Universidad Nacional Autónoma de México

Facultad de Ciencias

Genómica Computacional

Proyecto: Uso de gráficas de De Bruijn para el ensamble de un genoma completo

- *Francisco Alejandro Arganis Ramírez*
- *Erika Yusset Madera Baldovinos*

Introducción

Actualmente, el uso de instrumentos de secuenciación automatizada de DNA de nueva generación se ha vuelto una práctica común en biología molecular. Aunque la secuenciación de Sanger puede generar largas lecturas de alrededor de 1000 bp con una exactitud por base de 99.999 % (Shendure, J. & Ji, H., 2008), lo más común es usar plataformas de secuenciación de nueva generación tales como Illumina MiniSeq o similares que generan múltiples lecturas cortas de entre 100 y 200 bp. La secuenciación de nueva generación es más barata y rápida porque permite la secuenciación de múltiples fragmentos en una sola corrida a diferencia del método de Sanger que solo puede secuenciar una molécula de DNA a la vez. Sin embargo, las lecturas cortas de la secuenciación de nueva generación también hacen más complicado el ensamble.

El ensamble de un genoma completo a partir de las lecturas de secuenciación es una de las tareas más importantes en biología molecular. Si ya se ha ensamblado el genoma del mismo organismo o de alguno muy similar, se puede realizar un ensamble por genoma de referencia en el que primero se alinean las lecturas contra la referencia para reducir la complejidad del ensamble (Lischer, H.E. & Shimizu, K.K., 2017). Si no se cuenta con un genoma de referencia, es necesario realizar un ensamble *de novo*.

Existen dos estrategias principales para realizar un ensamble *de novo*: el consenso por disposición de traslapes (overlap layout consensus u OLC) y las gráficas de De Bruijn (De Bruijn graph o DBG). Ambas estrategias solo corresponden a la etapa de construcción de contigs. El proceso completo de ensamble *de novo* se divide generalmente en cuatro etapas: corrección de errores, construcción de contigs, vínculo de scaffolds y llenado de huecos (Li, Z. *et al.,* 2011).

En algoritmos OLC se construye una gráfica de traslapes en la que las lecturas se colocan en vértices y se coloca una arista dirigida con peso *w* del vértice *a* al vértice *b* si existe un traslape de tamaño w entre un sufijo de *a* y un prefijo de *b*. Esto se hace típicamente alineando pares de lecturas todos contra todos (Li, Z. *et al.,* 2011). Después se simplifica la gráfica eliminando aristas que se pueden inferir por transitividad y se construyen los contigs siguiendo un camino hamiltoniano. Por último, se alinean los contigs para llegar a un consenso y decidir la base correcta de cada posición.

En algoritmos DBG se fragmentan las lecturas en *k*-meros y se construye una gráfica de De Bruijn en la que cada *k*-mero corresponde a una arista dirigida del vértice a al vértice b, donde *a* tiene las primeras *k*-1 bases y *b* las últimas *k*-1 bases del *k*-mero. Para los casos en los que se pueden repetir *k*-meros, la multiplicidad se representa como el peso de las aristas. Los contigs se construyen siguiendo un camino euleriano en la gráfica de De Bruijn.

Objetivos

- Implementar el algoritmo de ensamble *de novo* de genomas usando gráficas de De Bruijn
- Explicar el funcionamiento del algoritmo en un contexto biológico, así como sus ventajas y desventajas frente a otras alternativas

Metodología

Se implementó el algoritmo DGB en Python. La clase DeBruijnGraph construye una gráfica de De Bruijn a partir de un objeto iterable de *k*-mero y ensambla el genoma usando el algoritmo de Hierholzer para encontrar un camino euleriano.

También, se implementaron dos métodos de corrección de errores. El primero es una corrección por frecuencia, que descarta aquellos *k*-meros que aparecen menos de un valor umbral *t* de veces. Esto es bajo el supuesto de que los *k*-meros con errores son menos frecuentes que los *k*-meros correctos. El segundo es una corrección usando la distancia de Hamming. Este método, en lugar de descartar los *k*-meros poco frecuentes, explora la vecindad de *k*-meros que se encuentran a 1 unidad de distancia de Hamming, es decir, que solo difieren en una base. Si encuentra un vecino que aparece más del valor umbral *t* veces, reemplaza al k-mero poco frecuente por el vecino.

Se descargaron las lecturas de secuenciación de nueva generación del fago Bacata de *Xylella*, secuenciado con la plataforma Illumina MiniSeq, del ENA Browser. Los archivos fastq con las 54495 lecturas, uno por cada dirección, están disponibles en https://www.ebi.ac.uk/ena/browser/view/ERX5328366. También, se descargó del NCBI el archivo fasta con el genoma ya ensamblado del mismo fago, de tamaño 56232 bp, disponible en https://www.ncbi.nlm.nih.gov/nuccore/NC_052973.1?report=fasta. La calidad de las lecturas se evaluó con el software FastQC.

Se fijó el valor de *k* en 35 por ser el tamaño mínimo de una lectura en los datos de la secuenciación. Adicionalmente, valores de *k* entre 20 y 50 son comunes para el ensamble con algoritmos DBG (Mahadik, K. *el at.*, 2019). Con esta *k*, se aplicó el algoritmo implementado para ensamblar el genoma del fago Bacata, con los siguientes datos de entrada:

1. Datos crudos. Todos los *k*-meros de todas las lecturas tal cual están en el archivo fastq descargado.
2. Datos de calidad. Los *k*-meros de las lecturas en las que todas las bases tienen al menos un valor q-score de 30 y tienen una longitud entre 100 y 150 bp.

3. Datos corregidos por frecuencia. Los *k*-meros de los datos de calidad después de aplicar el método de corrección de errores por frecuencia con *t* igual a 4.
4. Datos corregidos por distancia de Hamming. Los *k*-meros de los datos de calidad después de aplicar el método de corrección de errores por distancia de Hamming con *t* igual a 4.
5. Datos simulados. Los *k*-meros de 54495 lecturas simuladas aleatoriamente del genoma ya ensamblado. Todas las lecturas simuladas corresponden a un fragmento de entre 100 y 150 bp del genoma y son libres de errores.

Para todos los datos de entrada, se usaron las lecturas forward y se construyeron dos gráficas de De Bruijn: una con todos los *k*-meros, incluyendo repeticiones, y otra solo con el conjunto de *k*-meros únicos. Para cada gráfica de De Bruijn se ejecutó el algoritmo y el resultado del ensamble se guardó en un archivo fasta.

Por cada secuencia ensamblada se realizó una consulta BLAST de nucleótidos en https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome. En los casos donde el resultado del ensamble fue mayor al tamaño del genoma del fago, solo se tomaron las primeras 56232 bases.

Se generó un genoma sintético de tamaño 26128 bp usando un proceso de expansión-modificación por 1000 iteraciones. Iniciando con una base aleatoria, en cada etapa del proceso, con probabilidad 0.75 se realizó una mutación aleatoria en la secuencia y con 0.25 de probabilidad se duplicaron las últimas 100 bases de la secuencia.

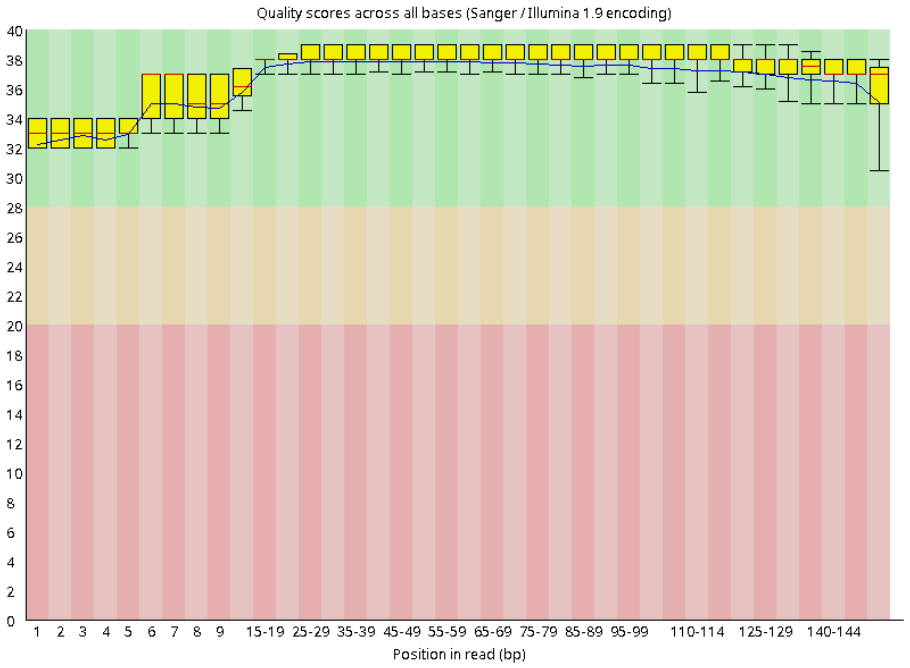
Se simularon 50000 lecturas de tamaño entre 100 y 150 bp del genoma sintético y se aplicó el algoritmo DBG con valores de *k* desde 20 hasta 50, en pasos de 5, para ensamblar el genoma usando todos los *k*-meros de las lecturas, incluyendo repetidos. El resultado de cada ensamble se alineó con el genoma sintético original usando una consulta BLAST. En los casos donde el resultado del ensamble fue mayor al tamaño del genoma sintético, solo se tomaron las primeras 26128 bases.

Se guardó el alineamiento de cada consulta BLAST en un archivo de texto. Todo el código y los archivos están disponibles en el repositorio https://github.com/FranciscoAlejandroArganis/genomica-computacional-proyecto.

Resultados y discusión

Del reporte de calidad de las lecturas forward y reverse se concluye que las lecturas son, en términos generales, de buena calidad. Como se observa en los módulos per base sequence quality, la media del q-score en todas las posiciones de las lecturas es mayor a 30. También, en el módulo per sequence quality scores se observa un único pico en 37, indicando que la mayoría de las lecturas tienen una calidad media alta. En el módulo per sequence GC content se observa que la distribución es similar a la teórica, con una ligera desviación. Las gráficas del módulo sequence length distribution para las lecturas forward y reverse tienen un pico pronunciado en el tamaño 150 a 152 bp, indicando que la mayoría de las lecturas son de tamaño uniforme. También, en ninguno de los dos reportes se tienen secuencias sobrerrepresentadas ni adaptadores.

En el módulo per base sequence content, tanto para las secuencias forward como para las secuencias reverse, se observa que las proporciones de las bases no son una línea recta, como es esperado, para las primeras 20 bases. Adicionalmente, en el módulo sequence duplication levels se observa que hay varias secuencias duplicadas al menos 10 veces. Sin embargo, esto no parece ser un indicativo de que la secuenciación haya sido de mala calidad. Los autores del estudio en el que se secuenció y ensambló el genoma del fago Bacata tampoco reportan ningún problema de calidad (Clavijo-Coppens, F. *et al.*, 2021).



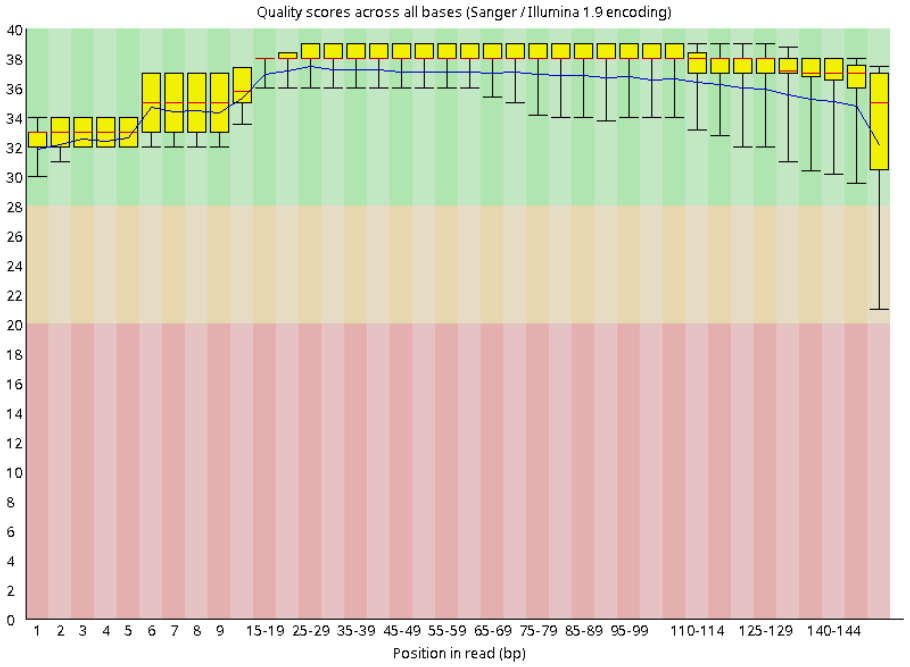


Figura 1. Modulos per base sequence quality del reporte FastQC, para las lecturas forward (arriba) y reverse (abajo) de la secuenciación del fago Bacata.

Los resultados del alineamiento del ensamble del genoma del fago Bacata con el algoritmo DBG implementado, se muestran a continuación.

Datos de entrada	Tamaño del ensamble (bp)	1er región alineada	2da región alineada	3er región alineada
Datos crudos	3612673	Región: 40466-42426 Coincidencias: 1960/1961	Región: 40466-42418 Coincidencias: 1952/1953	Región: 40466-42413 Coincidencias: 1944/1948
Datos crudos (*k*-meros únicos)	215852	Región: 29039-29367 Coincidencias: 328/330	Región: 42764-43090 Coincidencias: 326/329	Región: 34526-34849 Coincidencias: 322/326
Datos de calidad	108159	Región: 37114-43918 Coincidencias: 6804/6805	Región: 36427-43221 Coincidencias: 6794/6795	Región: 36427-41028 Coincidencias: 4600/4603
Datos de calidad (*k*-meros únicos)	3544	Región: 23283-25620 Coincidencias: 2337/2341	Región: 22543-23114 Coincidencias: 572/572	Región: 25588-25978 Coincidencias: 391/391
Datos corregidos por frecuencia	4375	Región: 47316-51690 Coincidencias: 4375/4375	NA	NA
Datos corregidos por frecuencia (*k*-meros únicos)	3989	Región: 36822-40810 Coincidencias: 3989/3989	NA	NA
Datos corregidos por distancia de Hamming	14949	Región: 29023-43971 Coincidencias: 14949/14949	NA	NA
Datos corregidos por distancia de Hamming (*k*-meros únicos)	14446	Región: 46096-56232 Coincidencias: 10137/10137	Región: 1-4309 Coincidencias: 4309/4309	NA
Datos simulados	3006626	Región: 1-40036 Coincidencias: 40036/40036	Región: 44432-56232 Coincidencias: 11801/11801	Región: 39955-44426 Coincidencias: 4472/4472
Datos simulados (*k*-meros únicos)	56190	Región: 1-26761 Coincidencias:	Región: 26759-44426	Región: 44432-56232

		26761/26761	Coincidencias: 17668/17668	Coincidencias: 11801/11801

Tabla 1. Por cada entrada, se muestra el tamaño completo de la secuencia ensamblada por el algoritmo, las posiciones de inicio y fin en el genoma del fago Bacata y la cantidad de bases que coinciden de las tres regiones mejor alineadas de la secuencia ensamblada.

Como se observa en la tabla anterior, en ningún caso el algoritmo DGB implementado genera una secuencia que contiene a las 56232 bases del genoma del fago.

La estrategia de gráficas de De Bruijn es muy atractiva para el desarrollo de software de ensamble a partir de lecturas cortas. En situaciones con datos idealizados, es un algoritmo que puede ensamblar eficientemente las lecturas. Sin embargo, en la práctica presenta muchas dificultades que se tienen que resolver para poder ensamblar correctamente un genoma.

El principal problema para un algoritmo DBG son los errores en las lecturas. La razón promedio de errores puede ser entre 1 y 1.5 % para secuenciación de nueva generación (Shendure, J. & Ji, H., 2008). Ya sea que los errores son cometidos por las enzimas durante la amplificación de la cadena de DNA, por el equipo que distingue la mayor intensidad de la fluorescencia o provengan de alguna otra fuente, siempre se debe asumir que los datos de secuenciación tienen errores, sin importar que los valores de q-score sean altos.

En la gráfica de De Bruijn, cada *k*-mero es una arista dirigida entre dos vértices, uno con el prefijo de las primeras *k*-1 bases y el otro con el sufijo de las últimas *k*-1 bases. El objetivo del algoritmo es encontrar un camino euleriano, si el genoma es lineal, o un ciclo euleriano, si el genoma es circular. Al pasar por todas las aristas, por construcción de la gráfica, se asegura que existe un traslape entre dos *k*-meros consecutivos, que es precisamente las *k*-1 bases del vértice en común. Por tanto, al recorrer todos los *k*-meros se recupera el genoma original.

Cuando una lectura tiene un error, todos los *k*-meros que pasan por el error se ven afectados. Esto puede causar que la gráfica deje de ser euleriana, que se generen nuevos vértices, que la gráfica se vuelva desconexa u otros cambios en su topología.

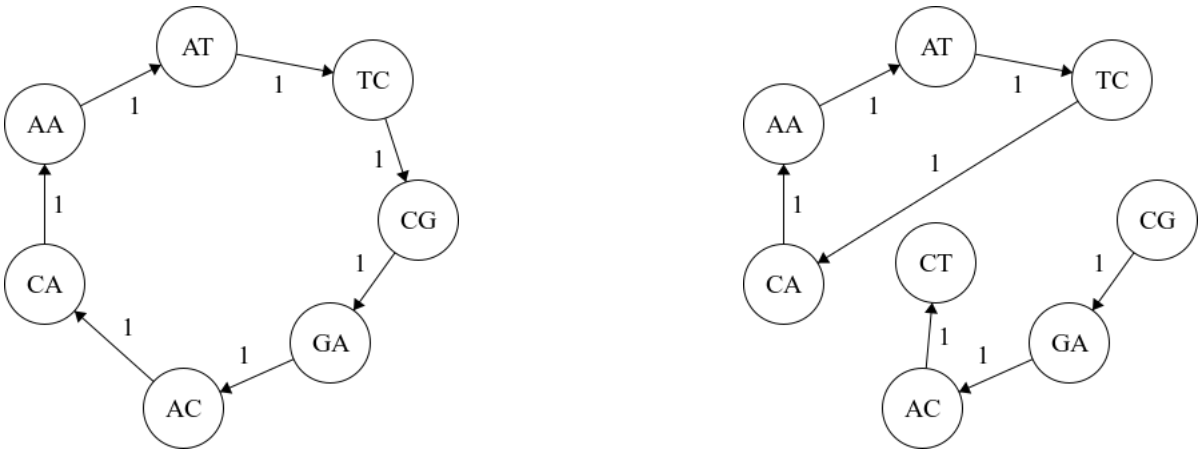


Figura 2. Gráficas de De Bruijn construidas a partir de las lecturas sin errores {AAT,ATC,TCG,CGA,GAC,ACA,CAA} (izquierda) y a partir de las lecturas con dos errores {AAT,ATC,TCA,CGA,GAC,ACT,CAA} (derecha), con *k* = 3. Solo es posible recuperar el genoma original AATCGACA a partir de la primera gráfica.

Los errores de secuenciación transforman una simple gráfica de De Bruijn, correspondiente a lecturas libres de errores, en una maraña de aristas erróneas (Pevzner, P.A., Tang, H. & Waterman, M.S., 2001). Los errores de secuenciación evitan que se puedan formar correctamente caminos eulerianos en la gráfica de De Bruijn, por lo que es una etapa fundamental el preprocesamiento de los datos para corregir errores en algoritmos DBG. Los *k*-meros erróneos también implican mayor consumo de memoria para construir la gráfica (Li, Z. *et al.*, 2011).

Esto claramente se refleja en el ensamble de con los datos crudos. La tabla 2 muestra que sin ningún preprocesamiento, la secuencia ensamblada apenas y se alinea en una región de 2000 bp, la cual se repite múltiples veces.

La forma más básica de preprocesamiento es un filtro de calidad. Se descartan aquellas lecturas que no tienen un valor mínimo de calidad. La plataforma Illumina recomienda considerar un valor q-score de al menos 30, que indica una probabilidad de error de 0.1 %. Sin embargo, solo este filtro no es suficiente. Para los datos de calidad, que solo incluyen lecturas donde todas las posiciones tienen q-score mayor o igual a 30, el mejor alineamiento de la secuencia ensamblada con el genoma es una región de 6800 bp. Si bien es una mejora respecto a los datos crudos, esta región sigue siendo muy pequeña y representa una fracción mínima del total de la secuencia ensamblada.

Se han propuesto diversos algoritmos de corrección de errores. Una clase de algoritmos están basados en alineamiento de lecturas. Primero se encuentran los traslapes entre lecturas haciendo múltiples alineamientos y después se distinguen los errores de las bases correctas a través de un modelo probabilístico (Li, Z. *et al.*, 2011). Sin embargo, los alineamientos son costosos en CPU.

Una segunda clase de algoritmos de corrección de errores está basada en la frecuencia de los *k*-meros. Se espera que dado, que los errores de secuenciación son poco frecuentes,

k-meros que pasan por un error también son menos frecuentes que el resto de *k*-meros correctos. En la práctica, la situación es más compleja que esto; algunos k-meros erróneos aparecen en alta frecuencia, algunos k-meros correctos aparecen en baja frecuencia y múltiples errores de secuenciación cercanos entre sí pueden crear un conjunto más grandes de k-meros de baja frecuencia (Li, Z. *et al.*, 2011).

Al utilizar los datos corregidos por frecuencia se obtuvo un contig de tamaño 4300 bp completamente libre de errores. El tamaño del contig sugiere que el filtro de *k*-meros poco frecuentes hace que la gráfica de De Bruijn se vuelva disconexa pero en un componente de conexidad se puede encontrar más fácilmente un camino euleriano libre de errores.

Otros algoritmos de corrección intentan corregir los k-meros poco frecuentes en lugar de descartarlos. Esta es la estrategia utilizada por muchos programas de ensamble como Euler, SOAPdenovo, Reptile y Quake (Li, Z. *et al.*, 2011). Para poder corregir las lecturas sin tener la secuencia ya ensamblada, se aproxima el conjunto de *k*-meros correctos del genoma con el conjunto de *k*-meros frecuentes de las lecturas (Pevzner, P.A., Tang, H. & Waterman, M.S., 2001). Dado un *k*-mero poco frecuente, se busca la mínima cantidad de mutaciones que lo conviertan en un *k*-mero frecuente. Una forma de simplificar esta búsqueda es considerar el cambio de una única base en el *k*-mero. Esto es, buscar un *k*-mero frecuente que se encuentra a 1 de distancia de Hamming del *k*-mero poco frecuente.

Al aplicar el algoritmo implementado a los datos corregidos con distancia de Hamming, se obtiene un contig de 15000 bp libre de errores, una mejora de casi 3 veces más grande sobre la corrección de errores por frecuencias. Es evidente de este resultado que corregir los *k*-meros poco frecuentes ayuda a que la gráfica de De Bruijn tenga caminos eulerianos más largos, potencialmente en componentes conexos grandes y sin aristas erróneas.

En software que usa la estrategia OLC, el preprocesamiento suele consistir solo en un filtro de lecturas de baja calidad, pero normalmente se omite un algoritmo de corrección de errores (Pevzner, P.A., Tang, H. & Waterman, M.S., 2001). OLC es mucho más tolerante a errores pues los alineamientos por pares pueden permitir bases que no coinciden sin aumentar la complejidad computacional. Los errores que queden después de construir los contigs en OLC se corrigen durante la etapa de consenso, en la que se decide la base correcta por mayoría. Dejar la corrección de errores hasta el final como parte del ensamble, en lugar de requerir un algoritmo especializado de corrección como preprocesamiento es una de las grandes ventajas de OLC.

Otro problema de los algoritmos DBG es que aún, si se pudiese garantizar que las lecturas son 100 % libres de errores, no hay garantía de que la gráfica de De Bruijn construida será euclidiana. En teoría de gráfica se demuestra que en una gráfica conexa dirigida tiene un ciclo euleriano si y solo si todos sus vértices son balanceados, es decir, la diferencia de aristas que salen del vértice menos aristas que entran del vértice es 0. De forma similar, existe un camino euleriano si y solo si todos los vértices son balanceados excepto dos, uno de los cuales debe tener balance 1 y el otro balance -1, que son los dos vértices de los extremos del camino.

Si el conjunto de *k*-meros de las lecturas correspondiera exactamente a los *k*-meros obtenidos al pasar una ventana de tamaño *k* por todo el genoma, se podría garantizar que la gráfica de De Bruijn siempre será euclidiana. Esto es porque, para cualquier posición *i* en el genoma, el *k*-mero que empieza en *i* aporta una arista que entra al vértice de las últimas *k*-1 bases de ese *k*-mero, pero el *k*-mero que empieza en *i*+1 aporta una arista que sale del mismo vértice. Todos los vértices entonces tienen una arista que entra por cada arista que sale y son balanceados, con la posible excepción de los vértices que corresponden a las primeras y las últimas *k*-1 bases de un genoma lineal. Si el genoma es lineal, no hay una ventana de tamaño *k* que termine en las primeras *k*-1 bases del genoma ni una ventana que empiece en las últimas *k*-1 bases. En este caso se tendría un vértice con una arista más que sale y otro con una arista más que entra, es decir, un par de vértices con balance 1 y -1, por lo que la gráfica tendría un camino euleriano.

Desafortunadamente, los *k*-meros de las lecturas de secuenciación reales no coinciden con la ventana de tamaño *k* idealizada. La cobertura de la secuenciación suele no ser uniforme. Regiones con poca cobertura resultan en una gráfica desconexa y diferencias en la cobertura resultan en una gráfica no euclidiana (Kingsford, C., Schatz, M.C. & Pop, M., 2010).

Aún si la gráfica de De Bruijn es euclidiana, no hay ninguna garantía de que el camino euclidiano es único, como se muestra en el ejemplo de la figura 3.

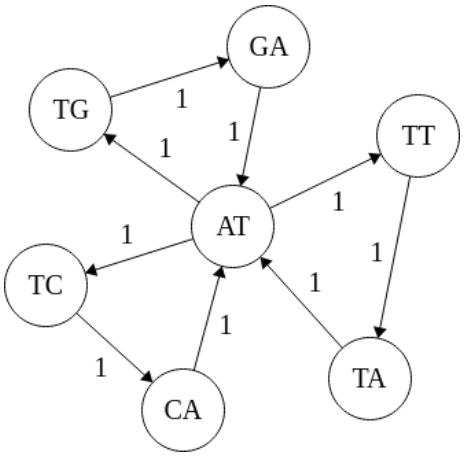


Figura 3. Gráfica de De Bruijn construida a partir de las lecturas {ATG,TGA,GAT,ATT,TTA,TAT,ATC,TCA,CAT} con *k* = 3. Existen múltiples ciclos eulerianos, los cuales corresponden a dos genomas circulares diferentes: ATGATTATC y ATGATCATT. No es posible distinguir cuál es el genoma correcto usando solo los datos de las lecturas.

Si es que existe, un ciclo o camino euleriano se puede encontrar en una gráfica con el algoritmo de Hierholzer. El algoritmo empieza en un vértice y, mientras haya aristas que recorrer, visita un nuevo vértice, eliminando la arista por la que pasó. Esto continúa hasta que eventualmente no hay más aristas en el vértice actual. En este punto, el algoritmo extiende el camino actual regresando al último vértice visitado que todavía tenía aristas que salen. Cuando ya no hay más vértices con aristas que salen, el algoritmo termina y regresa

el camino construido. Para generar un ciclo euleriano no importa el vértice inicial, pero para generar un camino euleriano es necesario empezar en el vértice con balance 1.

La gran ventaja del algoritmo de Hierholzer es que tiene una complejidad en tiempo lineal en la cantidad de aristas en la gráfica. Considerando que se tiene un genoma de tamaño *G* y la secuenciación produce N lecturas de tamaño *L*, la complejidad en tiempo para construir la gráfica de De Bruijn es O(*NL*) pues cada lectura tiene *L*-*k*+1 *k*-meros y por cada uno de los *N*(*L*-*k*+1) *k*-meros, se puede agregar una arista dirigida en tiempo constante a la gráfica. Consecuentemente, el tiempo para ejecutar el algoritmo de Hierholzer es también O(*NL*). En cuanto a espacio, la gráfica requiere a lo más 2 vértices por cada arista. Usando aristas con peso para indicar la multiplicidad, en el peor caso cada vértice guarda 4 aristas, una por cada posible base. De esta manera, una cota superior para el espacio es O(*NL*). Sin embargo, ya que no pueden haber más *k*-meros diferentes que los *G*-*k*+1 que hay en el genoma, la cantidad de aristas está acotada por O(*G*). Por el mismo argumento para los *k*-1-meros, los vértices también están acotados por O(*G*). Esto resulta en una complejidad en espacio de O(min(*NL*,*G*)).

El tiempo O(*NL*) y espacio O(min(*NL*,*G*)) en DBG es muy favorable comparado contra la complejidad en OLC. Construir la gráfica de traslapes se puede realizar en tiempo y espacio O(max(*NL*,N^2)) usando un árbol de sufijos o en O(N^2L^2) usando programación dinámica, lo cual permite alineamientos con bases que no coinciden y espacios (Langmead, B., 2018). Esto es sin contar el tiempo necesario para construir el contig a partir de la gráfica de traslapes, lo cual es un problema de camino hamiltoniano y es NP-completo.

Durante el algoritmo de Hierholzer, si un vértice tiene más de una arista que sale, es posible elegir cualquiera de ellas. La elección resulta en caminos eulerianos diferentes. En estos casos se pueden hacer modificaciones al algoritmo para intentar encontrar un camino que sea consistente con todas las lecturas. Esto es, plantear el ensamble como un problema de supercamino. El problema del supercamino consiste en encontrar el camino tal que contiene a todos los subcaminos de un conjunto dado. En el contexto de ensamblar genomas, el conjunto de subcaminos es el conjunto de caminos definidos por los *k*-meros de cada lectura. El problema del supercamino es NP-completo (Narzisi, G., Mishra, B. & Schatz, M.C., 2014).

Al simular lecturas del genoma del fago Bacata, todos los k-meros obtenidos son correctos. Por tanto, es como si se contara con un algoritmo de corrección de errores perfecto. Como se observa en la tabla 2, al usar los datos simulados se obtiene una secuencia con una región de 51800 bp que se alinea sin errores con el genoma del fago, seguido de una región de 4400 bp que se alinea sin errores con el resto del genoma. Al tomar las primeras 56232 bases del resultado del ensamble, esencialmente se logra recuperar el genoma completo, menos unas cuantas bases que se pueden corregir en la etapa de cerrado de huecos.

El hecho de que se haya logrado obtener el genoma casi 100 % completo del fago es porque, para el valor de k = 35, casi todos los *k*-meros del genoma son únicos. Por tanto, durante el algoritmo de Hierholzer no se tiene que decidir entre seguir por dos aristas diferentes, sino que todas las aristas que salen del vértices son la misma, solo con

multiplicidad mayor a 1. La multiplicidad de las aristas, que se debe a las repeticiones de *k*-meros por la gran cantidad de lecturas, hace que el camino se repita muchas veces, resultando en el tamaño de 3006626 bp.

Esto no es una dificultad ya que el tamaño del genoma del fago se conoce desde antes de empezar el proceso de ensamble. Por este motivo, se pueden tomar solo las primeras 56232 bases. De forma alternativa, se puede construir la gráfica de De Bruijn usando el conjunto de *k*-meros únicos. Como se observa en la tabla 2, usar el conjunto de *k*-meros únicos no ayuda a obtener secuencias que se alinean mejor con el genoma del fago que usando todos los *k*-meros para los datos crudos ni de calidad. En el caso de los datos corregidos con ambos algoritmos de corrección de errores se observa que el contig obtenido es ligeramente menor en tamaño pero en general no cambia mucho el resultado del ensamble, posiblemente por que el algoritmo está encontrando un camino en un componente disconexo de la gráfica, como se mencionó anteriormente. En el caso donde usar los k-meros únicos sí ayuda es cuando se tienen lecturas completamente libres de errores. En la última entrada de la tabla 2 se observa que se obtuvo un contig casi del tamaño exacto del genoma del fago y se alinea perfectamente en todas excepto que repite 2 bases y le faltan 6 bases. Estos errores se pueden corregir en las siguientes etapas del ensamble o incluso usando la secuencia obtenida en un ensamble por genoma de referencia.

Aunque el genoma del fago Bacata casi no tiene repeticiones, las secuencias repetidas son comunes en los genomas de otros organismos. Los genomas de plantas y animales usualmente contienen una gran cantidad de secuencias repetidas, distribuidas a través de todo el genoma y compuestas de transposones, microsatélites y largas duplicaciones segmentales. Genomas en los que más del 30 % son secuencias repetidas incluyen el gusano de seda, panda, arroz y pepino, entre otros (Li, Z. *et al.*, 2011).

Las secuencias repetidas presentan un desafío serio para DBG, aún en el caso de datos libres de errores (Pevzner, P.A., Tang, H. & Waterman, M.S., 2001). Las secuencias repetidas causan ambigüedades en el orden de los segmentos del genoma. Su localización y distribución afectan la calidad del ensamble. Muchas secuencias repetidas localizadas en una región pequeña hacen esa región difícil de ensamblar, pero no afectan la organización global del genoma. Por otro lado, las secuencias repetidas esparcidas por todo el genoma lo dividen en muchos bloques pequeños cuyo orden no se puede determinar (Kingsford, C., Schatz, M.C. & Pop, M., 2010).

El principal problema de las secuencias repetidas en DBG es que no se puede distinguir cuándo la multiplicidad de las aristas se debe a múltiples lecturas sobre la misma región y cuando se debe a las repeticiones, como se muestra en la figura 4. En principio, la capacidad de resolver secuencias repetidas es mayor cuanto más grande sea *k*, pero *k* no puede ser mayor al tamaño de las lecturas.

Las secuencias repetidas crean problemas al ensamblar fragmentos porque hay varias aristas que entran a la secuencia repetida y varias aristas que salen pero no es claro qué

salida debe ser visitada después de qué entrada en el camino euleriano (Pevzner, P.A., Tang, H. & Waterman, M.S., 2001).

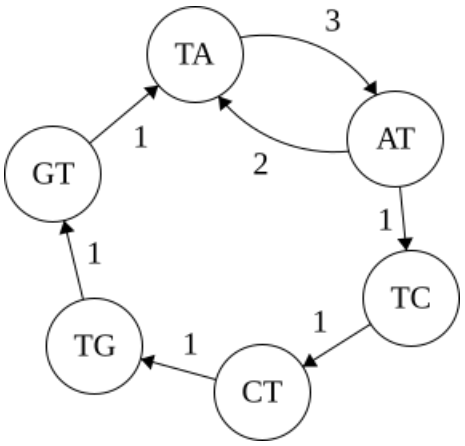


Figura 4. Gráfica de De Bruijn construida a partir de las lecturas {GTA,TAT,ATA,TAT,ATA,TAT,ATC,TCT,CTG,TGT} con *k* = 3. El ciclo euleriano corresponde al genoma GTATATATCT. El genoma original era GTATATCT, pero se obtuvo una lectura TAT y ATA adicionales durante la secuenciación.

Los resultados del alineamiento del ensamble del genoma sintético, para los diferentes valores de *k*, se muestran a continuación. Al usar el vértice con las primeras *k*-1 bases para iniciar el algoritmo de Hierholzer, en todos los casos, la mejor alineación con el genoma sintético es la región completa a partir de la posición 1.

k	Tamaño del ensamble (bp)	Coincidencias	Espacios
20	18616	16754/18631 (89.9254 %)	30/18631 (0.1610 %)
25	34703	23283/26146 (89.0500 %)	72/26146 (0.2754 %)
30	50628	23982/26126 (91.7936 %)	2/26126 (0.0077 %)
35	16271	14680/16277 (90.1886 %)	12/16277 (0.0737 %)
40	45077	23670/26133 (90.5751 %)	10/26133 (0.0383 %)
45	16094	14651/16095 (91.0282 %)	2/16095 (0.0124 %)
50	25632	22703/25649	34/25649

		(88.5142 %)	(0.1326 %)

Tabla 2. Por cada valor de *k*, se muestra el tamaño completo de la secuencia ensamblada por el algoritmo, la cantidad de bases que coinciden y los espacios en la mejor región alineada con el genoma sintético.

Como se observa en la tabla 2, para todos los tamaños comunes de k, se ensambla un contig de al menos 16000 bp, sin embargo, la secuencia ensamblada tiene una gran cantidad de bases que no coinciden con el genoma sintético.. En todos los casos, el porcentaje de coincidencia es de alrededor del 90 %, aunque hay pocos espacios. Esto se debe a que la longitud de las regiones con secuencias repetidas en el genoma sintético tienen un tamaño mayor a todos los valores de *k* usados. El algoritmo de Hierholzer no puede distinguir a qué parte del camino euleriano pertenecen las aristas múltiples y, al tomar la primera arista que encuentra, la tasa de error es elevada.

Diferentes métodos se han propuesto para resolver el problema de las secuencias repetidas cuando son más grandes que los valores de *k* usados. Una alternativa es asignar manualmente la multiplicidad de las aristas cuando la cantidad de veces que se repiten las secuencias en el genoma es conocida (Compeau, P.E., Pevzner, P.A. & Tesler, G., 2011). Otras alternativas usan los caminos definidos por los *k*-meros de las lecturas para obtener información acerca de las aristas que se debe seguir durante la construcción del camino.

Sin embargo, una ventaja que tiene DBG sobre OLC es que en casos donde hay muchas secuencias repetidas, aumenta el costo en tiempo y espacio de OLC, mientras que en DBG se mantiene igual. Las secuencias repetidas aumentan el tiempo de cómputo requerido para los alineamientos por pares en OLC porque las lecturas de la región con secuencias repetidas tienen muchos traslapes entre sí. Adicionalmente, es muy intensivo en el uso de memoria almacenar la relación de tales traslapes (Li, Z. *et al.*, 2011). En DBG los *k*-meros y *k*-1-meros repetidos se colapsan en aristas donde la multiplicidad se puede almacenar como un entero y en nodos únicos en la gráfica.

Conclusiones

Ventajas de DBG:

- Es escalable. La complejidad en tiempo es O(*NL*) y en espacio es O(min(*NL*,*G*)).
- Transforma el problema NP-completo de camino hamiltoniano en un problema de camino euleriano con solución en tiempo lineal.
- La implementación del algoritmo es muy simple.
- Cuando hay pocos *k*-meros repetidos en el genoma, se puede ensamblar casi todo el genoma sin necesidad de considerar un problema de supercamino.
- Las secuencias repetidas no significan un aumento en el costo computacional, como en OLC.

Desventajas de DBG:

- No es tolerante a errores de secuenciación. Requiere un algoritmo especializado de corrección de errores como parte del preprocesamiento.

- Los errores y la cobertura de las lecturas pueden hacer que la gráfica sea desconexa y no euleriana.
- Aún si la gráfica es euleriana, hay múltiples caminos eulerianos que corresponden a diferentes genomas. Distinguir el camino correcto es un problema de supercamino y es NP-completo.
- Si hay regiones grandes con secuencias repetidas se producen errores en la secuencia ensamblada a menos que se usen métodos auxiliares para resolver las repeticiones.

Referencias

- Shendure, J. & Ji, H. (2008) “Next-generation DNA sequencing,” *Nature Biotechnology*, 26(10), pp. 1135–1145. https://doi.org/10.1038/nbt1486.
- Lischer, H.E. & Shimizu, K.K. (2017) “Reference-guided de novo assembly approach improves genome reconstruction for related species,” *BMC Bioinformatics*, 18(1). https://doi.org/10.1186/s12859-017-1911-6.
- Li, Z. *et al.* (2011) “Comparison of the two major classes of assembly algorithms: Overlap-layout-consensus and de-bruijn-graph,” *Briefings in Functional Genomics*, 11(1), pp. 25–37. https://doi.org/10.1093/bfgp/elr035.
- Mahadik, K., Wright, C., Kulkarni, M., Bagchi, S., & Chaterji, S. (2019). Scalable genome assembly through parallel de Bruijn graph construction for multiple K-MERS. *Scientific Reports*, *9*(1). https://doi.org/10.1038/s41598-019-51284-9
- Clavijo-Coppens, F. *et al.* (2021) “Novel virulent bacteriophages infecting Mediterranean isolates of the plant pest xylella fastidiosa and Xanthomonas Albilineans,” *Viruses*, 13(5), p. 725. https://doi.org/10.3390/v13050725.
- Pevzner, P.A., Tang, H. & Waterman, M.S. (2001) “An Eulerian path approach to DNA Fragment assembly,” *Proceedings of the National Academy of Sciences*, 98(17), pp. 9748–9753. https://doi.org/10.1073/pnas.171285098.
- Kingsford, C., Schatz, M.C. & Pop, M. (2010) “Assembly complexity of prokaryotic genomes using short reads,” *BMC Bioinformatics*, 11(1). https://doi.org/10.1186/1471-2105-11-21.
- Langmead, B. (2018) “Assembly in Practice: Part 1: OLC” . Recuperado el 13 de diciembre de 2022 de: https://www.cs.jhu.edu/~langmea/resources/lecture_notes/18_assembly_olc_v2.pdf
- Narzisi, G., Mishra, B. & Schatz, M.C. (2014) “On algorithmic complexity of Biomolecular Sequence Assembly problem,” *Algorithms for Computational Biology*, pp. 183–195. https://doi.org/10.1007/978-3-319-07953-0_15.
- Compeau, P.E., Pevzner, P.A. & Tesler, G. (2011) “How to apply de Bruijn graphs to genome assembly,” *Nature Biotechnology*, 29(11), pp. 987–991. https://doi.org/10.1038/nbt.2023.