta a parte del programa, permitiendo ambos modos de uso.
2. El modo de operación donde fusionamos los k-meros es distinto al de `KEC`, ya que este ultimo no cerciora que los k-meros estén presentes en todos los genomas targets. Si deseáramos emular el comportamiento de `KEC` con la salida de `GenMap` seria necesario generar y procesar todos los archivos csv del conjunto target. Como ya mencionamos consideramos que esto es sumamente costoso y, de cierta manera, redundante.

2.2 Resumen de comparación de los métodos

A partir de las descripciones de las herramientas, se puede establecer una comparación basada en su enfoque, su principio de funcionamiento y el tipo de resultado que producen. Cada herramienta aborda el problema de la identificación de regiones únicas desde una perspectiva diferente, presentando distintos compromisos entre velocidad, consumo de recursos, precisión y necesidad de post-procesamiento. En la tabla 2.2 se resumen las características clave de cada método analizado.

	`fur`	`KEC`	`GenMap`
Principio Algorítmico	Sustracción de secuencias homólogas y validación por alineamiento.	Exclusión de k-meros exactos por referencia cruzada.	Cálculo de la mapeabilidad (k, e) basado en frecuencia de k-meros.
Estructura de Datos Principal	Array de Sufijos Mejorado (ESA), base de datos `BLAST`.	Tablas Hash (mapas de Go).	Índice FM bidireccional.
Manejo de Homología	Explícito y riguroso. Utiliza `BLAST` para detectar similitud de secuencia, no solo identidad.	Implícito y estricto. No detecta homología; solo coincidencias exactas de k-meros.	Parametrizable. Permite errores (mismatches) para detectar homología cercana.
Salida	Conjunto refinado de regiones candidatas únicas de alta confianza.	Secuencias largas formadas por k-meros únicos contiguos.	Mapa cuantitativo de mapeabilidad por cada posición del genoma. Requiere post-procesamiento.

Tabla 2.2: Tabla comparativa en los aspectos claves de los 3 métodos relevados

Capítulo 3

Metodología

En este capítulo, se detallan los métodos y materiales empleados para llevar a cabo la evaluación y comparación de los 3 métodos relevados. Se describen los criterios utilizados para evaluar el desempeño, el diseño de los experimentos, los procesos de generación de datos simulados y la recolección de datos genómicos reales.

3.1. Diseño experimental sobre datos simulados

Para llevar a cabo una comparación exhaustiva y para calibrar parámetros de manera efectiva resulta más práctico utilizar datos genómicos simulados, ya que nos permiten tener el control de las regiones únicas presentes.

3.1.1. Simulación simple con marcador conocido

Definimos una metodología para generar un *target* y un *neighbor* simples a los que se les introdujo artificialmente una única región genómica diferencial de longitud y posición conocida. Esta configuración permite comparar directamente la salida de los algoritmos de identificación de regiones únicas con un valor de referencia exacto (ground truth).

La simulación se llevó a cabo utilizando herramientas del paquete BioBox [12] de la siguiente manera:

1. Generamos una secuencia *target* aleatoria de 10 kb (10000 bases) de longitud mediante el programa `ranSeq`.
2. Generamos una secuencia *neighbor* copiando la secuencia *target* creada anteriormente y eliminando un segmento de la misma (con ubicación y longitud conocidas)

Como resultado de este procedimiento, la secuencia *target* contiene una región que no posee homología en la secuencia *neighbor*, constituyendo así la región única esperada. Dado que la secuencia base generada es aleatoria, la probabilidad de encontrar repeticiones intragenómicas significativas que pudieran confundir a los algoritmos es baja.

3.1.2. Generación de escenarios simulados realistas con `stan`

Generamos datos genómicos simulados que imitan escenarios reales de descubrimiento de marcadores diagnósticos utilizando `stan` [13]. El simulador sigue estos pasos fundamentales:

1. **Creación de una historia evolutiva simulada:** Primero, `stan` construye una estructura de "árbol genealógico" que representa la historia evolutiva de los dos grupos de organismos (*target* y *neighbor*).
2. **Simulación de la evolución del ADN:** A continuación, el programa simula cómo las secuencias de ADN evolucionan sobre de las ramas de este árbol. Comenzando con una secuencia de ADN ancestral hipotética en la raíz e introduciendo cambios aleatorios (mutaciones) siguiendo las ramas.
3. **Introducción de marcadores:** Finalmente `stan` modifica una región de ADN específica exclusivamente en el grupo de los *neighbor*. Por defecto, la región del marcador es eliminada por completo

de las secuencias vecinas. Como alternativa, también puede ser alterada mediante una alta tasa de mutaciones.

El resultado final es un conjunto de datos donde las secuencias del grupo *target* contienen una región genética específica que está ausente en todas las secuencias del grupo neighbor.

La figura 3.1 muestra la estructura de un ejemplo generado para 10 *targets* y 10 *neighbors*.

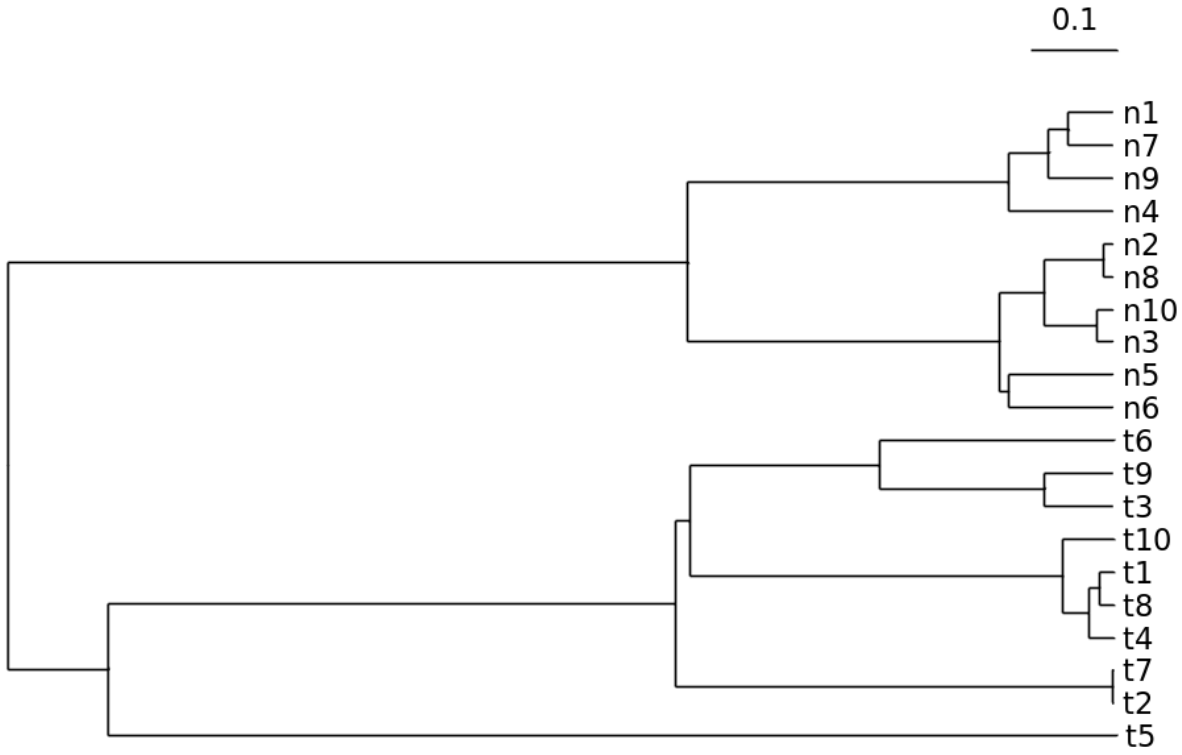

Figura 3.1: Ejemplo de árbol generado por stan

Cabe destacar que `stan` nos permite generar cualquier cantidad de *targets* y *neighbors* aunque su longitud debe ser la misma. También permite especificar la posición de las regiones únicas introducidas.

3.2. Métricas de Calidad de Marcadores en Datos simulados

Para evaluar la calidad de las regiones genómicas únicas (para su uso como marcadores PCR) que predecimos, consideramos dos enfoques complementarios que han sido utilizados en la literatura [6][16]. Ambos se basan en la misma fórmula utilizada en la literatura de predicción de genes [10] pero difieren en cómo calculan los valores involucrados.

El primer enfoque evalúa el rendimiento de un marcador frente a colecciones de genomas *target* y *neighbor* utilizando `BLAST` para buscar coincidencias. El segundo compara una predicción de la posición de una región única (producida por uno de los algoritmos) contra un ground truth previamente definido en una secuencia específica.

A continuación detallamos ambos métodos.

3.2.1. Verificación de la definición de unicidad mediante `BLAST`

Este método mide qué tan bien un marcador predicho cumple el doble criterio de unicidad utilizando `BLAST` para calcular los valores clásicos de una matriz de confusión, t_p, f_p, f_n y t_n.

- t_p **(Verdaderos Positivos):** Es el número total de bases del marcador que alinean contra el conjunto de genomas target. Se calcula sumando las longitudes de todos los *hits* de `BLAST` del marcador en los genomas target.
- f_p **(Falsos Positivos):** Es el número total de bases del marcador que alinean contra el conjunto de genomas neighbor.

- f_n **(Falsos Negativos):** Son las bases del marcador que *deberían* haberse encontrado en los genomas *target* pero no se encontraron. Se calcula como la longitud total esperada de alineamientos (longitud del marcador multiplicada por el número de genomas objetivo) menos los t_p.
- t_n **(Verdaderos Negativos):** Son las bases que correctamente *no* se encontraron en los genomas neighbor. Se calcula como la longitud total esperada de alineamientos en los vecinos (longitud del marcador multiplicada por el número de genomas vecinos) menos los f_p.

A partir de estos valores podemos calcular:

Sensibilidad (S_n) Mide la capacidad del marcador para detectar los genomas target. Se define como:

$$S_n = \frac{t_p}{t_p + f_n}$$

Especificidad (S_p) Mide la capacidad del marcador para excluir a los genomas neighbor. En este contexto, se define como la proporción de aciertos en los genomas objetivo frente al total de aciertos en ambos conjuntos.

$$S_p = \frac{t_p}{t_p + f_p}$$

Métrica Combinada (C) Para obtener una medida única de la calidad del marcador que considera tanto la sensibilidad como la especificidad, se utiliza el Coeficiente de Correlación de Matthews (C). Un buen marcador debe tener altos valores de S_n y S_p simultáneamente, y C captura esto en un solo valor. La fórmula es:

$$C = \frac{t_p t_n - f_p f_n}{\sqrt{(t_p + f_p)(t_n + f_n)(t_n + f_p)(t_p + f_n)}} \tag{3.1}$$

Este enfoque es costoso, debido a la necesidad de generar bases de datos de `BLAST`, pero es útil para validar la utilidad práctica de un marcador en un escenario de diagnóstico incluso si la posición de una posible región única es desconocida.

2. Evaluación Basada en Exactitud de Predicción

Este segundo método, utilizado en una publicación más reciente [16] no evalúa el marcador contra colecciones de genomas, sino que mide la exactitud de una única predicción comparándola con el ground truth, bien definido dentro de una secuencia de longitud conocida.

Dado:

- Una secuencia objetivo de longitud l.
- Una región única real con inicio ts y fin te.
- Una región predicha con inicio ps y fin pe.

Los términos se definen así:

- t_p **(Verdaderos Positivos):** El número de nucleótidos que están **dentro tanto de la región predicha como de la región verdadera**. Son las posiciones correctamente identificadas como parte de la región única.
- f_p **(Falsos Positivos):** El número de nucleótidos que están **en la región predicha pero fuera de la región verdadera**. Son posiciones que el programa identificó incorrectamente como únicas.
- f_n **(Falsos Negativos):** El número de nucleótidos que están **dentro de la región verdadera pero no en la región predicha** . Son posiciones de la región única que el programa no logró identificar.
- t_n **(Verdaderos Negativos):** El número de nucleótidos que están **fuera tanto de la región predicha como de la región verdadera**. Son las posiciones correctamente identificadas como no pertenecientes a la región única.

Con estos valores se pueden calcular los mismos cocientes de sensibilidad, especificidad y coeficiente de correlación que con la metodología anterior. Pese a que la fórmula es la misma, con fines de claridad a la hora de mencionar ambas métricas (y mantenernos alineados con las publicaciones), nos referiremos al coeficiente combinado calculado con esta metodología como Acc.

$$Acc = \frac{t_p t_n - f_p f_n}{\sqrt{(t_p + f_p)(t_n + f_n)(t_n + f_p)(t_p + f_n)}} \tag{3.2}$$

Esta métrica se puede calcular de manera muy rápida, pero requiere tener un ground truth. Además, no mide la calidad de una región para usarse como marcador, sino la capacidad de un algoritmo para producir una predicción perfecta.

Uso en este trabajo

Si bien tanto C como Acc buscan ofrecer una visión global del desempeño de un marcador sobre el conjunto de datos en el que fueron calculados, presentan diferencias conceptuales y prácticas relevantes. El criterio que seguiremos es utilizar Acc en escenarios simples con un único marcador conocido y bien definido. En escenarios complejos y/o con múltiples marcadores donde nos interese evaluar la calidad del marcador se le dará mayor relevancia a la métrica C .

3.3. Diseño experimental sobre datos reales

Se utilizaron los genomas de los bancos de datos de NCBI [20] para realizar experimentos sobre datos reales. Un problema frecuente con las bases de datos de genes es que organizan la información principalmente por taxonomía, un sistema de clasificación jerárquico (reino, filo, clase, etc.) cuyas categorías se basan históricamente en características observables de los organismos. Si bien es posible utilizar esta estructura jerárquica para seleccionar genomas y agruparlos como targets y neighbors para un ensayo, es necesario notar que el propósito de la clasificación taxonómica es catalogar la diversidad biológica, no necesariamente reflejar las relaciones evolutivas precisas.

En contraste, nuestro objetivo es comparar al grupo *target* contra sus parientes más cercanos a nivel de código genético. Y, por lo tanto, necesitamos que nuestros conjuntos de datos reflejen la historia evolutiva real y las relaciones de descendencia desde ancestros comunes. A esta organización basada en la derivación genética se la conoce como filogenética.

Es muy frecuente que la clasificación taxonómica no coincida con la realidad filogenética. Como consecuencia, confiar únicamente en las etiquetas taxonómicas de la base podría resultar en un conjunto *target* incorrectamente definido, lo que afectaría negativamente la precisión de nuestros análisis posteriores.

Para lidiar con esto es necesario seguir un proceso de selección de datos que contemple estas diferencias de clasificación. Para este proyecto, la metodología utilizada y descrita a continuación es una versión simplificada del proceso utilizado en una publicación reciente donde se construyen conjuntos *target* y *neighbor* con las características filogenéticas adecuadas [23].

3.3.1. Recopilación y Preparación de Datos Reales

1. **Selección Preliminar Basada en Taxonomía:** Inicialmente, realizamos una consulta a la base de GenBank utilizando la herramienta `neighbors` [9]. Esto nos proporcionó un conjunto inicial de genomas candidatos para los grupos *target* y *neighbor* basándonos en sus etiquetas taxonómicas.

2. **Refinamiento Filogenético:** Para asegurar que el conjunto *target* no excluya organismos relacionados, buscamos que el mismo constituya un grupo monofilético (que contenga un ancestro común y a todos sus descendientes). La herramienta `Phylonium` [14] nos permite reconstruir el árbol filogenético de nuestra selección preliminar del paso 1 estimando las distancias genéticas entre todas las secuencias. Basándonos en la topología de este árbol, aplicamos una regla de corrección a la selección inicial: si algún genoma inicialmente etiquetado como *neighbor* se encuentra dentro una misma rama que algún genoma target, se lo re-clasifica como target.

Este proceso garantiza que la partición final de los conjuntos *target* y *neighbor* sea coherente con las relaciones evolutivas reales, minimizando el sesgo que podría introducir una clasificación puramente taxonómica.

3.3.2. Diseño y evaluación de primers

Una vez aplicados los algoritmos para conseguir los candidatos a marcadores, el siguiente paso fue diseñar primers para la amplificación por PCR y evaluar su rendimiento. A continuación describimos los puntos más importantes del proceso.

Utilizamos `PRIMER3` [21], una herramienta estándar en bioinformática, para generar pares de primers candidatos a partir de los candidatos a marcadores. `PRIMER3` requiere como entrada que los marcadores estén en un formato específico al que llegamos usando el programa `fa2prim`.

La salida de `PRIMER3` consiste de múltiples pares de primers con un valor de penalidad asociado. La penalidad es un valor que contempla distintos aspectos químicos y biológicos de los primers y un valor muy bajo es un buen indicio de que ese par funcionará de manera adecuada en una reacción de PCR real. En nuestro proceso seleccionamos el par de primers con la menor penalidad encontrada.

Los pares de primers seleccionados luego se someten a una simulación de PCR in silico utilizando la herramienta `scop`, la cual puntúa los primers calculando su sensibilidad y especificidad *in silico*. La entrada pueden ser uno o más primers destinados a una mezcla de reacción. Adicionalmente, toma un conjunto de identificadores de los *target* y una base de datos Blast vinculada a la taxonomía del NCBI, en concreto se utilizó la base `nt_prok` que contiene todos los procariotas indexados en la base de datos de NCBI.

La sensibilidad y especificidad se calculan de la siguiente manera:

$$s_n = \frac{t_p}{t_p + f_n} \tag{3.3}$$

$$s_p = \frac{t_p}{t_p + f_p} \tag{3.4}$$

Donde t_p es el número de verdaderos positivos (amplificaciones esperadas que efectivamente se dieron), f_n el número de falsos negativos (amplificaciones deseadas, no encontradas) y f_P es el número de falsos positivos (amplificaciones no deseadas).

Además, `scop` imprime los verdaderos positivos, los falsos positivos y los falsos negativos, si los hubiera, para una verificación posterior con el programa `cops`, que corrige las puntuaciones de los primers obtenidas por `scop` realizando un análisis filogenético de los genomas involucrados.

Las herramientas `cops`, `scop`, `fa2prim` son parte de un paquete para el diseño y evaluación de primers diseñado por Bernard Haubold llamado `prim` [11].

Capítulo 4

Resultados experimentales

En el presente capítulo se presenta el diseño experimental y los resultados de su ejecución basándonos en los procedimientos y métricas del capítulo 3.

4.1. Evaluación del Consumo de Recursos

Evaluamos la escalabilidad de los algoritmos frente a variaciones en el tamaño y la cantidad de los datos de entrada, manteniendo los parámetros internos de cada algoritmo en sus valores predeterminados. Se realizaron 2 experimentos para cubrir ambos posibles escenarios de crecimiento de los datos de entrada. Para todos los casos medimos el tiempo de ejecución de usuario (contemplando el uso de cómputo paralelo de los algoritmos) y el consumo máximo de memoria RAM utilizando la herramienta de `time` de Unix. Todas las ejecuciones se realizaron en una PC de mesa con las siguientes especificaciones: CPU Intel Core i5-10400F @ 2.90GHz, 16.0 GiB de memoria RAM, y sistema operativo Ubuntu 22.04.5 LTS.

4.1.1. Escalabilidad con respecto al conjunto *neighbor*

En escenarios reales es esperable que el conjunto *neighbor* sea el más numeroso, mientras más genomas incluyamos en este conjunto más confianza podemos tener en los marcadores producidos por las herramientas, puesto a que nos cubrimos de que se puedan llegar a detectar. Si lo llevamos a un extremo teórico, el conjunto *neighbor* podría potencialmente consistir de ejemplares de todos los organismos que no queremos identificar. Por eso es importante evaluar como escalan los algoritmos cuando el conjunto de entrada *neighbor* crece.

En este experimento se configuró un escenario con un único genoma *target* de tamaño fijo (1 Mb, 10^6 nucleótidos) y un único genoma neighbor, cuyo tamaño se incrementó progresivamente desde 1 Mb hasta 20 Mb. El tiempo de ejecución y el consumo de memoria resultantes se presentan en las Figuras 4.1 y 4.2, respectivamente.

En la figura 4.1 se aprecia que, la fase de búsqueda de `GenMap` (genmap-map) es, por un amplio margen, el proceso más lento y que peor escala en todos los escenarios. Aunque su etapa de indexado (genmap-index) es comparable al de `makeFurDb`, el mal rendimiento de `GenMap` es explicable, ya que además de realizar el conteo de los k-meros para todos los genomas (no solo los *target*) también calcula la mapeabilidad lo que implica un costo extra. Por otro lado, la ejecución de `fur` es definitivamente la más rápida, aunque si tenemos en cuenta la suma de los tiempos de construcción y búsqueda, el tiempo es muy similar al de `KEC` que no separa el cálculo en dos etapas.

En cuanto a memoria, en la figura 4.2 se puede ver que el indexado de `GenMap` es más eficiente que `makeFurDb` y escala muy lentamente con respecto al tamaño de la entrada gracias a la compresión que utilizan los índices FM. Pero nuevamente, la etapa de mapeo es con diferencia la más demandante y la que peor escala. `makeFurDb` escala de manera lineal y predecible y la búsqueda de marcadores con `fur` utiliza muy poca memoria incluso para las entradas más grandes. `KEC` presenta un consumo de memoria menos predecible, probablemente resultado del dimensionamiento de las tablas hash que solo almacenan k-meros no repetidos, los cuales aumentan a la par con el tamaño del genoma pero no de manera lineal.

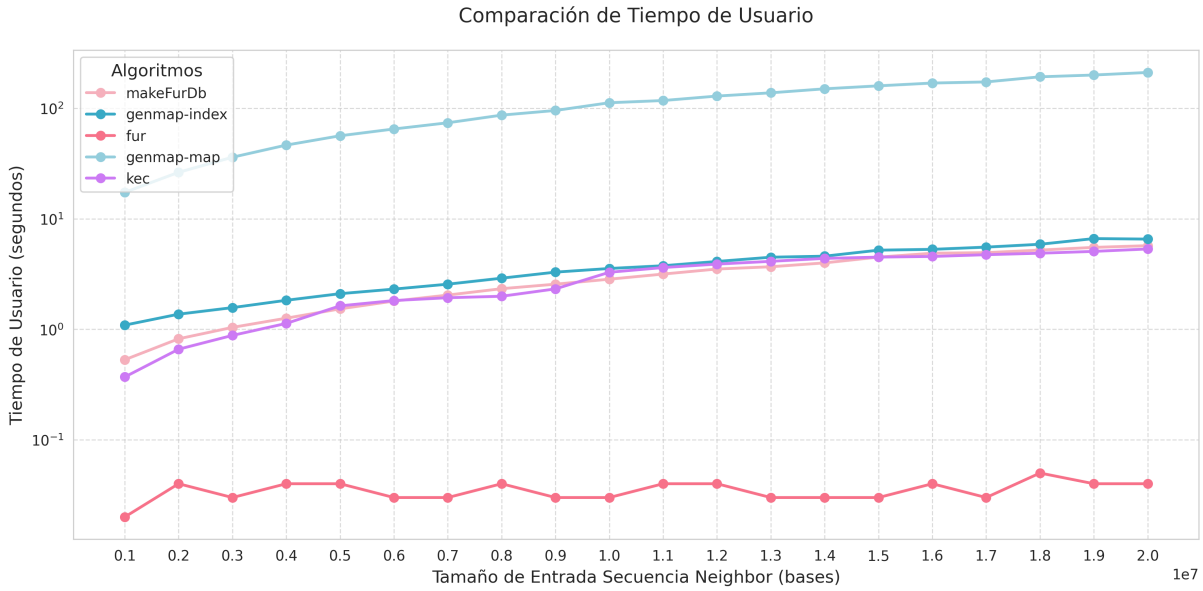

Figura 4.1: Tiempo de ejecución (segundos) en función del tamaño del genoma *neighbor* (Mb) para Experimento 1.

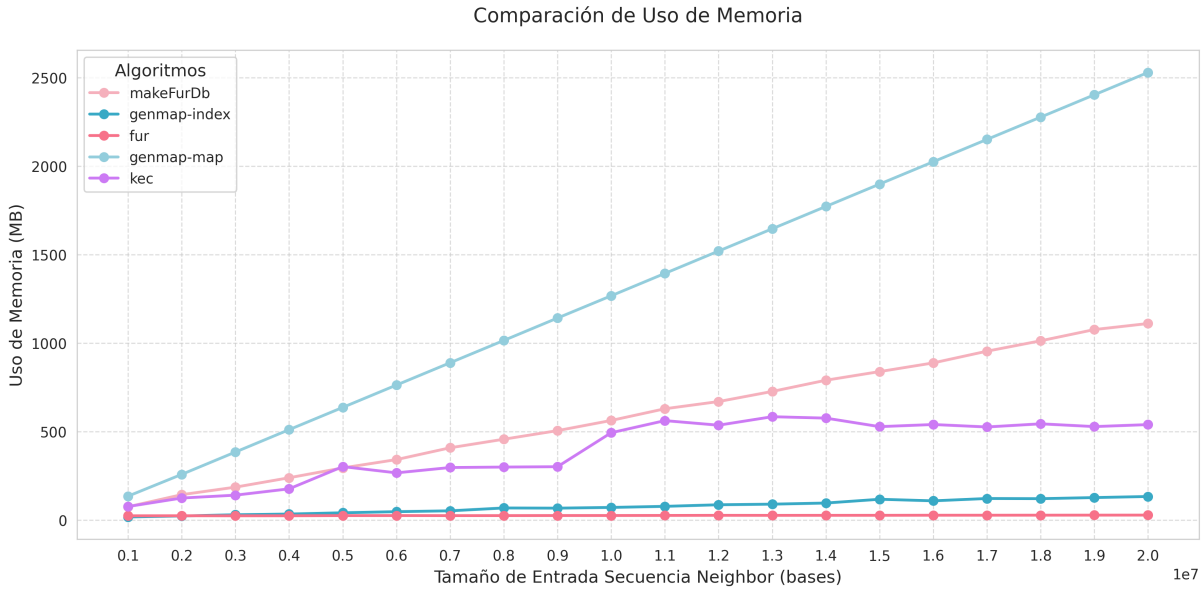

Figura 4.2: Consumo máximo de memoria (GiB) en función del tamaño del genoma *neighbor* (Mb) para Experimento 1.

4.1.2. Escalabilidad con respecto al conjunto *target*

Como se mencionó anteriormente, el crecimiento en el conjunto *neighbor* es quizás el más relevante a tener en cuenta por la naturaleza de la definición de los conjuntos en los problemas. De todas formas, el conjunto *target* puede eventualmente ser de gran tamaño y, por lo tanto, también es relevante identificar cómo se comportan los algoritmos cuando se da el caso.

De forma análoga al Experimento 1, este experimento se diseñó para medir la influencia del tamaño del genoma *target* en el rendimiento. Se mantuvo un único genoma *neighbor* de tamaño fijo (1 Mb), mientras que el tamaño del único genoma *target* se incrementó desde 1 Mb hasta 20 Mb. Las Figuras 4.3 y 4.4 ilustran los resultados obtenidos para tiempo y memoria.

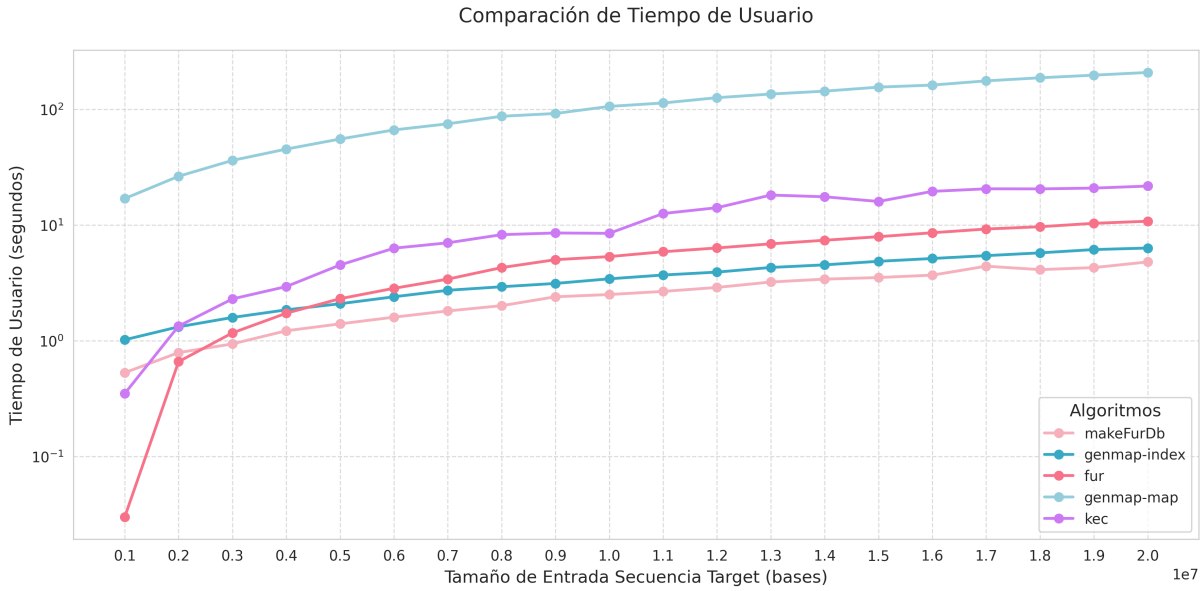

Figura 4.3: Tiempo de ejecución (segundos) en función del tamaño del genoma *target* (Mb) para Experimento 2.

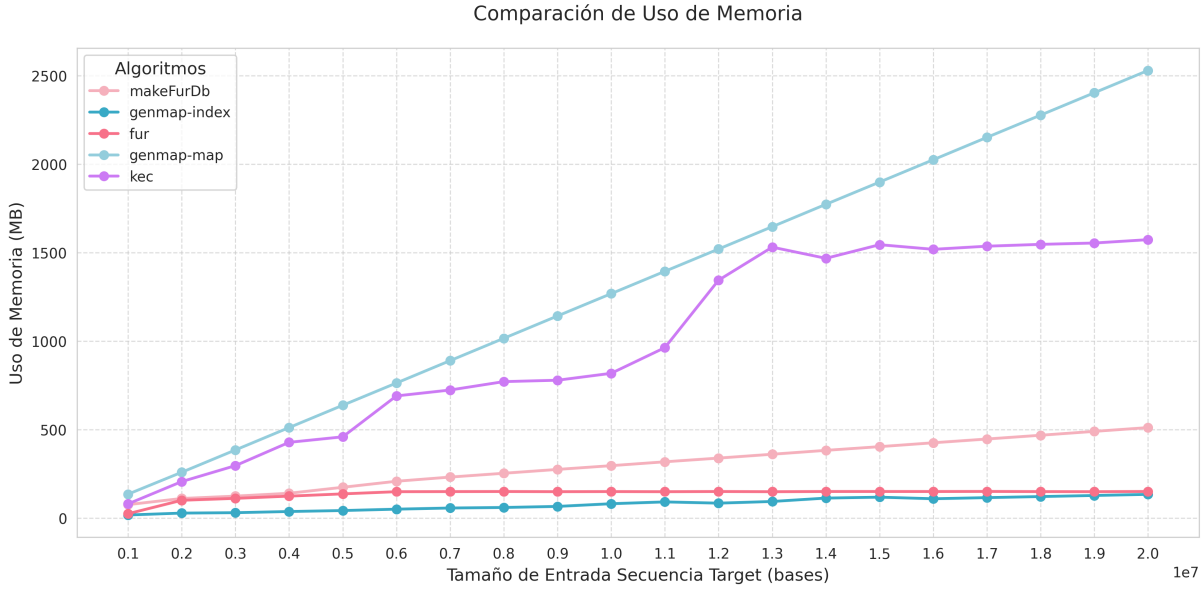

Figura 4.4: Consumo máximo de memoria (GiB) en función del tamaño del genoma *target* (Mb) para Experimento 2.

4.1.3. Conclusiones sobre Escalabilidad

El comportamiento de los algoritmos se corresponde a lo esperado por su diseño. El tiempo de ejecución de `GenMap` y `makeFurDb` muestra una dependencia casi lineal con el tamaño total de los datos, consistente con sus métodos basados en la construcción de índices sobre las secuencias. La velocidad de fur en la fase de búsqueda evidencia un buen uso del índice pre-calculado. Por otro lado, la memoria de `KEC` escala con la cantidad de k-meros únicos, lo que explica su crecimiento moderado pero constante.

4.2. Análisis de la Sensibilidad a los Parámetros

4.2.1. Análisis de la Sensibilidad a los Parámetros en Escenario Reducido

Para una primera evaluación de la calidad de los resultados de cada algoritmo se simularon dos secuencias con un marcador genético único conocido (1 *target* y 1 neighbor), generadas como se describe en la sección 3.1.1.

Sobre estos datos ejecutamos`KEC`, `GenMap` y `fur` con el objetivo de observar cómo varía la salida de cada algoritmo al modificar su parámetro principal: el tamaño del k-mero para `KEC` y `GenMap` (con tolerancia a 3 errores), y el tamaño de la ventana para `fur`. Se evaluaron los 2 métodos definidos en la sección 3.2 para métricas de calidad. Tratándose de un escenario controlado donde la única región única existente es la introducida, se toma el candidato más largo producido por cada ejecución para calcular la métrica.

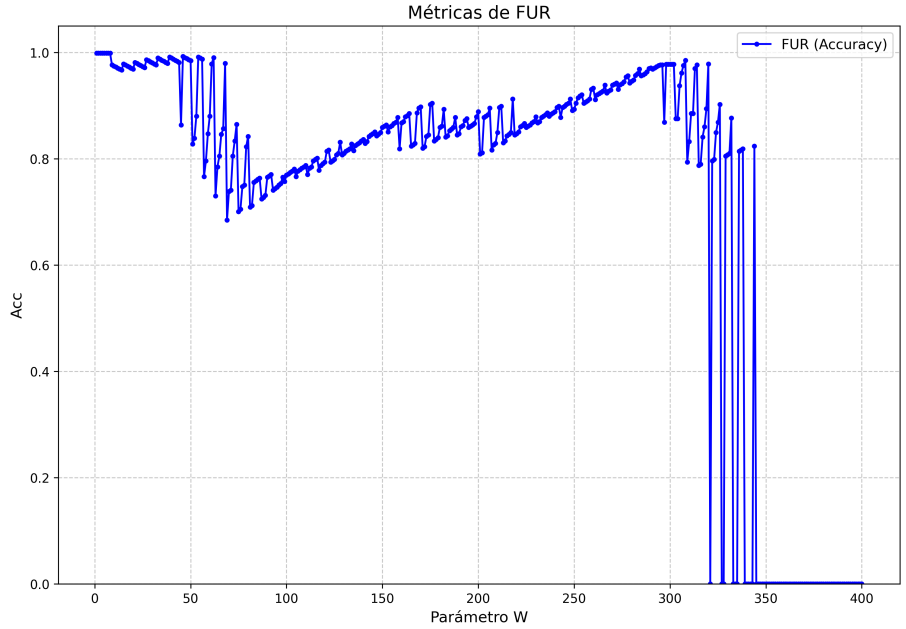

Figura 4.5: Comparación de las métricas de calidad para `fur` al variar su parámetro principal (W) en el escenario de marcador simple.

En la figura 4.5 se aprecia una notable oscilación en el valor Acc de `fur` al variar el tamaño de la ventana de análisis. Si bien existe una tendencia a que los resultados empeoran para valores altos y a partir $k = 350$ el algoritmo no produce resultados, se muestra que fur es altamente sensible al tamaño de la ventana, y esto dificulta definir una heurística clara para la selección del parámetro.

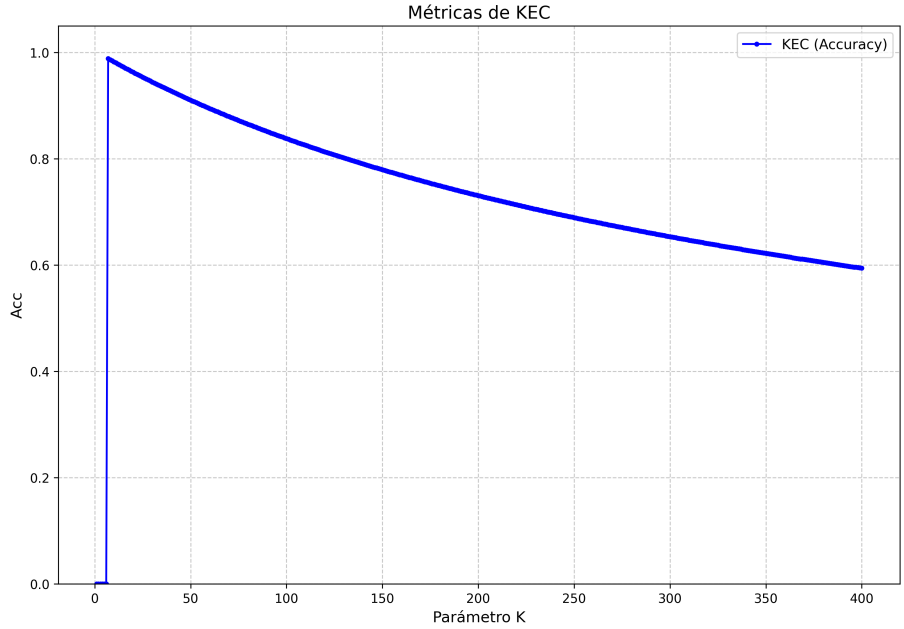

Figura 4.6: Métrica de exactitud (*Acc*) para `KEC` al variar su parámetro principal (K) en el escenario de marcador simple.

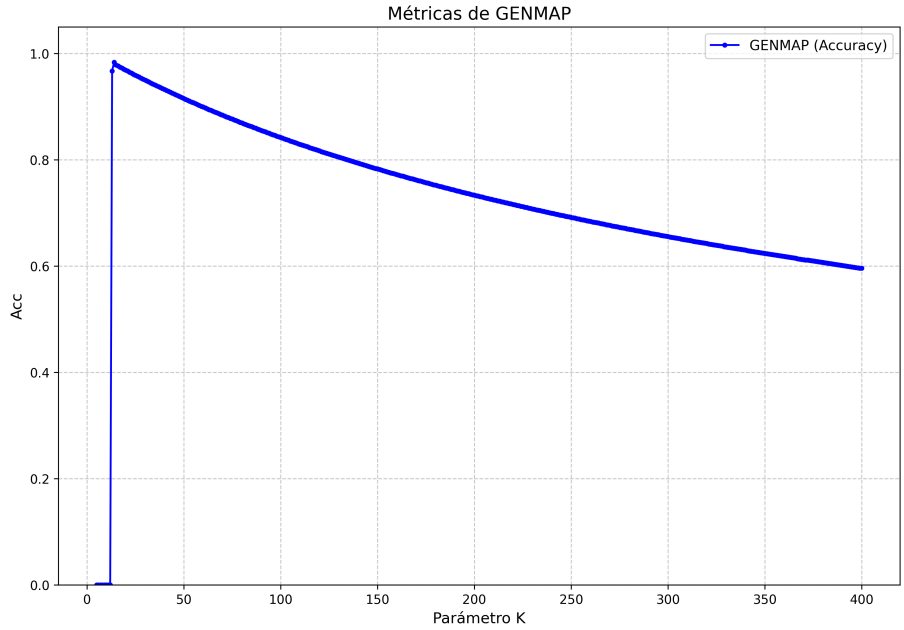

Figura 4.7: Métrica de exactitud (*Acc*) para `GenMap` al variar su parámetro principal (K) en el escenario de marcador simple.

A diferencia de `fur`, tanto `GenMap` (ver figura 4.7) como `KEC` (ver figura 4.6) muestran un comportamiento mucho más predecible: ambos algoritmos no producen resultados para valores de K muy bajos. Luego, de manera abrupta, los resultados mejoran hasta alcanzar un valor óptimo para este conjunto de datos (alrededor $K = 15$). Luego decaen cuando el valor de K aumenta.

Notemos que en este escenario los valores de `KEC` y `GenMap` son notablemente similares. Resulta que en escenarios con un solo *target* ambos programas funcionan como implementaciones distintas de una misma metodología de búsqueda, con la salvedad de que `GenMap` tiene el potencial de realizar búsquedas aproximadas y no exactas de los k-meros. Si la tolerancia a errores de `GenMap` se configura en 0, los resultados son efectivamente los mismos.

4.2.2. Análisis de la Sensibilidad a los Parámetros en Escenario Numeroso

Se procedió a replicar y extender el experimento realizado en [16] para tener una mejor visión de cómo se comportan los algoritmos al variar los parámetros principales. Se generaron distintos escenarios con 10 targets y 10 neighbors de 5 Mb, a los que se les introdujo de manera independiente cuatro marcadores de longitudes variables (M1: 200 bases, M2: 400 bases, M3: 800 bases y M4: 1600 bases) generados por un evento de eliminación completa en el genoma *neighbor*.

Se realizaron 50 ejecuciones sobre datos distintos pero generados con la misma configuración (variando solo la semilla de stan). Para cada set de datos se ejecutaron los algoritmos variando sistemáticamente su parámetro principal (tamaño de la ventana para `fur` y longitud de k-mero para `KEC` y `GenMap`), calculando utilizando el coeficiente de la ecuación 3.2 como métrica para evaluar la calidad de los marcadores producidos.

Tratándose de un escenario controlado donde la única región única existente es la introducida, se toma el candidato más largo producido por cada ejecución para calcular la métrica.

Resultados `fur`

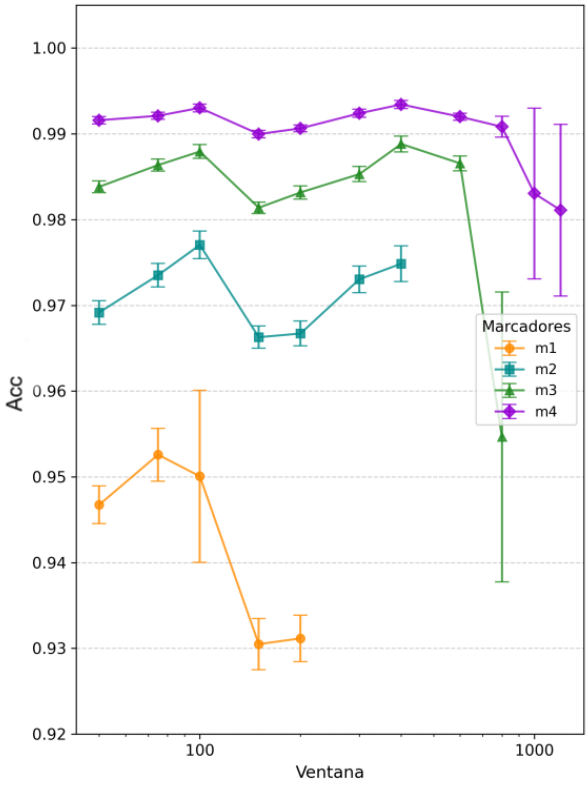

Figura 4.8: Comparación de la métrica *Acc* de `fur` en función del tamaño de ventana (w) para marcadores de distinta longitud (m1-m4) en el escenario de deleción. Las barras de error representan la desviación estándar de 50 ejecuciones.

En el caso de `fur`, la figura 4.8 muestra que la precisión es muy alta y con poca variabilidad. Los resultados tienden a variar más cuando la ventana es grande, y no se obtienen resultados cuando esta se aproxima al tamaño del marcador existente.

Por otro lado, `KEC` exhibe un comportamiento predecible, como se observa en la figura 4.9. La exactitud es casi perfecta para un rango de valores de K, con una degradación lineal a medida que K aumenta. El rango en el que `KEC` produce buenos resultados es más reducido que el de `fur`, pero su predictibilidad permite consolidar una heurística de uso muy práctica: ejecutar `KEC` con un K inicial bajo e incrementarlo hasta que los resultados se estabilicen, identificando así el parámetro óptimo de forma sistemática.

En la figura 4.10 vemos que `GenMap` fue sin duda el algoritmo que obtuvo peores resultados para estos escenarios. Si bien el comportamiento es análogo al de `KEC`, decae de manera predecible según aumenta K, la variación de resultados y la precisión de las predicciones fue de peor calidad en general. Los

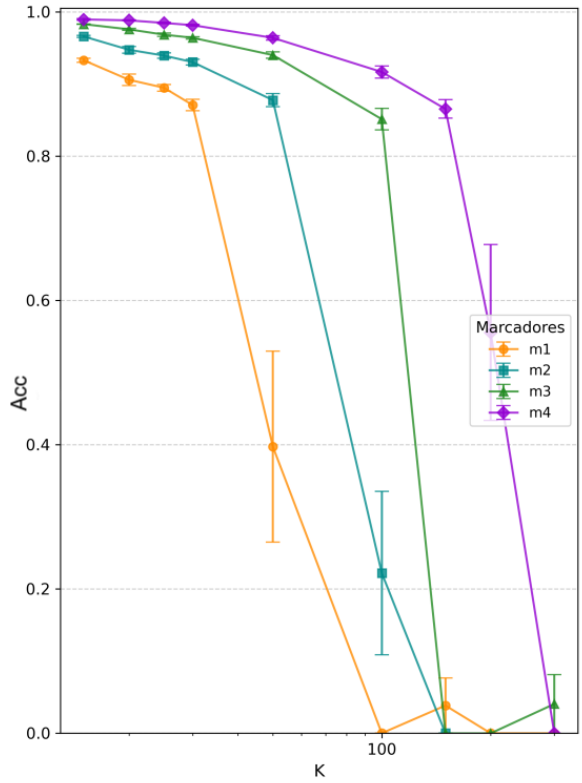

Figura 4.9: Comparación de la métrica *Acc* de `KEC` en función del tamaño del k-mero (K) para marcadores de distinta longitud (m1-m4) en el escenario de deleción. Las barras de error representan la desviación estándar de 50 ejecuciones.

malos resultados se deben en gran medida a que `GenMap` tiende a fragmentar los marcadores devolviendo porciones pequeñas de los mismos (la métrica de la ecuación 3.2.1 penaliza mucho esto), y es posible que aumentando la tolerancia a errores se puedan mejorar los resultados, pero esto aumenta significativamente el tiempo de búsqueda y, por lo tanto, dicha exploración fue omitida. Este comportamiento se puede ver con más claridad en los resultados de la sección 4.4.

4.3. Conclusiones sobre el análisis de sensibilidad de parámetros

Si bien `fur` mostró cierta irregularidad en la calidad de los resultados en el escenario simple, su comportamiento se regularizó al introducir más material a ambos conjuntos. El algoritmo es muy robusto y genera buenos resultados para un amplio rango de valores de su parámetro principal.

La calidad de los resultados de `KEC` está altamente relacionada al valor que se elige para k, y se observa que los mejores resultados se obtienen para valores relativamente bajos, existiendo un límite inferior para el cual no es capaz de producir regiones contiguas.

`GenMap` mostró un comportamiento similar al de `KEC` pero con mayor inestabilidad en los resultados. Achacamos esto a que exige a los k-meros estar presentes en todos los genomas target, volviéndolo muy sensible a mutaciones. En consecuencia, `GenMap` a menudo no logra superponer k-meros y realiza predicciones poco abarcativas.

4.4. Evaluación en Escenarios de Simulación Realista

Las pruebas sobre escenarios simples realizadas exponen algunas de las diferencias claves entre los 3 métodos, pero no son lo suficientemente realistas para asumir que ese comportamiento se extrapolará de forma directa al análisis de genomas reales. Con el objetivo de someter a los algoritmos a una prueba que refleje los desafíos del análisis genómico real, se construyeron 2 escenarios siguiendo las siguientes directrices que pretenden imitar la complejidad biológica real de un análisis sobre genomas de bacterias:

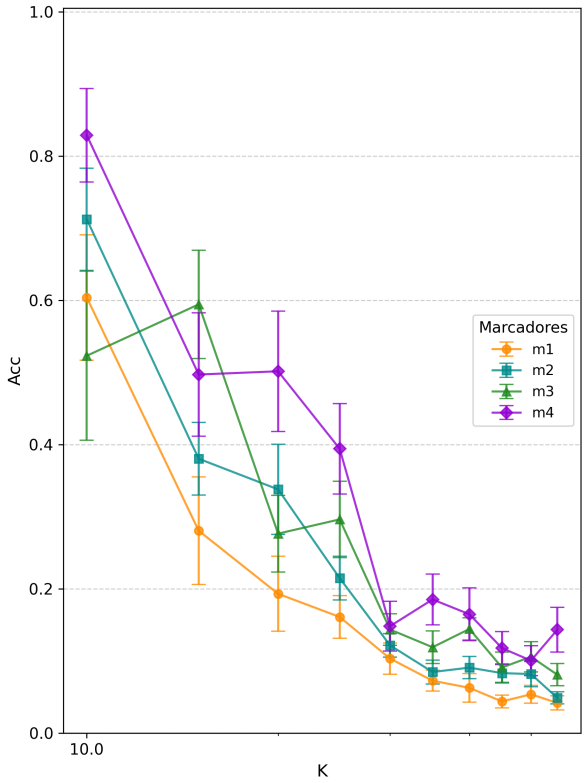

Figura 4.10: Comparación de la métrica *Acc* de `GenMap` sin tolerancia a errores ($E = 0$) en función del tamaño del k-mero (K) para marcadores de distinta longitud (m1-m4) en el escenario de deleción. Las barras de error representan la desviación estándar de 50 ejecuciones.

- **Composición de la Población:** En ambos escenarios se configuró un conjunto *target* de 15 genomas y un conjunto *neighbor* más extenso de 30 genomas. Esta asimetría es representativa de los estudios de diagnóstico, donde el conjunto *target* consiste de secuencias de una especie de interés, debe ser contrastado con un fondo filogenético amplio y diverso para garantizar la especificidad de los marcadores.
- **Tamaño de las secuencias:** Cada genoma simulado se estableció en 4 millones de pares de bases (4 Mb), un tamaño característico de genomas procariotas como el de *Escherichia coli*.
- **Diversidad Genética de Fondo:** Se aplicó una tasa de mutación de fondo moderada y realista ($\theta = 0{,}01$) en todos los genomas para simular el polimorfismo natural que se esperaría entre cepas relacionadas.
- **Marcadores Múltiples y de Tamaño Variable:** Se introdujeron dos regiones marcadoras en los genomas *target*: una de 500 pb (M1) y otra más larga de 4,000 pb (M2). Esta configuración evalúa la capacidad de los algoritmos para detectar regiones de distinta magnitud, reflejando que las regiones únicas de interés pueden variar desde porciones pequeñas como un solo gen hasta regiones más grandes como por ejemplo los operones (agrupación de varios genes común en bacterias) o islas genómicas que a menudo involucran múltiples genes.
- **Mecanismo de introducción de de marcadores:** Compartiendo las características anteriores,los dos escenarios difieren fundamentalmente en el mecanismo evolutivo que da origen a los marcadores, en el primer caso se introducen eliminando una región homologa en los genomas neighbors y en el segundo introduciendo muchas mutaciones sobre una región específica.

Sobre estos escenarios realizamos un análisis exploratorio con los 3 algoritmos, iniciando con sus parámetros por defecto y ajustando hasta lograr identificar con niveles altos de especificidad los 2 marcadores introducidos en cada set de datos.

El escenario 1 presentado en la sección 4.4.1 simula la creación de regiones únicas mediante la eliminación completa de los segmentos correspondientes en los genomas del conjunto *neighbor*. Representa el caso

más directo y conceptualmente simple para la identificación de marcadores. Mientras que el escenario 2 presentado en la sección 4.4.2 representa una prueba de estrés más sutil y biológicamente compleja. En lugar de una eliminación, las regiones homólogas en los genomas *neighbor* fueron sometidas a una tasa de mutación extremadamente alta ($\theta = 0{,}5$), 50 veces superior a la del resto del genoma. Esto simula un proceso de divergencia evolutiva acelerada, donde la región se vuelve única no por su ausencia, sino porque la homología se degrada por una acumulación masiva de mutaciones. Como consecuencia, estas regiones están menos definidas y son más desafiantes de identificar.

4.4.1. Resultados escenario 1: Marcadores Generados por eliminación

Tabla 4.1: Resultados de los algoritmos en el Escenario 1: Marcadores Generados por Eliminación.

Algoritmo	***Región identificada***	***Longitud (pb)***	***Sn***	***Sp***	***C***
`fur` w = 80	M1	495	1.0	1.0	1.0
	M2	4024	1.0	0.98	0.99
`KEC` k = 12 (filtrados por homolgia)	M1	536	1.00	1.00	1.00
	M2	4022	1.00	1.00	1.00
`GenMap` K=30, E=2	M1	548	1.00	1.00	1.00
	M2	4053	1.00	1.00	1.00

Como se puede ver en la tabla 4.1 los 3 algoritmos lograron identificar con niveles muy altos de sensibilidad y especificidad las 2 regiones únicas introducidas. La tabla 4.2 presenta los detalles de consumo de recursos sobre este escenario.

Cabe destacar que en el caso de `KEC` el resultado inicial fue muy numeroso, generando 30 marcadores que correspondían a las mismas dos regiones únicas, multiplicadas 15 veces para cada uno de los genomas *target*. Este comportamiento es el esperado ya que el algoritmo no realiza un paso de intersección entre targets.

Para evitar presentar resultados redundantes eliminamos usando `BLAST`todos los marcadores que tuvieran un homologo entre el resto de candidatos.

4.4.2. Resultados escenario 2: Marcadores Generados por Divergencia Evolutiva Acelerada

`fur` no logró identificar ninguna región candidata con sus valores por defecto ni en diversas ejecuciones, variando el tamaño de la ventana de búsqueda. Este resultado negativo es ilustrativo de la naturaleza conservadora del algoritmo. La alta densidad de mutaciones en los genomas *neighbor* no eliminó por completo la señal de homología. Por lo tanto, `fur`, siguiendo su lógica diseñada para minimizar falsos positivos, interpretó correctamente que existía una homología remanente y descartó las regiones por no ser suficientemente "únicas" desde una perspectiva de alineamiento.

`KEC` con sus parámetros por defecto ($k = 12$) tampoco identificó la región M2 como un único marcador, sino que la identificó de manera fragmentada como se muestra en la tabla 4.4.2, estas porciones muestran valores perfectos de especificidad y sensibilidad. Incluso mejores que cuando eventualmente logra convergir a solo dos candidatos. Esto puede deberse a la acumulación de mutaciones en distintas partes de la región M2. La metodología de `KEC` se basa en la presencia o ausencia de k-meros de forma exacta y en este escenario, la alta tasa de mutación en los genomas *neighbor* destruyó eficazmente la identidad de los

Programa	***Configuración***	***Tiempo (s)***	***Memoria(GB)***	***Uso de CPU***
makeFurDb	-	19.12	3.4	497 %
fur	default	2.87	0.1	120 %
KEC	k = 12	30.4	1.0	103 %
GenMap index	-	64.0	1.1	112 %
GenMap map	k=30, e=2	39.5	2.2	288 %

Tabla 4.2: Recursos utilizados por los distintos en sobre los datos del escenario 1.

Tabla 4.3: Resultados de los algoritmos en el Escenario 1: Marcadores Generados por Eliminación.

Algoritmo	***Región identificada***	***Longitud (pb)***	***Sn***	***Sp***	***C***
`fur`	No se logró identificar marcadores variando el tamaño de ventana				
`KEC` (k = 12)	M1	513	1.00	1.00	1.00
	M2	244	1.00	1.00	1.00
	M2	204	1.00	1.00	1.00
	M2	470	1.00	1.00	1.00
	M2	1072	1.00	1.00	1.00
	M2	1225	1.00	1.00	1.00
	M2	472	1.00	1.00	1.00
`KEC` (k = 20)	M1	529	1.00	1.00	1.00
	M2	4023	1.00	0.62	0.66
`GenMap` K=30, E=2	M1	524	1.00	1.00	1.00
	M2	1059	0.79	0.82	1.00
	M2	2964	0.99	0.75	0.79

Programa	***Configuracion***	***Tiempo (s)***	***Memoria(GB)***	***Uso de CPU***
KEC	k = 20	30.6	1.1	104 %
GenMap index	-	67.1	1.2	112 %
GenMap map	k=30, e=2	40.1	2.2	302 %

Tabla 4.4: Recursos utilizados por los distintos en sobre los datos del escenario 2.

k-meros originales en esas regiones. Desde la perspectiva de `KEC`, que busca coincidencias exactas, los k-meros de las regiones *target* estaban completamente ausentes en el conjunto *neighbor* lo que permitió una identificación más clara.

`GenMap` también dividió el marcador más grande en 2 candidatos que lo abarcan casi en su totalidad. Sin embargo, dado el alto nivel de especificidad y la proximidad de las regiones devueltas, detuvimos el análisis para esa combinación de parámetros.

La tabla 4.4 presenta los detalles de consumo de recursos sobre este escenario.

4.4.3. Conclusiones de la Simulación Realista

La comparación de estos dos escenarios revela una visión crítica sobre las fortalezas y debilidades de las diferentes filosofías algorítmicas. Mientras que en un caso de eliminación las 3 herramientas funcionan a la perfección (siendo fur la que brinda un resultado más directo y sin necesidad de posprocesamiento), el escenario con marcadores por alta tasa de mutaciones expone un "punto ciego"fundamental en el enfoque de `fur`. Su rigor biológico puede volverse en su contra en escenarios donde las regiones son sutiles, llevándolo a pasar por alto regiones que son si cumplen el criterio de unicidad y son potencialmente útiles para el diseño de primers de PCR.

Por el contrario, el enfoque más simple de `KEC`, basado en k-meros exactos, demuestra ser una herramienta superior en este contexto específico y biológicamente plausible. Este hallazgo nos proporciona una justificación para invertir esfuerzos en mejorar la arquitectura y escalabilidad de los algoritmos basados en k-meros.

4.5. Resultados experimentales sobre datos reales

En este capítulo, presentamos la evaluación de los algoritmos de búsqueda de regiones únicas utilizando datos genómicos reales. Nuestro objetivo principal fue determinar la eficacia de las herramientas seleccionadas en un contexto práctico, específicamente para la identificación de marcadores genéticos útiles en el diseño de cebadores para pruebas de PCR. El proceso se centró en la especie *Legionella pneumophila*, la cual es de interés por ser potencialmente patógena [4] [22].

Si bien `GenMap` mostró ser capaz de encontrar marcadores con buenas cualidades en datos simulados, su uso elevado de recursos limita su uso en hardware estándar. Bajo el criterio de evaluar las herramientas

para su aplicación práctica en el diseño de primers decidimos excluir a `GenMap` de esta etapa del análisis, ya que su aplicación en datos reales fue problemática en el hardware disponible para el proyecto.

4.5.1. Resultados

Seleccionamos en concreto la cepa *Legionella pneumophila subsp. pneumophila str. Philadelphia 1* como objetivo. El dataset final luego del proceso de selección consistió de 159 neighbors y 24 targets.

Los resultados del diseño y la evaluación de los primers se presentan en las tablas 4.5 y 4.6, mostrando los pares de primers diseñados, su penalidad, y los valores de sensibilidad según la ecuación 3.3 y especificidad según la ecuación 3.4 obtenidos.

Forward	***Reverse***	***Penalidad***	***Sen.***	***Spec.***
GGAGCAAATTTTGGCGATTT	TCTCGTAACATCATTCCCCA	0.269	1	1

Tabla 4.5: Resultados de la evaluación de primers para *L. pneumophila* utilizando `fur`.

La salida de `fur` produjo alrededor de 100 candidatos a marcadores, mientras la salida original de `KEC` produjo más de 8000 candidatos con longitud superior a 200 bases. No es práctico realizar el desarrollo de primers sobre este gran volumen de datos (pues el tiempo de desarrollo de primers es lento). Además, no todos ellos son buenos marcadores de diagnóstico debido al funcionamiento de `KEC`, ya que muchos de estos candidatos solo están presentes en alguno de los targets. En vista de este problema, desarrollamos y utilizamos un script de refinamiento (o más bien ordenamiento) de marcadores mediante el uso de `BLAST`, construyendo una base de datos de consulta y buscando la aparición de homólogos tanto en targets como en neighbors para luego dar una "recomendación" (Confianza Alta, Media o Baja) basada en su cumplimiento del doble criterio de unicidad. Este enfoque cuantitativo, además, ayuda a tomar decisiones informadas incluso cuando no se encuentran marcadores "perfectos". El script está disponible en el repositorio de herramientas del proyecto bajo el nombre de *refine_ markers.py*.

Forward	***Reverse***	***Penalidad***	***Sen.***	***Spec.***
TTTATCTGGTGATGCAAGCC	GGGTTGAAATTCTGAACCGA	0.131	1	1

Tabla 4.6: Resultados de la evaluación de primers para *L. pneumophila* utilizando `KEC`.

4.5.2. Conclusiones sobre experimentación en datos reales

Se lograron resultados ideales en el dataset con ambas herramientas. `fur` sin embargo, se distingue por la facilidad de su aplicación, no requiriendo ningún tipo de postprocesamiento a sus resultados para proceder al diseño de primers de manera rápida, ya que todos los candidatos a marcadores brindados eran de una gran especificidad y sensibilidad. Por otro lado, el experimento dejó en evidencia la necesidad de interpretar los resultados obtenidos con `KEC` en un caso de uso real.

Capítulo 5

Implementación de mejoras de KEC

En una publicación reciente, se presenta una comparación de algoritmos para identificación de regiones únicas en un contexto más general, contemplando también el problema de buscar regiones no repetidas dentro de un mismo genoma[23]. En dicha publicación se excluyó a KEC, argumentando que es un algoritmo que falla en producir marcadores específicos.

Consideramos que esta exclusión es precipitada, por distintos motivos. Por un lado, KEC no fue desarrollada como una herramienta específica para el diseño de marcadores de alta especificidad, sino como un método rápido y sencillo a ser utilizado como parte de un análisis más completo. Por otro lado, la experimentación realizada para este proyecto sugiere que, en ciertos contextos y con la elección adecuada de parámetros, KEC puede llegar a brindar marcadores aptos cuando fur falla. Además, KEC tampoco implementa un refinamiento final como el caso de fur, proceso que mejora la especificidad y es independiente a la forma en que se computan los candidatos a marcadores. En la experimentación con datos reales, fue posible mitigar esto con el uso de un script que realice un refinamiento similar y se tuvo éxito para identificar marcadores igual de específicos que los de fur. En ese aspecto, el proceso de exclusión de k-meros implementado por KEC constituye una base sólida para generar candidatos, que compite con la heurística de fur y su salida puede someterse a un paso adicional de refinamiento, para obtener los beneficios de especificidad si es que estos existieran.

Más aún, el hecho de que KEC no verifica la homología entre todos los targets para los candidatos devueltos, lleva a que la salida del algoritmo suela ser muy numerosa. Aunque incómodo en el flujo de trabajo actual, esto puede representar una ventaja para realizar análisis en conjuntos de datos construidos sin rigor. Como vimos en la sección 3.3.1, la elección de los conjuntos targets y neighbors es un proceso desafiante en el que se pueden cometer errores. Identificar erróneamente un genoma *target* en el caso de fur y GenMap puede llevar a que no se encuentren candidatos a marcadores, aunque el resto de los genomas sean efectivamente parte de un grupo de alta similitud genética y que, por lo tanto, tengan regiones compartidas. Esta característica de KEC lo vuelve más tolerante a una mala elección de los conjuntos. Para estos casos, puede ser preferible contar con muchos candidatos que luego puedan ordenarse según su especificidad en los conjuntos utilizados sin imponer esa restricción a la hora de calcularlos.

Dicho esto, reconocemos que la principal debilidad de KEC es que no realiza su búsqueda en 2 fases como lo hacen fur o GenMap; en su lugar, crea y utiliza dos Hash Tables que se construyen para un valor de K específico. En la práctica, experimentación indica que a menudo es necesario probar distintos valores de K para obtener resultados óptimos. En ese aspecto, KEC es poco eficiente, pues no guarda ninguna información que pueda ser reutilizada de ejecuciones anteriores.

Si bien KEC es muy rápido, el algoritmo no explota la posibilidad de realizar búsquedas simultáneas en paralelo. Por lo que existe posibilidad de aumentar aún más su rendimiento en la práctica.

En este contexto, el presente capítulo proponemos y describimos el desarrollo de una nueva herramienta que, utilizando el principio algorítmico planteado por KEC para seleccionar candidatos, permita la construcción de un único índice comprensivo a partir de las secuencias *neighbor*. Dicho índice nos permite consultarlo para k-meros de diversas longitudes (K) sin necesidad de una reconstrucción completa para cada nuevo valor de K. En esta nueva versión, reconoceremos la necesidad de evaluar varios valores de K y abordaremos el problema directamente para un rango de valores de K, explotando el índice y los resultados de búsquedas anteriores para acelerar el proceso. Además, introducimos la posibilidad de realizar búsquedas en paralelo para acelerar el tiempo real de ejecución para el usuario final.

5.1. Descripción de la herramienta `KEC-FM`

El enfoque que planteamos es:

1. **Construcción de un Índice Global:** En la primera fase, se construye un único índice global a partir de la concatenación de las secuencias neighbors. Este índice se construye una sola vez y es independiente de cualquier valor de K específico.

2. **Consulta K-flexible del Índice:** En la segunda fase, para un valor de K dado (o un rango de valores K), se consulta este índice global para realizar la etapa de referencia cruzada. La lógica de comparación de k-meros *target* contra k-meros *neighbor* se mantiene, pero en lugar de búsquedas en tablas hash, se realizan consultas al índice global. El indice ademas permite una exploración más eficiente de diferentes longitudes de k-meros mediante el almacenamiento inteligente de búsquedas anteriores.

5.1.1. Elección de estructura de datos y librerías

Considerando los requisitos de nuestra modificación, especialmente la necesidad de manejar grandes volúmenes de datos y la flexibilidad para variar K, los Arreglos de Sufijos con LCP y los Índices FM emergen como las alternativas más viables para esta propuesta. Para este proyecto decidimos optar por los índices FM por dos motivos principales: en primer lugar, el uso de compresión en los índices FM brinda una base sólida para aplicar el algoritmo en grandes volúmenes de datos y, en segundo lugar, la posibilidad de realizar búsquedas en ambas direcciones del índice, lo que le da potencial a la herramienta para profundizar en estrategias de búsqueda más inteligentes a futuro.

Implementamos `KEC-FM` en C utilizando la librería `AvxWindowFmIndex` [2] para la creación y consulta del índice FM. Esta librería brinda una implementación de un algoritmo de construcción y búsqueda optimizada para nucleótidos y con soporte para búsquedas en paralelo que demostró ser entre 2 y 4 veces más rápida para la búsqueda de k-meros que la librería `SeqAn` [18], utilizada por `GenMap` y otras herramientas del estado del arte.

El flujo de trabajo de `KEC-FM` se divide en dos programas ejecutables:

1. `kecfm-index`**: Construcción del Índice de Secuencias Vecinas.** Este programa es responsable de la primera fase. Toma como entrada un directorio que contiene los archivos FASTA correspondientes a las secuencias *neighbors* . Procede a leer y concatenar todas estas secuencias en una única hebra de texto. Sobre esta secuencia combinada, construye un índice FM utilizando la interfaz de `AvxWindowFmIndex`. El resultado es un archivo de índice persistente que representa eficientemente el conjunto de todas las secuencias vecinas.

2. `kecfm-find`**: Búsqueda de Regiones Únicas en Secuencias Objetivo.** Este segundo programa implementa la fase de referencia cruzada. Requiere el archivo de índice FM (generado por `kecfm-index`), un directorio con los archivos FASTA de las secuencias target, y el valor K para la longitud de los k-meros. Para cada secuencia target:

    - Se procesa la secuencia por fragmentos para manejar eficientemente archivos grandes.
    - Se generan k-meros de longitud K a partir de la secuencia target.
    - Cada k-mero *target* se busca en el índice FM de las secuencias *neighbor* para determinar su presencia en dicho conjunto.
    - Los k-meros con una frecuencia de cero en el índice se consideran únicos.
    - Finalmente, se fusionan los k-meros adyacentes, para formar regiones únicas más largas y consolidadas .

El buen rendimiento de `KEC-FM` está basado en gran medida en las optimizaciones que implementa la librería. Donde se destaca el uso de una tabla de búsqueda pre-calculada de k-meros que complementa al índice. Esta tabla permite determinar, en un único paso, el rango inicial de los últimos r símbolos de la consulta, eliminando así los primeros r pasos del algoritmo de búsqueda. Durante la construcción del índice, se le define al constructor un valor r, que utiliza para poblar la tabla. Esta técnica acelera especialmente la operación `count()` del índice, y resulta particularmente útil para `KEC-FM` que se basa en buscar k-meros relativamente cortos.

El anexo A.5 presenta un reporte técnico en detalle de la herramienta, que se complementa con la documentación disponible en el repositorio que la contiene.

5.2. Comparación de performance

Repetimos el análisis de performance de la sección 4.1 incluyendo la nueva herramienta para la comparación directa con su antecesor `KEC` y con `fur` por tratarse del algoritmo en 2 fases con mejor performance de los analizados. Todas las ejecuciones se realizaron con los parámetros por defecto, en el caso de `KEC-FM` definimos el valor por defecto de $k = 12$.

5.2.1. Escalabilidad con respecto al conjunto neighbor

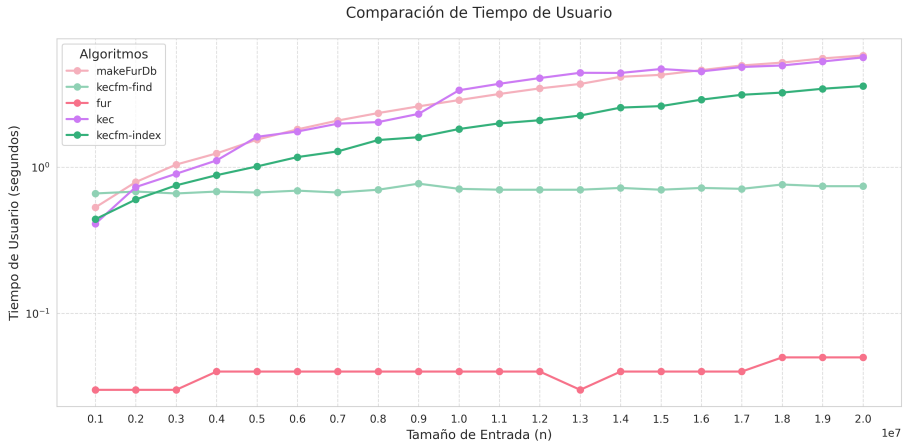

Figura 5.1: Tiempo de ejecución en función de tamaño de la secuencia neighbor

En la figura 5.1 se puede ver como la etapa de búsqueda de `KEC-FM` es más rápido que su antecesor a partir de las 7Mb de secuencia neighbor, pero es considerablemente más lento que `fur`. El tiempo de ejecución de la creación del índice crece linealmente conforme aumenta el tamaño del genoma, pero a una tasa menor que el proceso de indexado de `makeFurDb`. Además, la etapa find de `KEC-FM` presenta un tiempo de ejecución constante. Este es el comportamiento esperado, ya que tratándose de un índice FM las consultas deberían ser independientes del texto indexado.

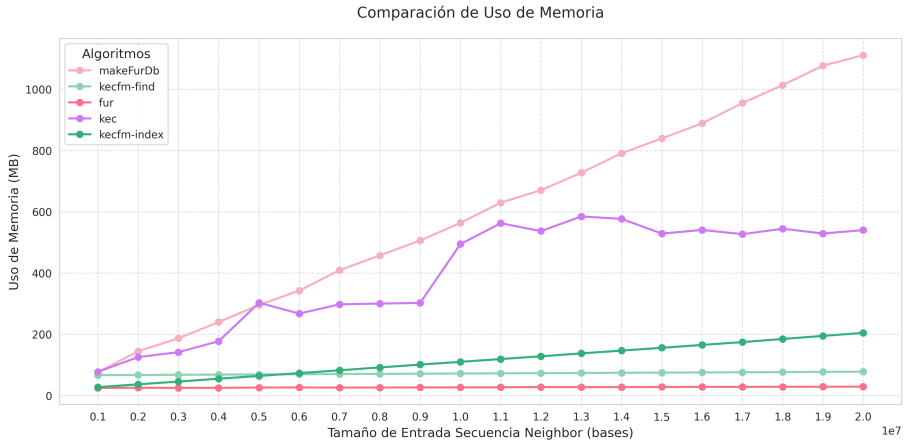

Figura 5.2: Uso máximo de memoria en función de tamaño de la secuencia *neighbor*

En cuanto al uso de memoria, tanto `KEC-FM-find` cómo `KEC-FM-index` tienen una huella mucho menor que la de `KEC` o `makeFurDb`. En este resultado influye tanto el uso de compresión del índice, como el manejo de memoria inteligente usando memory mapping y procesamiento por batches usados por ambos programas de `KEC-FM`.

5.2.2. Escalabilidad con respecto al conjunto target

La figura 5.3 muestra una de las debilidades de nuestra implementación de `KEC-FM`, y es que su etapa de búsqueda no escala bien cuando aumenta el tamaño de los genomas target. Esto se acentúa en este

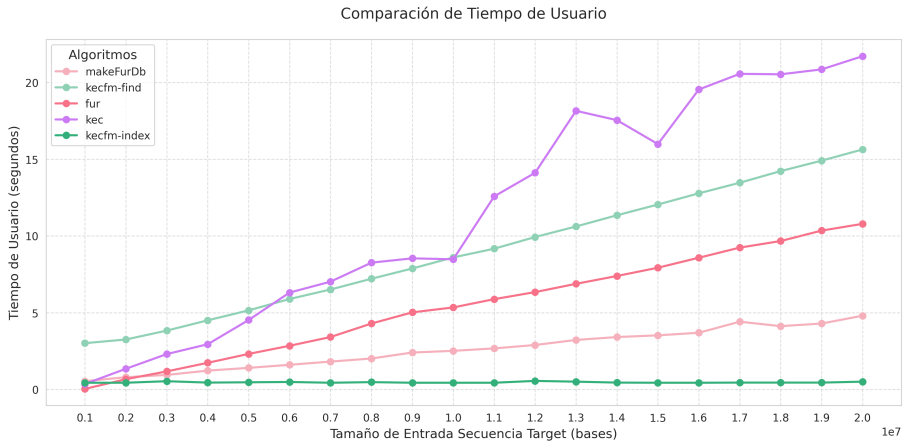

Figura 5.3: Tiempo de ejecución en función de tamaño de la secuencia target

escenario donde además hay pocos neighbors, puesto que no se puede explotar la tabla de búsqueda rápida y se deben realizar todas las búsquedas en el índice. Aun así, `KEC-FM-find` escala de manera predecible y es más rápida que `KEC`.

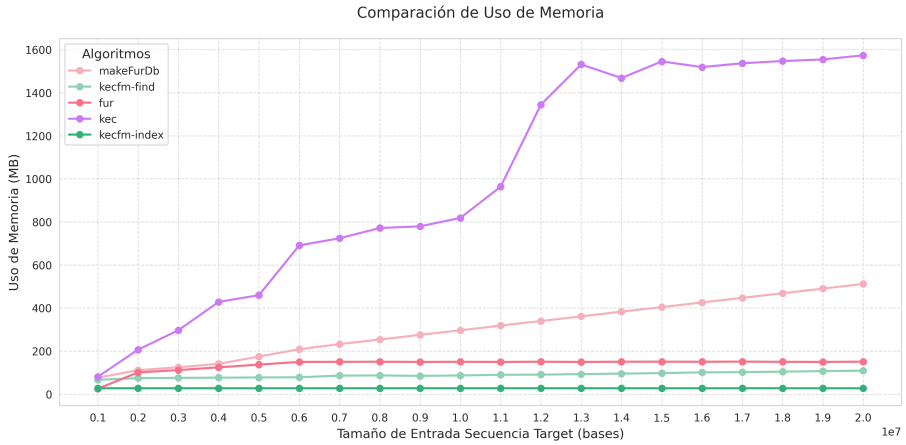

Figura 5.4: Uso máximo de memoria en función de tamaño de la secuencia *target*

Se aprecia en la figura 5.4 que al no aumentar el tamaño del genoma neighbor, la memoria requerida por `KEC-FM-index` es muy baja. Destacamos la baja huella de memoria de `KEC-FM-find`, la cual en este experimento resultó inferior incluso a la de `fur`.

5.3. Análisis para múltiples valores de k

`KEC-FM` implementa por defecto la capacidad de procesar múltiples valores de K de manera simultánea, esto permite explotar la tabla de búsqueda rápida del índice para acelerar el tiempo total de búsqueda.

Para visualizar el impacto de está funcionalidad generamos un escenario simulado con 5 targets y 50 neighbors de 1Mb cada uno sobre el que ejecutamos `KEC` y `KEC-FM` para valores de K desde 5 hasta 35. Para poder comparar, se presenta el tiempo acumulado de cada ejecución de `KEC` en contraste con 1 ejecución de `KEC-FM` para el rango de valores.

En la figura 5.5 vemos como, si comparamos tiempo de usuario, `KEC` es más rápido en total, mientras que si medimos el tiempo real transcurrido, `KEC-FM` es 3 veces más rápido. Esto es esperable, ya que la nueva implementación hace un fuerte uso del paralelismo, tanto a nivel de proceso como de hilos, para realizar el análisis múltiple.

En la figura 5.6 se presenta el uso de memoria. En su punto máximo `KEC-FM` en su etapa de indexado consumió el doble que `KEC` original, pero la etapa de búsqueda tuvo una huella de casi la mitad.

`KEC-FM` dispone de más parámetros ajustables, y es posible que calibrándolos correctamente se puedan mejorar los resultados tanto de tiempo como de memoria.

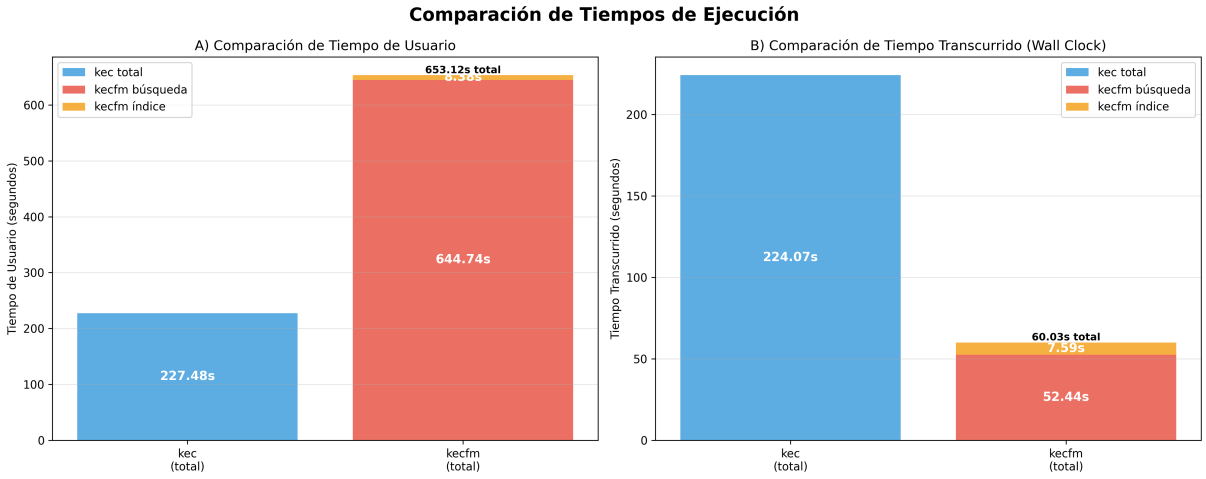

Figura 5.5: Tiempo acumulado de KEC en comparación con KEC-FM para $k = 5$ a $k = 35$

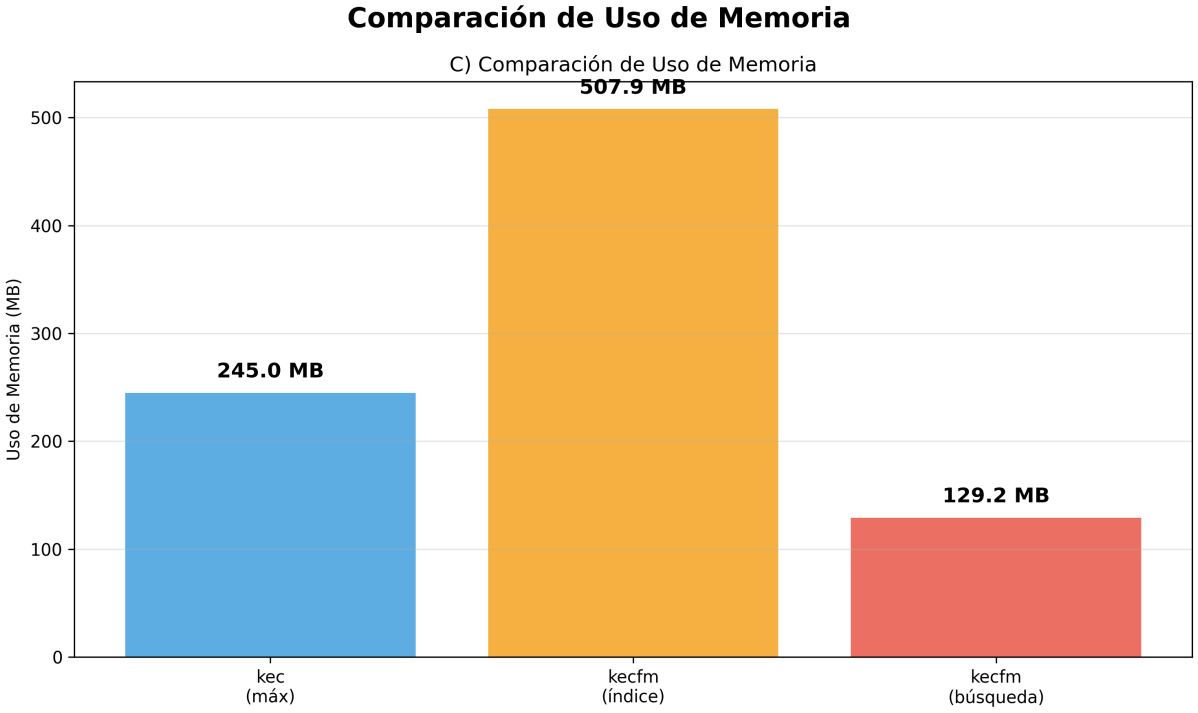

Figura 5.6: Tiempo acumulado de KEC en comparación con KEC-FM para $k = 5$ a $k = 35$

5.4. Conclusiones de implementación de KEC-FM

La nueva herramienta logra competir con las existentes, y representa una plataforma sólida sobre la que implementar futuras optimizaciones. Además, KEC-FM provee una variedad amplia de parámetros que se pueden modificar para lograr un balance entre uso de memoria y tiempo de ejecución según se requiera.

Se identificaron cuellos de botella en el manejo de múltiples archivos y cuando el conjunto *target* es muy numeroso. Aun así, consideramos que la herramienta es apta para su uso en genomas de gran tamaño o conjuntos muy numerosos debido a su empleo muy eficiente de la memoria.

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1. Conclusiones

Este trabajo de grado abordó el desafío de la identificación de regiones genómicas únicas para marcadores de diagnóstico, una tarea de fundamental importancia para aplicaciones como el diagnóstico molecular y la epidemiología. A través de una revisión de la literatura y posterior investigación, se ha proporcionado una visión clara de las fortalezas, debilidades y compromisos inherentes a los principales enfoques algorítmicos existentes.

Se estableció una metodología de evaluación comparativa que, mediante el uso de datos simulados con regiones únicas conocidas, permitió cuantificar objetivamente el desempeño de las herramientas `fur`, `GenMap` y `KEC`. Los experimentos de escalabilidad validaron y expandieron los resultados publicados en la literatura. Para toda la experimentación se desarrollaron scripts que permiten replicar y validar los resultados de manera automática. Su uso y acceso se detalla en el anexo A.2.

El análisis en escenarios de gran complejidad biológica desafió una noción preconcebida en la literatura reciente de que `KEC` no estaba a la altura de otras herramientas para la tarea. Si bien `fur` demostró una alta robustez y eficiencia, en escenarios de alta divergencia por mutación, el método de exclusión de k-meros de `KEC` fue capaz de identificar perfectamente las regiones marcadoras donde `fur` no obtuvo resultados. Este hallazgo, sumado a la identificación de posibles mejoras, motivó la profundización en el desarrollo de una herramienta basada en k-meros.

Como principal contribución, se diseñó e implementó `KEC-FM`, una reimplementación del algoritmo `KEC` la cual aborda una de sus debilidades arquitectónicas fundamentales: su incapacidad para reutilizar los cálculos entre ejecuciones con diferentes parámetros. Al adoptar un enfoque en dos fases, desacoplando la indexación de la búsqueda, y al reemplazar las tablas hash por un eficiente y comprimido Índice FM con tablas de búsqueda rápida, `KEC-FM` permite una exploración flexible y rápida de distintas longitudes de k-mero sin la necesidad de reconstruir el índice. Las pruebas de rendimiento validaron la escalabilidad de la nueva herramienta.

Adicionalmente, se desarrollaron herramientas para facilitar la búsqueda de marcadores de diagnóstico, las cuales están disponibles públicamente. Los detalles para su acceso se encuentran en el anexo A.3.

Trabajo Futuro

Los resultados y desarrollos de este proyecto abren diversas líneas de desarrollo a futuro:

- Creemos que es posible y deseable continuar el desarrollo de `KEC-FM` para soportar la coincidencia aproximada de k-meros, permitiendo un número e de errores, imitando a `GenMap` en su uso de esquemas óptimos de búsqueda. El Índice FM es adecuado para la implementación de dichas funcionalidades.
- Se debe realizar una perfilación en profundidad del tiempo de ejecución de `KEC-FM` para identificar cuellos de botella en la implementación. Especialmente para los casos con un conjunto *target* numeroso y la búsqueda para varios valores de k simultáneos.
- No se llegó a realizar el análisis de sensibilidad de parámetros para GenMap, considerando distintos valores para el parámetro E. Para una comparación más justa, es necesario estudiar en profundidad el impacto de la tolerancia a errores en la calidad de los marcadores obtenidos.

Referencias

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] Tim Anderson and Travis J. Wheeler. An optimized fm-index library for nucleotide and amino acid search. *Algorithms for Molecular Biology*, 16(1):25, 2021.

[3] Pavel Beran. Kec: A tool for finding unique regions in bacterial genomes for diagnostic pcr assay design. *Bioinformatics*, 2021.

[4] D. J. Brenner and A. G. Steigerwalt. Legionella pneumophila serogroup lansing 3 isolated from a patient with fatal pneumonia, and descriptions of l. pneumophila subsp. pneumophila subsp. nov., l. pneumophila subsp. fraseri subsp. nov., and l. pneumophila subsp. pascullei subsp. nov. *Journal of Clinical Microbiology*, 26:1695–1703, 1988.

[5] M. Alzamel C. Pockrandt and K. Reinert. Genmap: ultra-fast computation of genome mappability. *Bioinformatics*, 36:3687–3692, 2020.

[6] B. Haubold et. al. Fur: Find unique genomic regions for diagnostic pcr. *Bioinformatics*, 37:2081–2087, 2021.

[7] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 390–398. IEEE Computer Society, 2000.

[8] SimonGene Gottlieb and Knut Reinert. Search schemes for approximate string matching. *NAR Genomics and Bioinformatics*, 7(1):lqaf025, 03 2025.

[9] B. Haubold and B. Vieira Mourato. neighbors, 2024. Available at https://github.com/EvolBioInf/neighbors.

[10] B. Haubold and T. Wiehe. *Introduction to Computational Biology: An Evolutionary Approach.* Birkhäuser, Basel, 2006.

[11] Bernard Haubold. prim: A tool for designing primers for lineage-specific PCR, 2024.

[12] Bernhard Haubold. Biobox: Tools for bioinformatics, 2024. Available at https://github.com/EvolBioInf/biobox.

[13] Bernhard Haubold. *Simulate Targets and Neighbors, stan*, October 2024. Version 1.0-5. Documentation for the stan software.

[14] F. Klötzl and B. Haubold. Phylonium: fast estimation of evolutionary distances from large samples of similar genomes. *Bioinformatics*, 36:2040–2046, 2020.

[15] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[16] B. Vieira Mourato and B. Haubold et.al. Marker discovery in the large. *Bioinformatics Advances*, 4:vbae113, 2024.

[17] Anton Pirogov, Peter Pfaffelhuber, Angelika Börsch-Haubold, and Bernhard Haubold. High-complexity regions in mammalian genomes are enriched for developmental genes. *Bioinformatics*, 35(11):1813–1819, 2019.

[18] René Rahn and et al Budach. Generic accelerated sequence alignment in seqan using vectorization and multi-threading. *Bioinformatics*, 34(20):3437–3445, 05 2018.

[19] F Sanger, S Nicklen, and A R Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5463–5467, 1977.

[20] E.W. Sayers et al. Genbank. *Nucleic Acids Research*, 50(D1):D165–D170, 2022.

[21] Andreas Untergasser, Ioana Cutcutache, Triinu Koressaar, Jian Ye, Brant C. Faircloth, Maido Remm, and Steven G. Rozen. Primer3–new capabilities and interfaces. *Nucleic Acids Research*, 40(15):e115, aug 2012.

[22] D. Viasus and V. Gaia. Legionnaires' disease: update on diagnosis and treatment. *Infectious Diseases and Therapy*, 11:973–986, 2022.

[23] Beatriz Vieira Mourato and Bernhard Haubold. Fast detection of unique genomic regions. *Computational and Structural Biotechnology Journal*, 27:843–850, 2025.

[24] Peter Weiner. Linear pattern matching algorithms. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11. IEEE, 1973.

Anexo A

Anexo

A.1. Ejemplo de construcción de base de `makeFurDb`

Para ilustrar este proceso consideremos un ejemplo simple, tomado de la documentación adjunta a la publicación de `fur` [16].

Supongamos que tenemos una secuencia representante r dada por AAATAATATT y un conjunto de secuencias *neighbor* N que contiene dos secuencias: n_1=AATTTAAATT y n_2=TAATAATATT.

Si intentamos hacer coincidir la secuencia r con la secuencia n_1, observamos que existe una coincidencia del sub-string AAAT que comienza en la posición 1 de r y en la posición 6 de n_1 ($r[1.,4] = n_1[6.,9]$=AAAT).

A este sub-string coincidente en estas posiciones se le denomina "match factor".

La longitud de este match factor (en este caso, 4) se guarda en un arreglo que llamaremos `ml`, en la posición correspondiente al inicio del match factor en r.

Inicialmente, el arreglo `ml` se crea como un vector de ceros con la misma longitud que r.

En nuestro ejemplo, después de encontrar la primera coincidencia con n_1, el arreglo `ml` se vería así:

i	1	2	3	4	5	6	7	8	9	10
r	A	A	A	T	A	A	T	A	T	T
`ml`	4	0	0	0	0	0	0	0	0	0

Para continuar el proceso de búsqueda de otros match factors dentro de r que coincidan con n_1, se elimina de r el match factor ya encontrado (AAAT) y se repite el proceso con la parte restante de r.

En este ejemplo, se encontrarían posteriormente los factores AAT y ATT.

Al final de la comparación de r con n_1, el arreglo `ml` podría tener un aspecto similar a este:

i	1	2	3	4	5	6	7	8	9	10
r	A	A	A	T	A	A	T	A	T	T
`ml`	4	0	0	0	3	0	0	3	0	0

Una vez que se ha consumido toda la secuencia r en la búsqueda de coincidencias con n_1, se pasa a repetir el mismo proceso de comparación, pero ahora con la siguiente secuencia *neighbor*, n_2.

En esta etapa, solo se actualiza el valor en el arreglo `ml` si la nueva longitud de congruencia encontrada es superior al valor que ya estaba almacenado en esa posición.

En nuestro ejemplo, al comparar r con n_2, se encuentran los factores AA (de longitud 2) y ATAATATT (de longitud 8).

Como en la posición 1 de `ml` ya teníamos un valor de 4, que es mayor que la longitud del nuevo factor encontrado (2), no se actualiza esa entrada.

Sin embargo, en la posición 3 de `ml` teníamos un valor de 0, que es menor que la longitud del nuevo factor encontrado (8), por lo que se actualiza `ml` a 8. El arreglo `ml` después de la comparación con n_2 sería:

i	1	2	3	4	5	6	7	8	9	10
r	A	A	A	T	A	A	T	A	T	T
`ml`	4	0	8	0	3	0	0	3	0	0

Una vez que se ha comparado la secuencia representante r con cada una de las secuencias *neighbor* en el conjunto N, se fusionan los resultados obtenidos en el arreglo `ml`.

Esta fusión se realiza recorriendo el arreglo y, para cada posición, ingresando la longitud de congruencia implícita.

Esta longitud implícita se calcula como uno menos que la entrada a su izquierda, a menos que la entrada actual sea mayor.

Aplicando esta lógica a nuestro ejemplo, obtenemos el siguiente arreglo fusionado:

i	1	2	3	4	5	6	7	8	9	10
r	A	A	A	T	A	A	T	A	T	T
`ml`	4	3	8	7	6	5	4	3	2	1

En lugar del número total de factores de coincidencia encontrados, generalmente resulta de mayor interés la densidad local de estos factores.

La razón detrás de esto es que las regiones de la secuencia r donde localmente se observa una alta frecuencia de coincidencias, aunque sean de longitud pequeña, son indicativas de regiones que no están presentes en el conjunto de secuencias N. Esta densidad se cuantifica mediante un análisis de ventana deslizante.

Para simplificar el análisis de ventana deslizante, se construye un arreglo de posiciones finales de la siguiente manera: `en`[i] = 1 si una coincidencia termina en la posición i, y `en`[i] = 0 en caso contrario.

Aplicando esto a nuestro ejemplo, obtenemos:

i	1	2	3	4	5	6	7	8	9	10
r	A	A	A	T	A	A	T	A	T	T
`ml`	4	3	8	7	6	5	4	3	2	1
`en`	0	0	0	1	0	0	0	0	0	1

Este arreglo `en` es el principal insumo que utiliza el programa fur para su posterior análisis de búsqueda de regiones únicas.

Además de realizar todo el proceso descrito anteriormente, `makeFurDb` genera otros archivos y directorios que se listan a continuación:

1. Un archivo de texto llamado *v.txt* que contiene información sobre la versión del programa.
2. Un archivo en formato FASTA que contiene la secuencia representante de los *targets*, denominado *r.fasta*.
3. Un directorio que contiene todos los archivos FASTA de las secuencias *target*, excepto la secuencia representante r.
4. Un archivo llamado *e.fasta* que contiene el arreglo de posiciones finales de las coincidencias, `en`.
5. Una base de datos en formato compatible con el programa `BLAST`, construida a partir de las secuencias *neighbor* N.
6. Un archivo de texto llamado *n.txt* que contiene información sobre la longitud combinada de las secuencias *neighbor* y su contenido de nucleótidos guanina y citosina (GC).

A.2. Reproducción de experimentos

Para replicar todos los experimentos realizados durante el proyecto se provee el repositorio experimentacion-pgrado. En el se encuentran separados en directorios cada uno de los escenarios planteados. En el README del repositorio se encuentran las dependencias necesarias para correr dichos experimentos.

A.3. Herramientas para búsqueda de marcadores

En el repositorio Herramientas para búsqueda de marcadores se disponen las herramientas para calcular métricas, refinar marcadores, entre otras. En el README del repositorio se encuentran las dependencias necesarias, así como ejemplos de uso de las mismas.

A.4. Pipeline de refinamiento de marcadores

Partiendo de un archivo fasta de candidatos a marcadores implementamos un proceso de refinamiento cuantitativo mediante el script *refine_ markers.py*. Este pipeline está diseñado para evaluar sistemáticamente cada candidato en función de su especificidad y conservación, utilizando un análisis de dos etapas basado en la herramienta BLASTn contra genomas de un grupo *target* y un grupo *neighbor*.

El flujo de trabajo automatizado sigue los siguientes pasos:

1. **Preparación de las Bases de Datos:** Se crean dos bases de datos BLAST. Se agrupan y concatenan todos los archivos genómicos en formato FASTA correspondientes a los directorios *target* y neighbor. Posteriormente, se utiliza el comando `makeblastdb` para indexar cada conjunto de secuencias, generando una base de datos de referencia para cada grupo.

2. **Etapa de Sustracción (Análisis contra Genomas Vecinos):** En esta fase, se realiza una búsqueda `blastn` de todos los marcadores candidatos contra la base de datos neighbor. El propósito de esta etapa es identificar y cuantificar la presencia de marcadores en genomas no objetivo, lo que representa una medida de su potencial de reacción cruzada. Los parámetros para esta búsqueda fueron los siguientes:

   ```
   - E-value: 1e-10
   - Identidad: 95.0%
   - Cobertura (query coverage): 90.0%
   ```

3. **Etapa de Intersección (Análisis contra Genomas Objetivo):** Se realiza una segunda búsqueda `blastn` con los mismos parámetros, pero esta vez alineando todos los candidatos contra la base de datos target. Esta etapa es fundamental para verificar que los marcadores se encuentren conservados dentro del grupo de interés.

4. **Generación de Reporte y Clasificación:** Con los resultados de ambas búsquedas, el script calcula para cada marcador el número y el porcentaje de genomas en los que se encontró un "hit", tanto en el grupo objetivo como en el vecino. Basándose en estos porcentajes, se asigna una categoría de confianza a cada marcador:

   - **Alta Confianza:** Marcadores presentes en $> 95\,\%$ de los objetivos y en $0\,\%$ de los vecinos.
   - **Media Confianza:** Marcadores presentes en $> 90\,\%$de los objetivos y en $< 5\,\%$ de los vecinos.
   - **Baja Confianza:** Todos los demás casos.

5. **Selección Final de Marcadores:** El pipeline genera dos archivos de salida clave. Primero, un reporte completo en formato CSV (`final_marker_report.csv`) con las métricas detalladas y la clasificación de confianza para todos los candidatos. Segundo, un archivo FASTA (`high_confidence_markers.fasta`) que contiene únicamente las secuencias de los marcadores clasificados como de ' ' Alta Confianza' ', aislándolos como los candidatos óptimos para el diseño de primers y la validación experimental subsecuente.

A.5. Descripción técnica de `KEC-FM`

A.5.1. Creación del indice: `kecfm-index`

La configuración inicial del programa se realiza a través de argumentos de línea de comandos, procesados mediante la función `getopt`. Los parámetros por defecto incluyen el directorio de secuencias (`neighbors`), el nombre base del índice (`index`), un modo rápido activado por defecto, un ratio de compresión del arreglo de sufijos igual a 4, y una longitud de k-meros de 10 para la tabla de semillas (*seed table*).

1) Lectura de Secuencias FASTA El programa admite dos estrategias alternativas para la lectura de secuencias, seleccionadas según la disponibilidad de recursos del sistema:

En el modo por defecto, orientado a maximizar la velocidad de procesamiento, se realiza un pre-escaneo de las secuencias para estimar el tamaño total de los datos, tras lo cual:

- Se utiliza `mmap()` para mapear los archivos FASTA directamente en memoria virtual, eliminando la necesidad de copiar los datos a buffers intermedios.
- Se asigna un buffer continuo de memoria con una capacidad que excede en un 20 % la estimación inicial, minimizando la necesidad de realocaciones dinámicas.
- Las secuencias se extraen directamente desde la memoria mapeada, lo que permite un procesamiento eficiente.

Como alternativa más robusta, aunque menos eficiente, el programa puede procesar los archivos línea por línea mediante `fgets()`, utilizando buffers dinámicos que se expanden a medida que crece el tamaño de las secuencias.

2) Procesamiento de Archivos

Durante el procesamiento, cada archivo con extensión `.fa` o `.fasta` es recorrido omitiendo las líneas que comienzan con el carácter `>`, correspondientes a los encabezados. Las secuencias de nucleótidos se extraen y concatenan en un único buffer continuo, que posteriormente será indexado.

3) Configuración del Índice

El índice FM se configura utilizando la estructura `AwFmIndexConfiguration`, cuyos parámetros están ajustados para priorizar la velocidad de búsqueda:

```
struct AwFmIndexConfiguration config = {
.suffixArrayCompressionRatio = compression_ratio, // Menor = más rápido
.kmerLengthInSeedTable = kmer_seed_length, // Menor = más rápido
.alphabetType = AwFmAlphabetDna,
.keepSuffixArrayInMemory = true, // Para velocidad
.storeOriginalSequence = false // Para velocidad
};
```

Creación del Índice

Con las secuencias concatenadas y la configuración definida, el programa llama a `awFmCreateIndex()`, generando un archivo binario con extensión `.awfmi` que almacena el índice FM resultante. El proceso incluye verificaciones de éxito en la creación del índice.

Limpieza

Finalmente, el programa libera la memoria asignada para las secuencias y el índice, y retorna un código de salida que indica el estado final de la ejecución.

Optimización del Rendimiento

Las principales optimizaciones implementadas en el proceso son:

- **Memory Mapping**: Permite un acceso rápido y eficiente a archivos grandes.
- **Buffers Pre-asignados**: Minimiza la sobrecarga de realocaciones dinámicas.
- **Procesamiento Directo**: Extrae secuencias sin realizar copias intermedias, reduciendo la latencia.
- **Configuración Ajustada**: Los parámetros del índice se ajustan para maximizar la velocidad de acceso durante las búsquedas posteriores.

A.5.2. Búsqueda de regiones: `kecfm-find`

1) Inicialización y Configuración El proceso comienza con la lectura de los parámetros de ejecución proporcionados por el usuario, que incluyen el archivo de índice FM pre-construido, el directorio que contiene las secuencias objetivo en formato FASTA, el rango de valores de k a utilizar, y el archivo de salida donde se registrarán los resultados. El índice FM, que contiene las secuencias de referencia (*neighbors*), se carga mediante la función `awFmReadIndexFromFile()`, permitiendo búsquedas posteriores de k-meros sobre este conjunto.

2) Procesamiento por Valores de k Una de las optimizaciones clave implementadas es el procesamiento secuencial por cada valor de k. A diferencia de procesar múltiples tamaños de k-mer simultáneamente, este enfoque minimiza el uso de memoria, ya que solo mantiene en memoria las estructuras necesarias para el valor de k actual.

3) Procesamiento de Secuencias FASTA Cada archivo FASTA del conjunto objetivo es procesado mediante técnicas de *memory mapping*, utilizando la función `mmap()`. Esto permite mapear el archivo completo en memoria virtual, evitando la carga explícita de datos en buffers y facilitando un acceso eficiente sin duplicación innecesaria de datos.

Posteriormente, se realiza el análisis del encabezado de cada secuencia para extraer su identificador. Las secuencias se procesan en fragmentos (*chunks*) de 512KB, con un solapamiento de $k-1$ caracteres entre fragmentos consecutivos, garantizando que ningún k-mer se pierda debido a los límites del fragmento.

4) Generación y Búsqueda de K-meros Dentro de cada fragmento, los k-meros se generan y almacenan en estructuras de tipo `KmerBatch`, optimizadas mediante buffers contiguos para reducir la sobrecarga de memoria.

Las búsquedas se realizan en paralelo utilizando `awFmParallelSearchCount()` con 8 hilos, lo que permite consultar simultáneamente todos los k-meros contra el índice FM. El análisis posterior identifica regiones continuas donde los k-meros no tienen ocurrencias en las secuencias de referencia (cuentas igual a cero), lo que indica potenciales regiones únicas.

5) Gestión y Optimización de Regiones Las regiones candidatas se almacenan como nodos simples que contienen solo las posiciones inicial y final `RegionNode`, sin almacenar las secuencias completas, lo que reduce el consumo de memoria. Además, se implementa una función de fusión inteligente (`merge_regions()`) que ordena las regiones por posición y consolida aquellas adyacentes o superpuestas, reduciendo la fragmentación y el volumen de los resultados.

6) Extracción y Almacenamiento de Secuencias Las secuencias correspondientes a las regiones identificadas se extraen del archivo mapeado solo cuando es necesario escribirlas en el archivo de salida, mediante la función `extract_sequence_from_region()`, que accede eficientemente a los datos mapeados. El formato de salida incluye encabezados informativos con el número de región, el valor de k, el nombre de la secuencia original, las posiciones de inicio y fin, y la longitud total de la región. En caso de errores durante la extracción, se sustituyen los caracteres por 'N' como mecanismo de contingencia.

A.6. Manual `KEC-FM`

KEC-FM funciona en dos fases: creación del índice y búsqueda.

A.6.1. Fase 1: Creación del Índice (`kecfm-index`)

```
kecfm-index [OPCIONES] <directorio_neighbors> [nombre_indice]
```

Parámetros:

Parámetro	Tipo	Descripción	Valor por defecto
neighbor_directory	Requerido	Directorio con FASTA de referencia	-
index_name	Opcional	Nombre base del índice (.awfmi)	`index`
`-f`, `-fast`	Flag	Lectura rápida con memory mapping	Desactivado
`-c`, `-compression`	Entero	Ratio de compresión usado por el índice (1-10)	4
`-s`, `-seed-length`	Entero	Longitud del k-mero semilla para la look-up table (6-16)	8
`-h`, `-help`	Flag	Mostrar ayuda	-

Ejemplo:

```
kecfm-index -f -c 2 -s 6 neighbors/ fast_index
```

A.6.2. Fase 2: Búsqueda de Regiones Únicas (`kecfm-find`)

```
kecfm-find [OPCIONES] -i <archivo_indice> -d <directorio_target>
```

Parámetros:

Parámetro	Tipo	Descripción	Valor por defecto
`-i`, `-index`	Requerido	Archivo de índice (.awfmi)	-
`-d`, `-directory`	Requerido	Directorio con FASTA objetivo	-
`-k`, `-kmer`	Entero	Tamaño único de k-mer	12
`-r`, `-range`	Rango	Rango de k-meros (ej: 10-15)	-
`-o`, `-output`	String	Archivo de salida (FASTA)	`result.fasta`
`-h`, `-help`	Flag	Mostrar ayuda	-

Ejemplo con rango de k-meros:

```
kecfm-find -i index.awfmi -d targets/ -r 8-20 -o unique_regions.fasta
```