UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Análisis topológico de De Bruijn Graphs generados a partir de datos metagenómicos

TRABAJO DE TITULACIÓN

AUTOR: Ojeda Poma, Freddy Fernando

DIRECTOR: Riofrío Calderón, Guido Eduardo, Ing.

CO-DIRECTOR: Puchaicela Huaca, Luis Patricio, Ing.

LOJA-ECUADOR

2016

Septiembre, 2016

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Ing.
Guido Eduardo Riofrío Calderón
DOCENTE DE TITULACIÓN DE INGENIERÍA EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

De mi consideración:

El presente trabajo de titulación: **Análisis topológico de De Bruijn Graphs generados a partir de datos metagenómicos** realizado por **Ojeda Poma Freddy Fernando**, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, diciembre de 2015

f) . . . . . .

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

Yo **Ojeda Poma Freddy Fernando** declaro ser autor del presente trabajo de titulación: Análisis topológico de De Bruijn Graphs generados a partir de datos metagenómicos de la Titulación de Sistemas Informáticos y Computación, siendo Guido Eduardo Riofrío Calderón director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad. Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad.

f.....
Ojeda Poma Freddy Fernando
1104111578

DEDICATORIA

A mis padres Myrian y Domingo por creer en mí, por brindarme su cariño y apoyo incondicional, a mis hermanos Lourdes y Santiago por estar siempre presentes acompañándome para poderme realizar.

A mi esposa Patricia y mi hija Tatiana que son mi impulso para salir adelante.

Freddy Fernando Ojeda Poma

AGRADECIMIENTO

Mis más sinceros agradecimientos a todas las personas que me apoyaron y de una u otra manera han contribuido a la culminación de este proyecto de fin de titulación.

A los docentes de la UTPL; Al Ing. Guido Riofrío por ser mi director y guía, al Ing. Patricio Puchaicela por la colaboración brindada durante la realización del proyecto y al Ph.D. Aminael Sánchez que con sus conocimientos, empeño y motivación impulsaron la realización de este proyecto.

Gracias

ÍNDICE DE CONTENIDOS

Índice de figuras

Índice de cuadros

RESUMEN

El estudio de la metagenómica con el uso de las tecnologías de secuenciación de nueva generación (NGS) han permitido la secuenciación de un gran número de genomas en muy poco tiempo La cantidad de información es tan grande que cada vez se usa con más frecuencia el término Big data para referirnos a la información que está en bases de datos lista para ser analizada.

Existen varios problemas para que estos datos sean analizados a un ritmo adecuado, se presentan retos computacionales, falta de programas de análisis y tal vez lo más importante la falta de formación de personal en el análisis e interpretación de datos genómicos.

El uso de los De Bruijn Graphs para el ensamblaje genómico nos permitirá analizar las propiedades topológicas de los De Bruijn Graphs en dependencia de las relaciones filogenéticas de las especies microbianas presentes en muestras metagenómicas del suelo.

PALABRAS CLAVE: Metagenómica, secuenciación, genomas, Bruijn Graph, filogenética.

Parte I

INTRODUCCIÓN

En las últimas décadas se ha verificado un acelerado avance tecnológico en las plataformas de secuenciación masiva de ADN. Ello ha dado lugar a las llamadas Tecnologías de Nueva Generación (NGS –por sus siglas en inglés) dentro de las que destacan Illumina, Roche 454 y Pacific Biosystems (Lin Liu, 2012). Con el surgimientos de las NGS, es ahora posible movernos desde la perspectiva del estudio de un solo genoma a la vez, hacia el estudio de comunidades de especies lo que se conoce como metagenómica (Restrepo Restrepo, González, & Cárdenas, 2012). A pesar de que contamos ahora con la tecnología para ello, aún no se han desarrollado a cabalidad las herramientas computacionales que permitan la reconstrucción de genomas individuales a partir de muestras metagenómicas. (Bonilla-Rosso, Souza, & Eguiartea, 2008) El proceso de reconstrucción de genomas individuales se conoce como "ensamblaje genómico".

Una de las estrategias pioneras de ensamblaje genómico, es la que se basa en la construcción de una estructura de datos conocida como De Bruijn Graphs (Daniel R. Zerbino, 2008). La utilidad de los De Bruijn Graphs ha sido extensamente demostrada para el ensamblaje de genomas simples, siendo Velvet (Zerbino, 2011) la mejor implementación de De Bruijn Graphs reportada hasta el momento.

Los esfuerzos actuales se están dirigiendo entonces hacia demostrar la aplicabilidad de los De Bruijn Graphs en la reconstrucción de genomas simples así como también de metagenomas. Es en este escenario temático que se desarrollará el presente proyecto donde evaluaremos como varían las propiedades topológicas de los De Bruijn Graphs en dependencia de las relaciones filogenéticas de las especies microbianas presentes en muestras metagenómicas del suelo ya que es este probablemente uno de los ambientes más complejos por su diversidad y complejidad microbiológica (Hernández-León R, 2010).

Parte II

PLANTEAMIENTO DEL PROBLEMA

La metagenómica, una joven disciplina en auge nos permite extraer el ADN característico de una comunidad sin la necesidad de pasar por el cultivo in-vitro de las especies constituyentes. Una vez extraído el ADN que ahora contiene tanto a especies mayoritarias como minoritarias y cultivables como no cultivables, este puede ser secuenciado parcialmente o en su totalidad. La información de secuencia que se obtiene mediante esta metodología constituye el metagenoma de la comunidad en estudio.

Si bien es cierto en la actualidad se cuenta con la tecnología para la secuenciación de metagenomas, estamos aún lejos de poder analizar a cabalidad el alto volumen de datos derivados de esta secuenciación. Tal análisis, cuyo objetivo es la reconstrucción de genomas individuales a partir de una mezcla de millones de fragmentos de secuencia, presenta grandes retos computacionales y el diseño de nuevos algoritmos y modelos, para esta tarea es un campo de investigación muy fértil.

Contar con métodos capaces de reconstruir genomas individuales a partir de la información del metagenoma nos permitirá no solo estudiar la estructura sino también hacer inferencias sobre la capacidad funcional de una comunidad microbiana.

Para dar solución al problema se seguirá las siguientes fases:

- *Selección de especies genómicas para la simulación._* Son especies que se encuentren en el suelo, las mismas tienen una estrecha relación filogenética Se trabaja con mezclas de 2, 3, 6 y 12 genomas para las simulaciones
- *Estudio del funcionamiento de ensamblador Velvet._* Análisis de las funciones principales del ensamblador.
- *Modificación de Velvet (VelvetLite)._* Creación de una versión de Velvet modificada para que se adapte a nuestros requerimientos.
- Generación de datos con la utilización de VelvetLite._ Se ejecuta el ensamblador modificado.
- Análisis de los datos generados. - Se extrae las propiedades topológicas de los grafos de De Bruijn y se las almacena para su posterior análisis.

Parte III

OBJETIVOS

Objetivo general

- Estudiar las propiedades topológicas de los De Bruijn Graphs y su aplicabilidad en la reconstrucción de genomas individuales a partir de datos de secuenciación metagenómica.

Objetivos específicos

- Generar conjuntos de datos simulados de secuenciación metagenómica que contengan diferente grado de complejidad en cuanto al número de secuencias consideradas y las distancias filogenéticas entre las mismas.
- Desarrollar un conjunto de scripts que permitan procesar la gran cantidad de datos generados por el ensamblador; y así extraer las propiedades de los grafos de De Bruijn generados.

Parte IV
ESTRUCTURA DE LA TESIS

La tesis está estructurada de la siguiente manera:
Una introducción general sobre lo que es el proyecto, el trabajo realizado, los objetivos trazados y lo que se pretende conseguir a la culminación del mismo.

En el capítulo 1, titulado: Secuenciación de Genomas, se trata acerca de la revisión del estado del arte, conceptos, definiciones, funcionamiento, etc., acerca de los métodos de secuenciación genética de nueva generación. También se trata términos utilizados en la metagenómica, principales métodos de ensamblaje, ensambladores más utilizados. Las propiedades de los grafos de De Bruijn ¿cómo se generan y cómo representarlos gráficamente?

En el capítulo 2, titulado: Simulación de datos metagenómicos, ya se trabaja seleccionando los genomas para el proyecto y la utilización del ensamblador Velvet, se realiza el análisis y modificación de Velvet dando origen a VelvetLite el cual se adapta a las nuestras necesidades y se realiza la generación de los De Bruijn graphs resultantes de la ejecución del ensamblador.

En el capítulo 3, titulado: Análisis de grafo de De Bruijn, se trabaja con los archivos generados por el ensamblador Velvet. A estos archivos se los analiza y se les extrae los datos necesarios para el análisis topológico. Se almacenan los datos extraídos en una base de datos para en esta realizar las consultas referentes a las propiedades del grafo.

En el capítulo 4 titulado: Análisis de resultados, se realiza las comparaciones entre los datos obtenidos de la ejecución de Velvet vs VelvetLite que es la versión modificada del ensamblador Velvet

CAPÍTULO 1

1. Secuenciación de genomas

Un genoma es una colección completa de ácido desoxirribonucleico (ADN) de un organismo, o sea un compuesto químico que contiene las instrucciones genéticas necesarias para desarrollar y dirigir las actividades de todo organismo. Las moléculas del ADN están conformadas por dos hélices torcidas y emparejadas. Cada hélice está formada por cuatro unidades químicas, denominadas bases nucleótidas. Las bases son adenina (A), timina (T), guanina (G) y citosina (C). (National Human Genome Research Institute, 2011)

La secuenciación del ADN es un conjunto de métodos y técnicas cuya finalidad es determinar el orden exacto de los pares de bases en un segmento de ADN.

La secuenciación de genomas permite secuenciar cadenas cortas de ADN, para lo cual necesitamos una estrategia para fragmentar un genoma y luego re-ensamblar el genoma secuenciado

La figura 1 nos muestra una visión general del proceso de secuenciación de un genoma

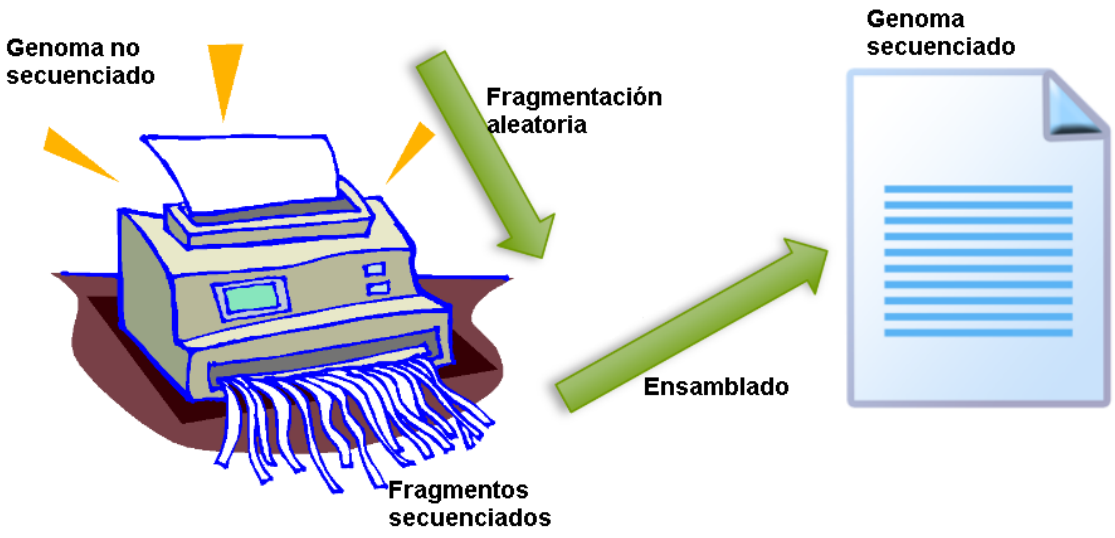


Figura 1: Proceso de secuenciación de un genoma

Fuente: **(Santamaría, 2011)**

En el estudio de la secuenciación de genomas nos encontramos con ciertos términos, los cuales se describen a continuación:

Fragmento.- Secuencia contigua de ADN, sin huecos

Contig._ Conjunto se fragmentos solapantes de los que se puede obtener una secuencia más larga.

Scaffold._ Serie de contigs ordenados, pero no necesariamente conectados en una secuencia sin huecos

Covertura._ Número medio de lecturas de cada nucleótido en la secuencia reconstruida

1.1. Herramientas de secuenciación genómica de Nueva Generación

Los avances en materia de secuenciación del ADN han acelerado la investigación y los descubrimientos en biología. Las técnicas modernas nos dan la posibilidad de realizar esta secuenciación a gran velocidad, lo cual ha sido muy importante a la hora de trabajar en proyectos a gran escala como el Proyecto Genoma Humano. Otros proyectos a escala mundial, en ocasiones resultado de la cooperación investigadora de varias instituciones, han establecido la secuencia completa de ADN de muchos genomas de animales, plantas y microorganismos.

En los últimos años los procedimientos de secuenciación de nueva generación-NGS están siendo utilizados para la secuenciación de genes de una forma más barata y eficiente, a pesar de obtener fragmentos-reads de menor tamaño. Estas secuencias aisladas obtenidas requieren el empleo de potentes herramientas informáticas para su ensamblaje.

Actualmente, los sistemas de secuenciación de nueva generación disponibles en el mercado son el método 454 de Roche, Solexa de Illumina y SOLiD de Applied Biosystems.

Secuenciación 454

La pirosecuenciación o secuenciación 454, es una tecnología de determinación de secuencia de ADN a gran escala, aplicable a genomas completos, mediante luminiscencia. El método 454 puede determinar la secuencia de 20 millones en 4 horas, lo cual abarata enormemente el coste del proceso. (Universidad de Málaga, Moreno, Rocío Bautista, 2010)

Este fue el primer método de nueva generación que salió al mercado y en la actualidad es el que más publicaciones tiene a su haber. Esta técnica se conoce desde hace algunos años, pero gracias al impulso dado al desarrollo de nuevas tecnologías de secuenciación más eficientes y baratas se han conseguido una serie de mejoras que han hecho posible la aplicación de esta técnica para la secuenciación a una escala genómica y con bajo coste.

Las muestras primero son reducidas a una librería de fragmentos de 300-800pb de longitud. Los fragmentos se unen a nano-esferas y la secuenciación se basa en la síntesis de la cadena complementaria gracias a la DNA polimerasa. El proceso de amplificación (realizar copias del mismo fragmento) se realiza en las propias nano-esferas (una especie de nano-PCR). La posterior identificación de las secuencias se realiza por un sistema basado en la piro-secuenciación, adecuado para secuenciación de novo, re-secuenciación de genomas completos o de regiones diana, estudio de variantes de virus, detección de puntos con polimorfismos genéticos y análisis de ARN.

Solexa

La técnica de secuenciación de Solexa consta de dos partes, en primer lugar se realiza un fraccionamiento del genoma, seguido de la amplificación clonal de fragmentos de ADN sobre la superficie sólida de una celda de flujo. De esta forma se crean racimos de fragmentos clonados. El siguiente paso consiste en la secuenciación mediante síntesis. Lo característico de este sistema es el empleo de los cuatro nucleótidos (A, C, T, G) marcados con fluorescencia de distintos colores para la secuenciación de los millones de racimos presentes en la superficie de la celda de flujo. Estos nucleótidos modificados contienen terminaciones reversibles, lo que permite que cada ciclo del proceso de secuenciación ocurra en presencia de los cuatro nucleótidos simultáneamente. Es un procedimiento más barato y potente que el 454 pero al producir lecturas de pequeño tamaño no puede resolver repeticiones cortas.

SOLiD

Se trata del último método de secuenciación de los analizados en salir al mercado, ya que se empezó a comercializar en 2008. Permite el empleo de distintos tipos de muestras ya sean fragmentos sueltos o grupos de dos fragmentos unidos mediante un cebador. El proceso de amplificación es similar al de 454 a excepción del anclaje final de las nano-esferas a una superficie de cristal donde se da la secuenciación. El proceso de secuenciación es realmente innovador, dándose distintas rondas de hibridación y ligado con 16 dinucleótidos marcados con 4 colores distintos. Empleando un código de colores, cada posición es evaluada dos veces lo que aumenta drásticamente la discriminación entre errores de secuencia y polimorfismos de un solo nucleótido.

Cuadro 1: Comparativo de los métodos de secuenciación de ADN

Tecnología	**454**	**Solexa**	**SOLiD**
Secuenciador	**GS FLX serie Titanium**	**HiSeq 2000**	**SOLiD™ 4 System**
Potencia por análisis	500 · 106 pb	200. 109 pb	100. 109 pb
Tiempo de ejecución	10 horas	8 días	14 días
Longitud de los fragmentos (reads)	450 pb	2 · 100 pb	2 · 50 pb
Fragmentos por ejecución	>106	2 · 109	1,4 · 109
Precio por secuenciar un genoma humano	$180.000 (USD)	< $10.000 (USD)	$6.000 (USD)

Fuente: (Ainhoa Ogiza, 2010)

1.2. Estudio de metagenomas

Metagenómica

La metagenómica es el estudio del conjunto de genomas de un determinado entorno directamente a partir de muestras de ese ambiente, sin necesidad de aislar y cultivar esas especies.

La metagenómica es una de las nuevas aplicaciones que han sido posibles con la aparición de las tecnologías para secuenciar el ADN a bajo costo. Estas nuevas tecnologías permiten reducir el costo de secuenciar el ADN a costos mucho menores que aquellos usando la tecnología basada en secuenciamiento.

Secuenciación

La recuperación de secuencias de ADN mayores de unas pocas miles de pares de bases era muy difícil hasta la reciente llegada de avanzadas técnicas de Biología Molecular que permiten la construcción de genotecas de cromosomas artificiales bacterianos

Metagenómica balística

Los avances en Bioinformática, las mejoras en la amplificación del ADN, y el aumento de la capacidad de cómputo de los sistemas informáticos actuales han ayudado, en conjunto, al análisis de las secuencias de ADN obtenidas a partir de muestras ambientales, permitiendo la aplicación de la secuenciación balística a muestras metagenómicas. El abordaje balístico sigue una serie de pasos básicos: fragmentación del ADN al azar en secuencias muy pequeñas, y ensamblaje por mapeo contra una secuencia consenso.

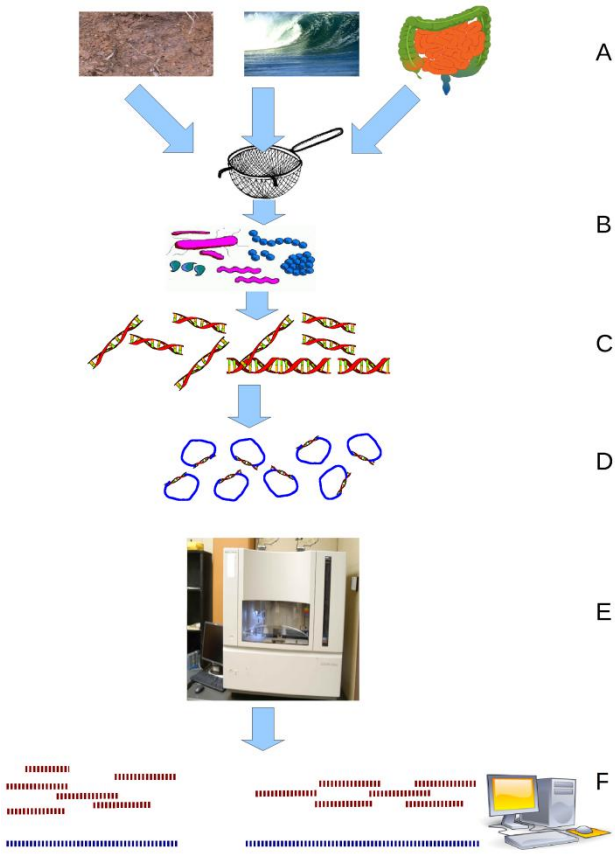


Figura 2: Secuenciación ambiental balística

Fuente: (John C. Wooley, 2011)

(A) Recogida de muestras ambientales;

(B) Filtrado de partículas, comúnmente por tamaño;

(C) Lisis y extracción del ADN;

(D) Clonación y construcción de la genoteca;

(E) Secuenciación de los clones;

(F) Ensamblaje de las secuencias en contigs y scaffolds.

La metagenómica basada en balística permite la secuenciación directa de genomas microbianos casi completos, a partir de muestras ambientales. En cada muestra ambiental existen una gran variedad de microorganismos en unas proporciones determinadas. Los organismos más abundantes estarán más representados en la

información de secuencia obtenida. Se suelen requerir grandes muestras para asegurar una cobertura suficiente para resolver por completo los genomas de especies poco representadas en la comunidad. Por otra parte, la naturaleza aleatoria de la secuenciación por balística asegura que muchos de estos organismos, que pasarían desapercibidos si se usaran técnicas culturales clásicas, estarán representados al menos por pequeños fragmentos de secuencia.

1.3. Formatos para la representación de secuencias de genomas

En bioinformática es necesario representar la información de las secuencias de genes de manera digital para poder trabajar con ella. En la actualidad existen muchos formatos utilizados para este fin.

Los principales formatos son: FASTA, FASTQ, GenBank, EMBL. Aunque los que más se usan son los dos primeros en la mayoría de experimentos con ensambladores y es precisamente a estos a los que estudiaremos para su uso en el presente proyecto

FASTA

Una secuencia representada en formato FASTA siempre tiene la misma estructura:

En primer lugar empieza con una descripción en una única línea, comúnmente línea de cabecera. Esta línea de descripción se distingue debido a que comienza con el símbolo **“>”** seguido del identificador de secuencia y de la descripción (ambos son opcionales). Es importante que no haya espacio entre el símbolo **“>”** y la primera letra del identificador de secuencia.

Para saber representar varias secuencias en un mismo fichero de formato fasta hay que seguir las indicaciones anteriores para todas las secuencias, ya que el símbolo “**>**” se utilizará como separador de secuencias.

Se puede considerar como el formato más sencillo y a la vez el más utilizado en bioinformática, con este formato trabajaremos en el presente proyecto.

```
>gi|255961261|ref|NC_007492.2| Pseudomonas fluorescens Pf0-1 chromosome
TGTTACCTGGTTCGTCCACAACGGGCCGGAATGGCCCCCGTTTTAAGAGACCGGGGATTCTAGAGAAAGC
AAGCCAATAGGTCAATTTCCAACCAACGTTTCCTTATAAATAGATCTTCGGAGCTGTCGGGACCTGATTA
CCCGTTTGCCAAAGATAAAAAATAAAGAAGGGAATTATTTAAAGCTTTTCTGTAAAGCTTGTAAAAGCTA
GGAACGCCACTGTCTGTGGATAACTGGCTCAAGGCCTTGTTTCAAGCGATGTACAGAGAATGACAACTAC
AGTGGAAAACCGTGTTCAGCCTGTGCTGCGCTGTCGGATAACCTGTGTGTGGAATAGGTTGTTATCCACA
```

Figura 3: Secuencia en formato FASTA generada

Fuente: Elaboración propia con datos obtenidos de archivo fasta generado

FASTQ

Es un formato de texto que permite almacenar la secuencia biológica y sus puntuaciones de calidad correspondientes.

Tanto la secuencia biológica como la secuencia de calidad están representadas con un solo carácter ASCII para mayor simplicidad.

Originalmente fue desarrollado en el “Welcome Trust Sanger Institute” para agrupar secuencias en formato FASTA y su calidad asociada en un solo fichero con un formato definido. Actualmente es uno de los formatos más utilizados en ensambladores, junto con el formato FASTA.

El formato normalmente utiliza cuatro líneas para representar una secuencia concreta:

Línea 1: Empieza con el carácter ‘@’ seguido de una secuencia de identificación más una breve descripción opcional. Esta línea es análoga a la línea de descripción del formato FASTA.

Línea 2: Es la codificación completa de la secuencia.

Línea 3: Empieza con el carácter ‘+’ y opcionalmente va seguido del mismo identificador y de la misma descripción de la línea 1.

Línea 4: Codifica los valores de calidad de la secuencia representada en la línea 2 por lo que debe de contener el mismo número de símbolos que dicha línea. Dichos valores de calidad son representados por una codificación ASCII de los valores numéricos asociados (por ese motivo pueden aparecer caracteres que se podrían considerar extraños)

En un principio en el formato FASTQ se permitía que la secuencia y las secuencias de calidad pudieran ser divididas en varias líneas. Pero esto se descartó debido a que hacía el análisis muy complicado porque los caracteres '@' y '+' podían aparecer en la secuencia de calidad y podía haber confusiones al realizar el procesamiento del fichero.

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

Figura 4: Secuencia en formato FASTQ generada

Fuente: Elaboración propia con datos obtenidos de archivo fasq generado

1.4. Ensamblaje de Genomas

Todas las tecnologías de secuenciación producen fragmentos más o menos cortos y a partir de estos se debe de recomponer la secuencia del fragmento analizado. En muchos de los casos el resultado no será la secuencia completa del fragmento y será necesario completar los huecos entre los conjuntos de secuencias. Estos procesos se denominan proyectos de secuenciación. (Cañizares Sales & Forment Millet, 2014)

1.4.1. Ensamblaje de secuencias mediante alineamiento

El proceso de ensamblaje consta de tres fases: limpieza de secuencias, alineamiento de secuencias y ensamblaje.

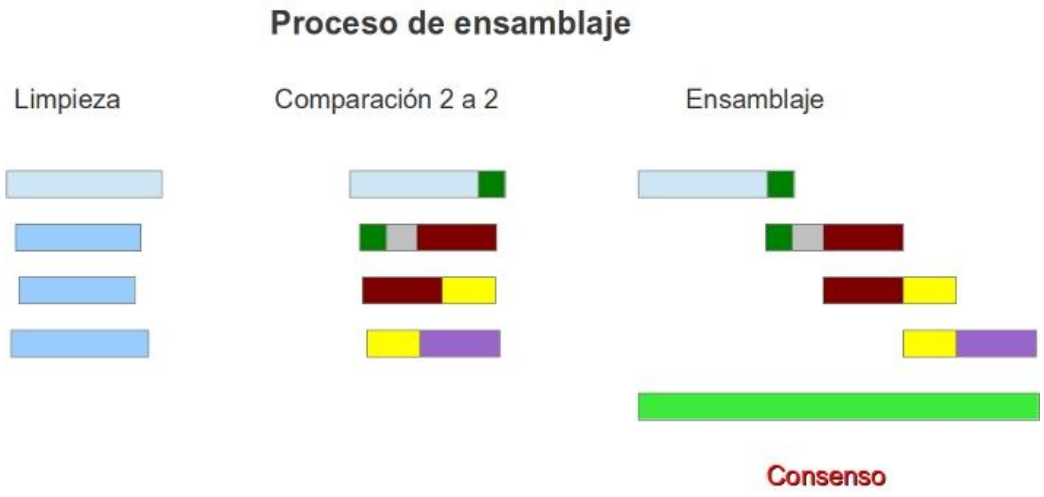


Figura 5: Ensamblaje de secuencias mediante alineamiento

Fuente: (Joaquin Cañizares)

Limpieza de secuencias:

En la fase de limpieza de secuencias se elimina las regiones de las secuencias que tengan resto de vector y las regiones repetitivas.

Alineamiento de secuencias

Ahora toca alinear todas las secuencias con todas las secuencias dos a dos para identificar que secuencias se solapan entre sí. Para realizar esto debemos identificar pares o grupos de secuencias con fragmentos de secuencias comunes. Primero se realiza una comparación rápida para identificar que secuencias comparten palabras o ventanas iguales. En un segundo paso estas parejas de secuencias se alinean mediante programas de alineación dinámica. Esto nos da como resultado una red de interacciones entre las secuencias que alinean entre si, en el que en cada borde está localizado una secuencia.

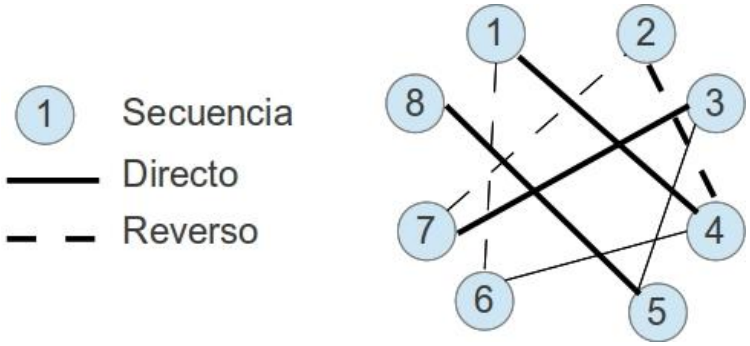


Figura 6: Alineamiento de secuencias.

Fuente: (Joaquin Cañizares)

Ensamblaje:

Ya con estos grafos de las secuencias identificadas que comparten regiones comunes, se crea el ensamblaje. A diferencia de un alineamiento múltiple los ensambladores no fuerzan que todas las secuencias cubran la totalidad del alineamiento, este se va alargando con la adición de nuevas secuencias. Existen diferentes algoritmos que a partir de la información almacenada en el grafo ordenan y ensamblan las secuencias utilizando alineación múltiple para obtener la mejor distribución de las secuencias solapantes. En muchos programas estos dos últimos pasos (alineamientos de secuencias y ensamblaje) son cíclicos. A partir de estos alineamientos se obtienen la secuencia consenso que representa a cada conjunto de secuencias o contig. Existen diferentes reglas para construir el consenso, pero es indispensable que tengan en cuenta el valor de la calidad de las secuencias.

1.4.2. Ensamblaje de secuencias mediante k-mers

A la hora de trabajar con genomas que demanden gran cantidad de cómputo ya que utilizan centenares de millones de secuencias, se ha desarrollado algoritmos más rápidos. Los más utilizados se basan en el uso de k_mers.

Los k-mers son una serie de bases de tamaño determinado (words, ventana) y con ellos podemos dibujar gráficos de sus relaciones o como se los conoce los Grafos de De Bruijn. Los nodos son todos los K-mers y las flechas los solapamientos entre ellos.

Podemos ensamblar las secuencias resolviendo estos grafos. El problema es que son mucho más sensibles a los errores de secuenciación y repetitivo que los métodos basados en alineamiento, haciendo que la complejidad de los grafos aumente; pero tienen la ventaja de ser mucho más rápidos.

Los programas o ensambladores más utilizados con esta metodología son el SOAPdenovo y el Velvet, en el estudio de este último se basa este proyecto.

1.5. Grafos

Un grafo es una representación de las interacciones que tienen lugar entre las entidades de un sistema que dan lugar a un fenómeno estudiado.

Los nodos representan las entidades genéricas que constituyen el sistema (genes, proteínas, metabolitos, etc). Las aristas entre distintos nodos indican que las correspondientes entidades interactúan o están relacionadas entre sí de alguna forma.

Formalmente, una red o grafo G no dirigido es un par de conjuntos **G=(V,E)** donde:

V={v1,v2,....,vn} es el conjunto de vértices o nodos.

E={(vi,vj),(v k,v l),...} es un conjunto de pares no ordenados de elementos de V.

E se denomina conjunto de *aristas* de la red.

El número de *nodos* se denomina ***orden de la red*.**

El número de *aristas* se denomina ***tamaño de la red*.**

Un camino de un nodo **u** a un nodo **v** es una secuencia de nodos {v1,v2...,vk} con v1=u, vk=v y {vi-1,vi} una arista de la red tal que en {v1,v2,..,vk} no hay nodos repetidos. Un camino mínimo entre dos nodos es aquel de menor longitud de entre todos los posibles caminos entre ambos nodos.

Análisis de la topología de una red-información general

El análisis de las características topológicas de redes nos permite estudiar propiedades relevantes sobre las dinámicas que pueden producirse tales como la transmisión de información a lo largo de la red y robustez en su funcionamiento.

Dos nodos de una red son vecinos o adyacentes si existe una arista que los conecta.

El grado de un nodo es el número de vecinos que tiene dicho nodo.

La distribución del grado de nodos de una red **G=(V,E)** se define como:

P(k)= mk m

donde mk es el número de nodos con grado k y m es el orden de G número de nodos de la red.

Distancia

En teoría de grafos se denomina distancia entre dos vértices de un grafo al número de vértices mínimo que debe recorrerse para unirlos. La distancia entre dos nodos de un grafo es la longitud del camino más corto (a veces se denomina geodésico1). Si no hubiera conexión alguna entre dos vértices se dice que la distancia es infinita. Las distancias de todos los vértices de un grafo se computan en lo que se denomina matriz de distancias, el concepto se emplea en las medidas de centralidad de redes.

Densidad del grafo

Es la relación entre el número de aristas del grafo y el número de aristas máximo, el cual es el número de aristas de un grafo completo con el mismo número de nodos.

Se denomina grafo disperso a aquel cuya densidad tiene un valor pequeño, mientras que los grafos densos son aquellos que poseen gran número de aristas.

Métricas de centralidad

Centralidad de grado

La centralidad de grado es una medida radial de volumen que se calcula como el número de aristas incidentes en el vértice. Esta medida da una idea de la conectividad de un nodo.

Centralidad de cercanía:

La centralidad de cercanía es una métrica radial de longitud calculada como la inversa del valor medio de las distancias mínimas desde ese nodo al resto de nodos del grafo. Por tanto, cuanto mayor sea la distancia media menor será la centralidad del nodo y viceversa.

Centralidad de intermediación:

La centralidad de intermediación es una medida de centralidad medial que viene dada por la fracción de caminos más cortos entre nodos del grafo que atraviesan el nodo en cuestión. Se calcula mediante la siguiente ecuación:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

donde V es el conjunto de nodos del grafo, $\sigma(s,t)$ es el número de caminos más cortos existentes en el grafo entre los nodos y $\sigma(s,t|v)$ es la cantidad de esos caminos en los que V actúa de puente.

Centralidad de vector propio:

Otra medida de centralidad radial de volumen es la centralidad de vector propio. Esta métrica mide la influencia de un nodo en el grafo de modo que esta centralidad será elevada si un nodo está conectado a muchos nodos, los cuales están a su vez conectados a muchos nodos.

1.6. Grafos de De Bruijn

Un grafo de Bruijn es un grafo dirigido que representa solapamientos entre secuencias de símbolos. El grafo tiene **m** vértices que consisten en todas las posibles secuencias de longitud **n** de los símbolos dados. Este grafo permite la aparición de elementos repetidos.

Para un grafo de Bruijn para un entero fijo de valor k tenemos que:

Sus nodos son todos los k-mers (sub secuencias de longitud k) presentes en las lecturas.

Por cada (k+1)-mer (sub secuencias de longitud k+1) presente en las lecturas, hay una arista entre el prefijo k-mer y el sufijo k-mer de la subsecuencia.

Este tipo de grafos son importantes por su utilidad en la reconstrucción de secuencias genómicas y se utilizan en la mayor parte de las aplicaciones utilizadas en el ensamblaje de lecturas cortas como las obtenidas mediante las plataformas de Solexa y SOLiD.

Una secuencia se define como una sucesión de símbolos de la forma:

$s_i, s_{i+1}, s_{i+2}, \ldots s_k$ donde k es la longitud de la secuencia.

El número de sub-secuencias se calcula con $(m+1)^n$. Por ejemplo si m = 2 y n=2, entonces se tendrán $(2 + 1)^2 = 3^2 = 9$ sub-secuencias.

La sub-secuencia está definida como la sucesión de símbolos de una longitud n, que se escribe como: $s_i\ s_{i+1}\ s_{i+2}\ \ldots s_{i+(n-1)}$, para cada uno de los símbolos.

Continuando con el ejemplo, las sub-secuencias resultantes son: S_s={00,01,02, 10, 11, 12, 20, 21, 22}.

El uso de los grafos de De Bruijn se da ya que permite la elaboración de un grafo de k-mers (trozos de secuencia de longitud fija) en el que se representan los overlaps presentes en todas las secuencias. Este método tiene la ventaja de que ya no es

necesario realizar una comparación de todos contra todos, pero tiene unos altos costes de memoria para trabajar con el grafo completo. (The Computer Architecture Department at the University of Malaga, 2010)

Tras la elaboración inicial del grafo, que se realiza en tiempo lineal con respecto al número de secuencias. Posteriormente se aplican modificaciones y simplificaciones al grafo uniendo nodos, eliminando burbujas y seleccionando los mejores caminos hasta obtener un grafo irreducible del que se obtienen los contigs Hay multitud de aplicaciones o ensambladores de secuencias que trabajan con De Bruijn Graphs, entre los principales podemos mencionar a:

- Velvet
- Euler
- ABySS
- AllPaths
- SOAPdenovo

A continuación se muestran ejemplos de secuencias, el valor de k y el grafo resultante.

Caso 1. Lectura simple

sec1: ACTG

k = 3

$$ACT \rightarrow CTG$$

Figura 7: Grafo de lectura simple.

Caso 2. Lecturas múltiples

sec1: ACTG

sec2: CTGC

sec3: TGCT

Valor de k = 3

$$ACT \rightarrow CTG \rightarrow TGC \rightarrow GCT$$

Figura 8: Grafo de lecturas múltiples.

Caso 3. Redundancias

sec1: ACTG

sec2: CTGC

sec3: CTGA

sec4: TGCT

$$\begin{array}{l} ACT \rightarrow CTG \rightarrow TGC \rightarrow GCT \\ \qquad\qquad\quad \searrow \\ \qquad\qquad\quad\ TGA \end{array}$$

Figura 9: Grafo de lecturas redundantes.

Caso 4. Repeticiones

sec1: ACTG

sec2: CTGC

sec3: TGCT

sec4: GCTG

sec5: CTGA

sec6: TGAC

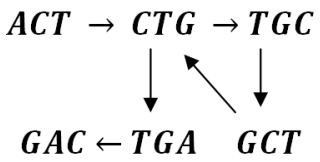


Figura 10: Grafo de lecturas con repeticiones.

Los ensambladores que utilizan estos tipos de algoritmos, generalmente, tienen varias fases diferenciadas para realizar su tarea. Estas fases son las siguientes:

Generar grafo inicial: En esta fase se trata de formar el grafo de Bruijn que contendrá la información de las lecturas, almacenando la información de los k-mers y sus nucleótidos vecinos. La representación del grafo se divide en tres pasos:

Encontrar la definición de todas las longitudes de los k-mers únicos.

Una vez clasificados dicha longitud se generan los nodos del grafo a partir de los k-mers anteriores

Se generan las aristas que relacionan dichos nodos.

Corrección de errores: Ya creado el grafo de Bruijn completo tenemos que analizar el mismo para corregir los posibles errores que se hayan generado en el proceso de creación. Dichos errores pueden ser debidos al almacenaje de información errónea, repetida o con asociaciones erróneas entre nodos. Si se han producido errores de este tipo puede dar lugar a un grafo erróneo y posiblemente con mayor longitud de lo deseado.

Agrupar nodos: Ahora se agrupa todos los nodos (lecturas) para conseguir los Scaffolds. Dicho proceso debe dar lugar a una super secuencia única que será considerada como la solución óptima del genoma buscado. En este punto podríamos considerar la opción de recorrer toda la secuencia generada para conseguir descubrir

si hay huecos en la misma. Si es así, y con toda la información que se dispone, se intenta rellenar dichos huecos para representar la secuencia del genoma lo más exacta posible.

1.7. Visualización de grafos

El estudio de redes complejas en ámbitos como la biología ha tomado forma como una disciplina en sí misma, la Network Science o análisis de redes.

Dado que uno de los objetivos del Network Science es obtener información útil y relevante a partir de una red, la visualización de la red es un factor clave. Una buena visualización puede resaltar los nodos relevantes, dar pistas sobre grupos cohesionados dentro de la red… o simplemente presentar grandes cantidades de información de forma atractiva.

Existen muchas herramientas para visualizar redes complejas, alguna de estas son:

Graphviz

Es una herramienta de código abierto para la visualización de grafos. Cuenta con importantes aplicaciones en la creación de redes, la bioinformática, la ingeniería de software, base de datos y diseño de páginas web, aprendizaje de máquina, y en las interfaces visuales para otros dominios técnicos.(Graphviz)

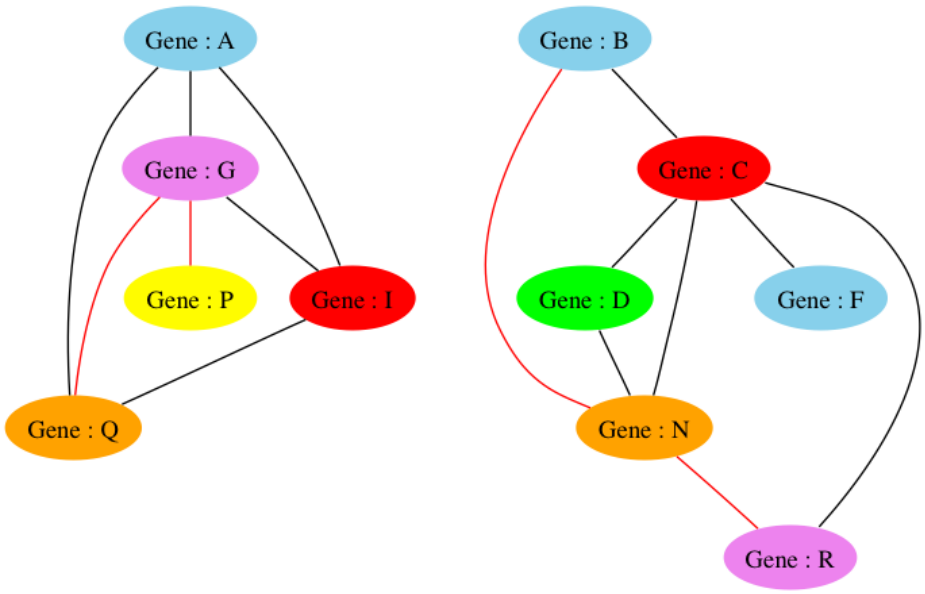


Figura 11: Visualización de una red utilizando la herramienta Graphviz.

Fuente: (Arif Bilgin, 2014)

Gephi

Es un software open source que nos permite visualizar redes complejas de todo tipo, para ámbitos tan dispares como el de la salud, la política, el social media, etc. Se trata de una herramienta muy completa que permite ser utilizada tanto por el profesional como por el usuario amateur

Antes de probar la herramienta es muy recomendable echarle un vistazo rápido al breve tutorial que han realizado desde Gephy para tener algunos conocimientos básicos. También es necesario tener algunos conocimientos sobre redes (conceptos como nodos, vértices, nodos principales,...) (gephi.org, 2014)

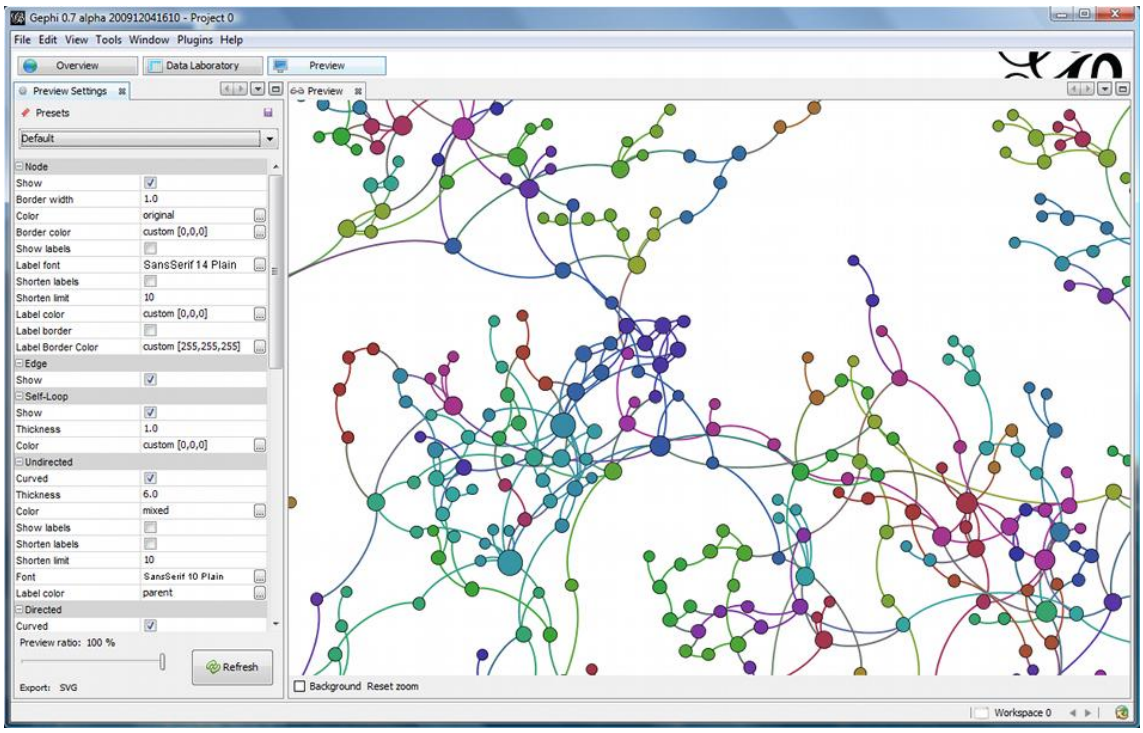


Figura 12: Visualización de una red con la herramienta Gephi

Fuente: (gephi.org, 2014)

Cytoscape

Es una plataforma de software para la biología computacional y bioinformática, útil para la integración de datos, y para visualizar y realizar cálculos sobre las redes de interacción molecular. Para empezar a trabajar con Cytoscape lo primero que necesitamos es crear o cargar una red. Cytoscape reconoce numerosos formatos de archivos (.sif, .nnf, .gml, .xls, . . .), permite la importación desde un archivo tabular (.txt, .xls) y ofrece la posibilidad de cargar una red a través de un webservice. Si nos interesa, también podemos crear o modificar nuestra propia red, añadiendo los nodos y los enlaces pertinentes. Además tiene la posibilidad de cargar una sesión previamente guardada (.cys). La sesión, además de guardar la red, también guarda los atributos modificados (color, forma, layout,.) así como la posición de las ventanas y algunas otras preferencias.

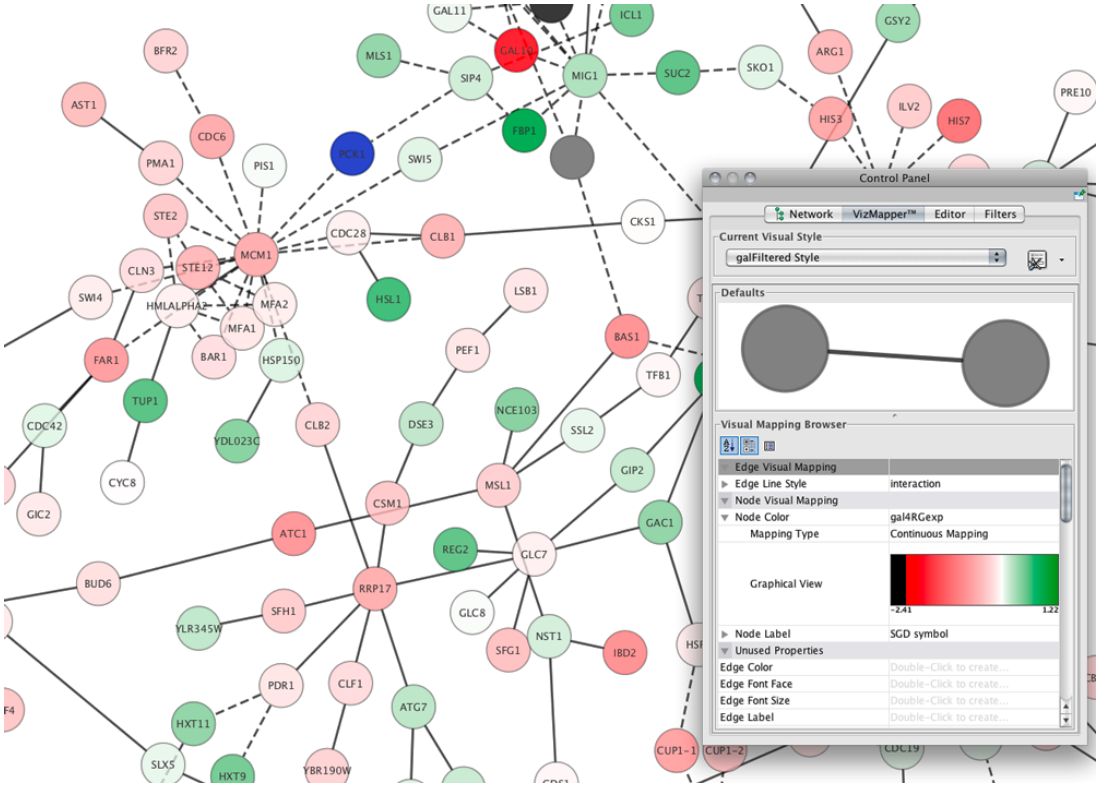


Figura 13: Visualización de una red con la herramienta Cytoscape.

Fuente: (Cytoscape Consortium, 2014)

aiSee

Es una herramienta comercial, gratuita para usos no comerciales, calcula automáticamente un diseño personalizable de gráficos especificados en GDL (Grafic Description Language).(aiSee)

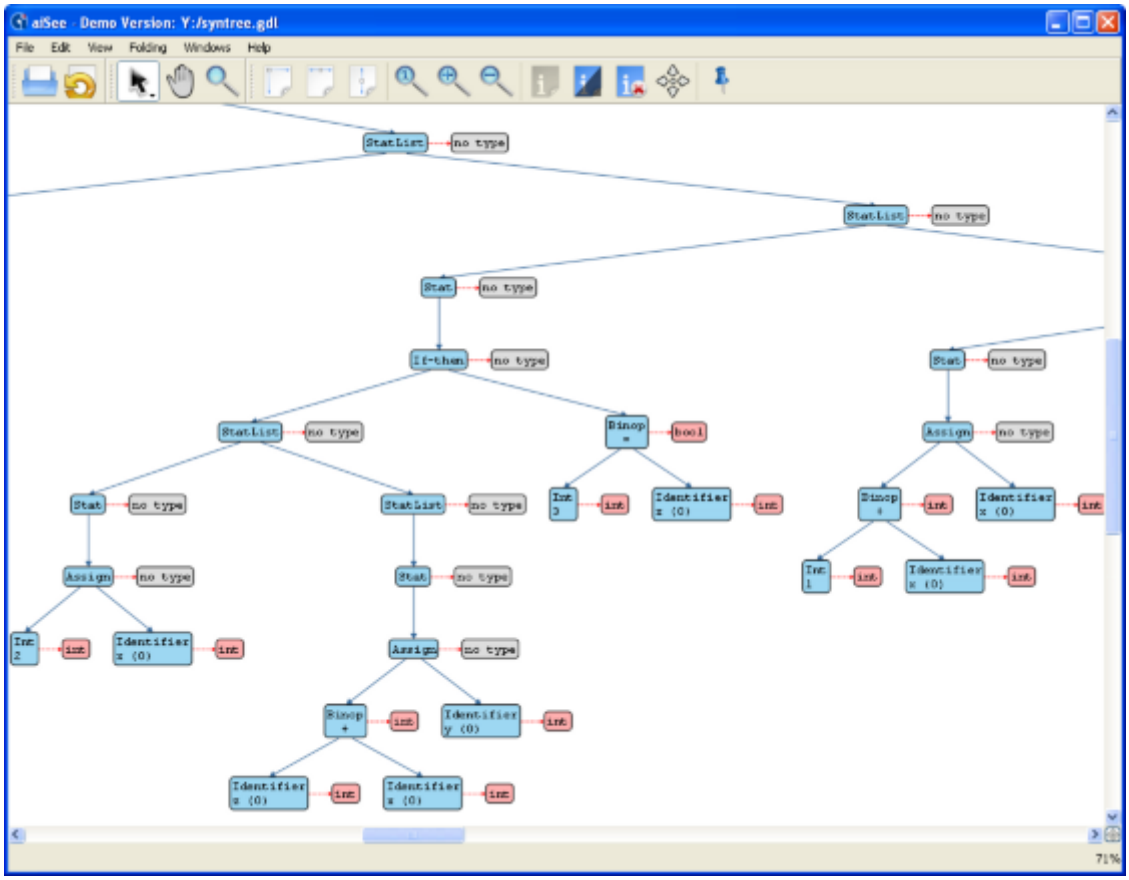


Figura 14: Visualización de una red utilizando la herramienta aiSee.

Fuente: (Informatik)

1.8. Velvet

Velvet (Daniel R. Zerbino, 2008) es un conjunto de algoritmos escritos en lenguaje de programación C creado para trabajar con el ensamblaje de genomas de lecturas cortas que van desde 25 a 50pb. Esto lo logra gracias a la manipulación de grafos de Bruijn mediante la eliminación de errores y simplificación de regiones repetidas

El ensamblador Velvet puede tener un rendimiento bastante óptimo y aceptable, pero se puede producir un cuello de botella a la hora de realizar la estructura de datos para el genoma.

ETAPAS

Las etapas que sigue el ensamblador Velvet son las siguientes:

- Generar ficheros y grafo de Bruijn
- Simplificar Grafo
- Eliminar errores
    - Nodos punta.
    - Burbujas.
    - Conexiones erróneas.

A continuación se presenta un diagrama que muestra los principales procesos que lleva a cabo el ensamblador Velvet.

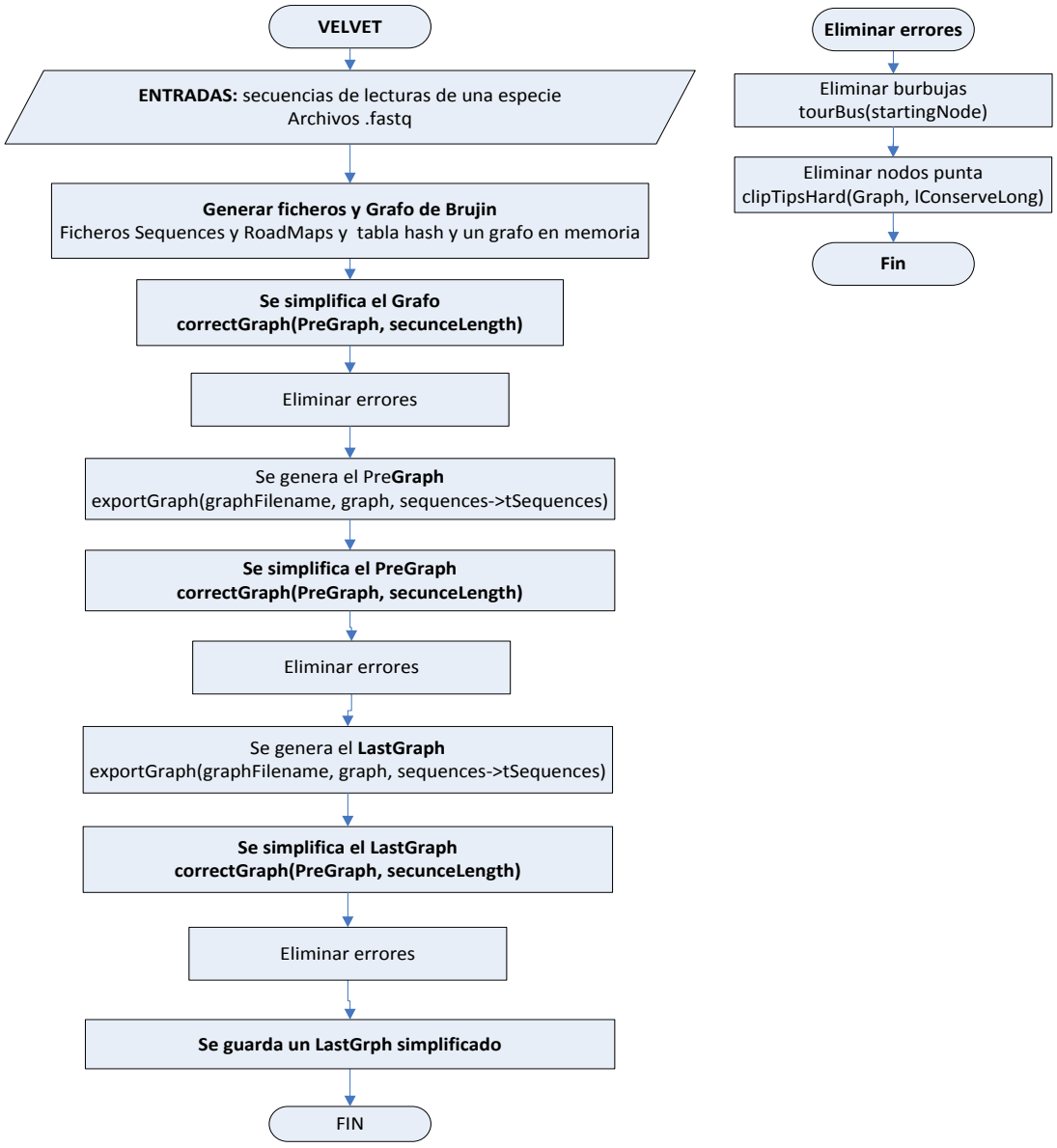


Figura 15: Procesos del ensamblador Velvet

Fuente: Elaboración propia en base a la revisión del código de Velvet

Generar ficheros y Grafo de Bruijn

En esta se desarrolla la estructura de datos general que va a utilizar el ensamblador en el resto de etapas para realizar su tarea.

En esta etapa las lecturas obtenidas a través de la máquina de secuenciación son divididas en k-mers en las cuales el valor de k es definido por el usuario en el comando de ejecución del ensamblador.

El valor de k determinará la calidad del ensamblaje, pero serán necesarias varias pruebas para poder indicar que valores de k son aceptables para un genoma en concreto utilizando este ensamblador. Por ello hay que realizar pruebas exhaustivas con varios valores de k, hasta conseguir ver valores más o menos aceptables.
De todas formas, debido a comprobaciones ya realizadas en experimentos se puede indicar que los valores de k que son cercanos a la longitud de las lecturas que tenemos pueden producir superposiciones en el ensamblado mientras que valores de k más pequeños pueden producir un mayor número de superposiciones que pueden generar errores y bucles en la generación de la estructura de datos. (Diez, 2013)

Una vez que las lecturas son divididas en k-mers se escanean, se transforman a un formato interno utilizado por Velvet y se guarda en un archivo denominado “Sequences”.

A continuación Velvet crea una tabla hash de n entradas y cada vez que un k-mer es leído se realiza un proceso de búsqueda en la tabla hash. Si el k-mer analizado no se encuentra en la tabla hash correspondiente se almacena el identificador de dicho k-mer y su posición. En cambio, si el k-mer analizado si es encontrado en la tabla hash, una referencia de este k-mer es almacenada en el archivo “RoadMaps”, el fichero RoadMaps almacena aquellos k-mers que tienen coincidencias con lecturas anteriores. La tabla hash se almacena temporalmente en memoria mientras el archivo RoadMaps se guarda permanentemente.

Finalmente, una vez que se han creado la tabla hash y el fichero RoadMaps, se utilizan para crear el Grafo de Bruijn. En dicho grafo cada k-mer perteneciente a la tabla hash (no ha sido vista antes) es un nodo del grafo mientras que las conexiones entre nodos se realizan gracias al fichero RoadMaps.
Velvet usa los Grafos de Bruijn para ensamblar lecturas cortas, Velvet representa cada k-mer obtenido de las lecturas como un único nodo en el grafo. Dos nodos se conectan si tienen un solapamiento de al menos k-1 caracteres. Osea el arco de un nodo A a un nodo B existe si los últimos k-1 caracteres del nodo A son los primeros k-1 caracteres del nodo B.

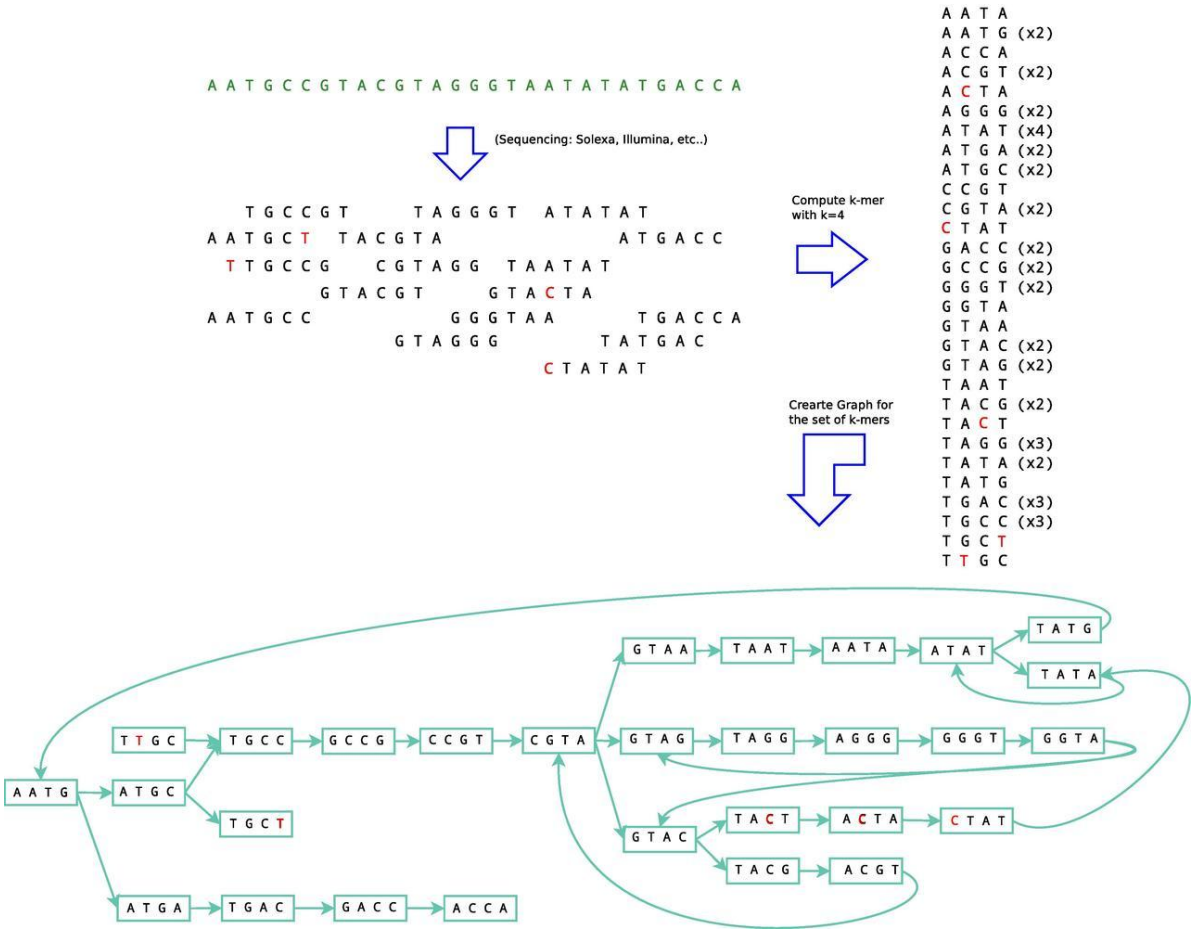


Figura 16: Ejemplo de una lectura y su grafo de Bruijn

Fuente: (Mog9207, 2013)

Simplificar Grafo

En esta etapa ya disponemos del Grafo de Bruijn totalmente construido en memoria.

Ahora tenemos que proceder a simplificar el grafo. Dicha simplificación se realiza fusionando nodos. Para fusionar dos nodos del grafo se considera lo siguiente: Si tenemos un nodo A que tiene solo una conexión saliente con un nodo B y dicho nodo B solo tiene una conexión entrante. Podemos simplificar ambos nodos en un solo nodo que combine la información de los nodos anteriores. La figura 17 ilustra este proceso en base a la figura 16.

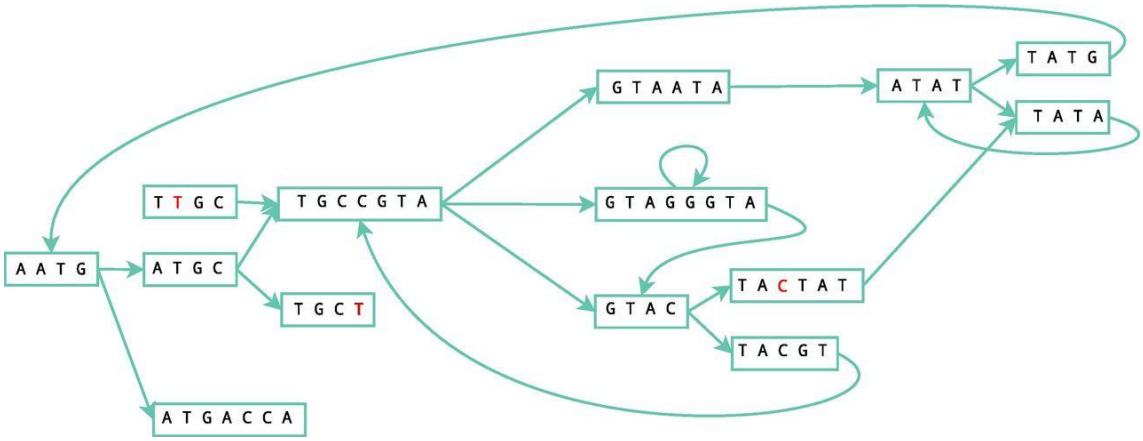


Figura 17: Simplificación del grafo de De Bruijn

Fuente: (Mog9207, 2013)

Eliminar errores

En esta tercera etapa ya disponemos de un Grafo de Bruijn simplificado.

Los errroes en el grafo pueden ser causados por los procesos de secuenciación, o simplemente pueden deberse a que la muestra biológica tiene errores. Velvet reconoce tres tipos de errores: Tips, bublles y conexiones erróneas

Eliminación de nodos Punta-Tips

Uno de los errores más comunes que podemos encontrar en un Grafo de Bruijn es encontrar caminos, sin salida dentro del grafo principal, y separados bastante de la ruta óptima principal. La eliminación de estas ramas no supone ningún problema ya que al no afectar a la ruta principal no se cambia la conectividad global del grafo. De todas formas hay que tener cuidado al eliminar estos caminos ya que puede darse el caso de que encontremos estos caminos y no estén producidos por errores.

Un nodo se considera punta y debe ser borrado si se desconecta de uno de sus extremos, si la longitud es menos de 2k y el si su multiplicidad es baja.

Una vez eliminados estos nodos punta, el grafo se somete nuevamente al proceso de simplificación.

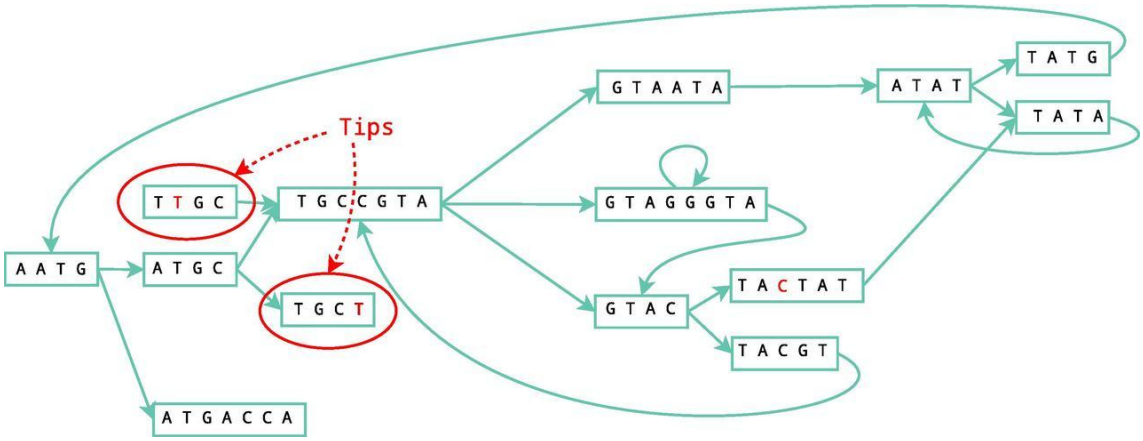


Figura 18: Eliminación de nodos punta

Fuente: (Mog9207, 2013)

Eliminar error de Burbuja

Este error se genera cuando dos caminos comienzan y terminan en los mismos nodos. Para solucionar este tipo de errores se utiliza el algoritmo “Tour Bus” el cual realiza una búsqueda en amplitud que detecta el mejor camino a seguir y determina cuáles deben ser borrados. Un ejemplo de esto se muestra en la figura 19.

Figura 19: Eliminación de error de burbuja

Fuente: (Mog9207, 2013)

Siguiendo el ejemplo de la figura 18 tenemos la figura 20 que muestra la detección del error de burbuja:

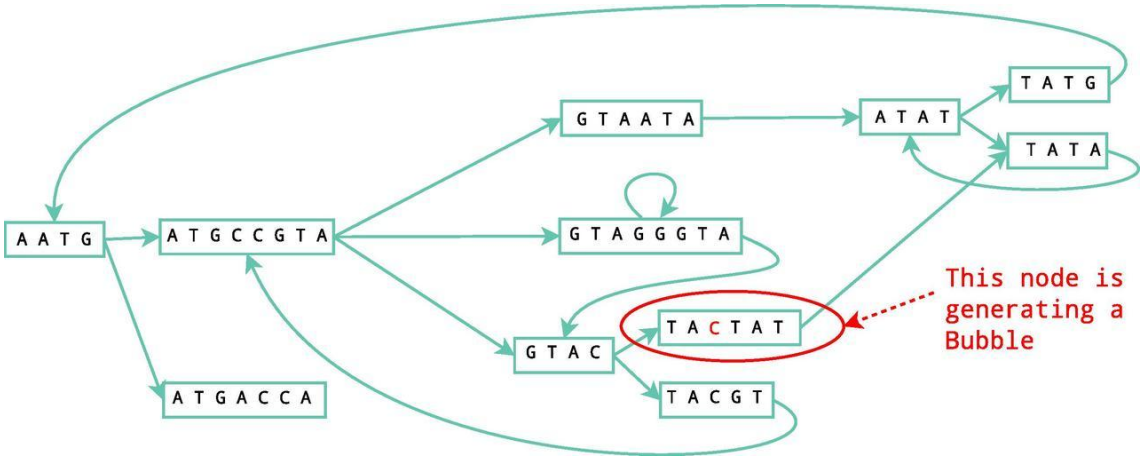


Figura 20: Eliminación de error de burbuja

Fuente: (Mog9207, 2013)

Eliminar conexiones erróneas

Estas conexiones son las que no generan rutas correctas o no crean ninguna estructura reconocible dentro del grafo. Velvet borra estos errores después de que finalice el algoritmo de Tour Bus. La aplicación de una cobertura debe ser definida por el usuario.

1.9. Metavelvet

Metavelvet es una extensión de Velvet, pero a diferencia de Velvet, está pensado para trabajar con el ensamblaje de varios genomas.

Lo procesos de Metavelvet se representan en la figura 21:

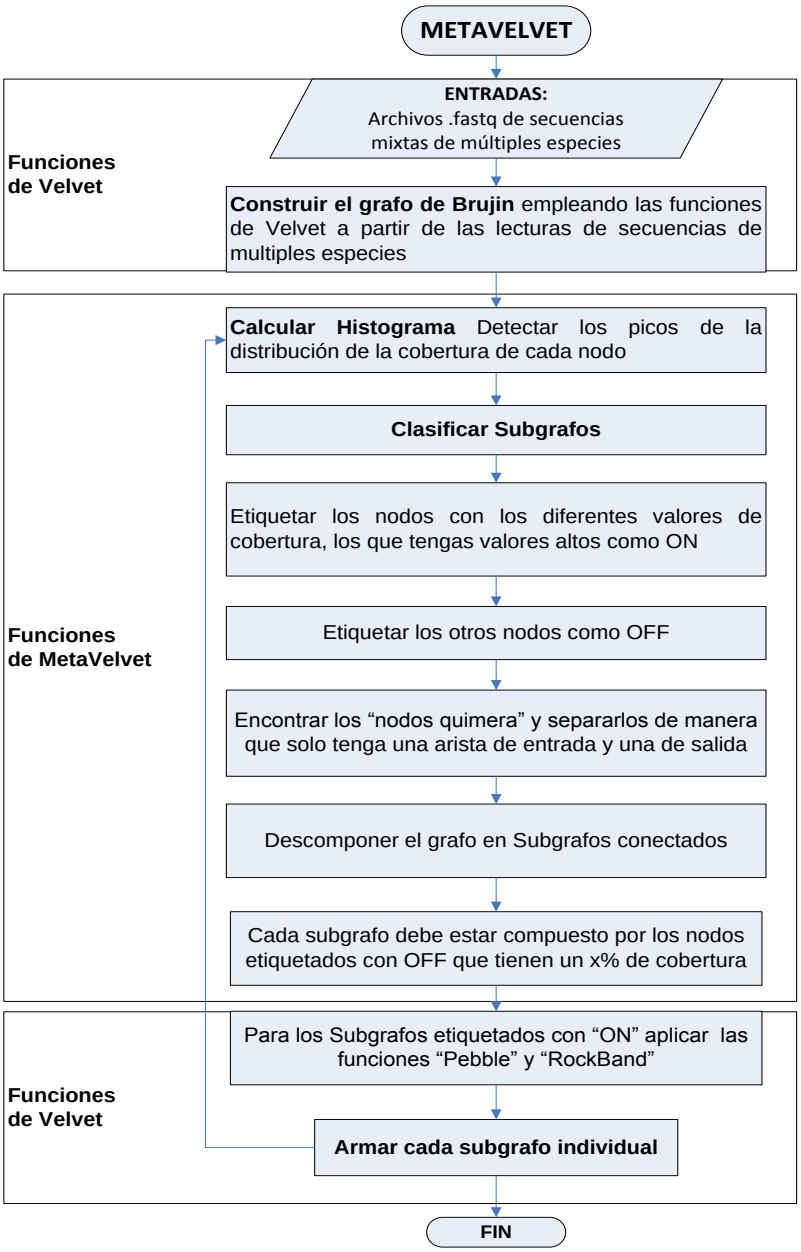


Figura 21: Fases de Metavelvet

Fuente: Elaboración propia basado en (Toshiaki N, 2012)

La figura 21 muestra los principales procesos que lleva a cabo Metavelvet, las cuales se describen a continuación (Toshiaki N, 2012).

Construir grafo de De Bruijn

Se construye un grafo de Bruijn a partir de un conjunto de lecturas de varios genomas usando la función de Velvet

Calcular Histograma

Se calcula el histograma de frecuencias k-mer y detecta múltiples picos en el histograma, cada uno de los cuales se correspondería con el genoma de una de las especies de la comunidad microbiana.

Clasificar subgrafos

MetaVelvet hace una distinción de los subgrafos compuestos por nodos que pertenecen a distintos genomas

Armar SubGrafo

MetaVelvet utiliza Velvet para armar cada subgrafo individual identificado en el paso anterior.

CAPÍTULO 2

2. Simulación de datos metagenómicos

En este capítulo se abordará lo desarrollado en el proyecto lo cual comprende:

- Selección de las especies para las simulaciones.
- El análisis y modificaciones de Velvet.
- El análisis topológico de los grafos de De Bruijn generados

2.1. Selección de especies para las simulaciones

Las especies que se seleccionaron para la simulación son especies que se encuentren en el suelo, las mismas tienen una estrecha relación filogenética (parentesco entre especies).

A continuación la lista de las especies que utilizaremos.

Cuadro 2: Especies y sus porcentajes utilizadas para las simulaciones

# de Genomas	Nro	Genomas	Abundancia	Cantidad
2	1	NC_000964.3| Bacillus subtilis	.5	500000
	2	NC_007492.2| Pseudomonas fluorescens	.5	500000
3	1	NC_000964.3| Bacillus subtilis	.25	500000
	2	NC_007492.2| Pseudomonas fluorescens	.25	500000
	3	NC_013446.2| Comamonas testosteroni	.5	500000
6	1	NC_000964.3| Bacillus subtilis	.16	500000
	2	NC_007492.2| Pseudomonas fluorescens	.16	500000
	3	NC_013446.2| Comamonas testosteroni	.16	500000
	4	NC_003997.3| Bacillus anthracis	.16	500000
	5	NC_010002.1| Delftia acidovorans	.16	500000
	6	NC_002516.2| Pseudomonas aeruginosa	.20	500000
12	1	NC_000964.3| Bacillus subtilis	.08	500000
	2	NC_007492.2| Pseudomonas fluorescens	.08	500000
	3	NC_013446.2| Comamonas testosteroni	.08	500000
	4	NC_003997.3| Bacillus anthracis	.08	500000
	5	NC_010002.1| Delftia acidovorans	.08	500000
	6	NC_002516.2| Pseudomonas aeruginosa	.08	500000
	7	NC_012560.1| Azotobacter vinelandii	.08	500000
	8	NC_004722.1| Bacillus cereus	.08	500000

	9	NC_005957.1| Bacillus thuringiensis	.08	500000
	10	NC_013716.1| Citrobacter rodentium	.08	500000
	11	NC_004578.1| Pseudomonas syringae	.10	500000
	12	NC_002947.3| Pseudomonas putida	.10	500000

Fuente: Elaboración propia

Todas estas especies han sido tomadas de la página http://www.ncbi.nlm.nih.gov/ en donde debemos descargar los archivos .fasta de cada genoma y los almacenamos en una carpeta, quedando de la siguiente manera.

Figura 22: Archivos fasta de los 12 genomas

Fuente: Elaboración propia

Para trabajar con estos archivos vamos a utilizar la herramienta Gensim, y la utilizamos para que nos proporcione un set de datos de 500000 lecturas pareadas de 100 nucleótidos cada una a partir de una mezcla de n genomas, los cuales ocupan un porcentaje denominado *abundance* en la mezcla, se utiliza un archivo denominado abundance.txt el cual especifica la abundancia de cada genoma.

Ejecución de Gensim para 2 genomas

Para utilizar Gensim necesitamos crear un archivo en el que se especifique el porcentaje de abundancia que tendrá cada especie en la mezcla, Esto lo hacemos en el archivo abundance.txt el cual podemos observar a continuación.

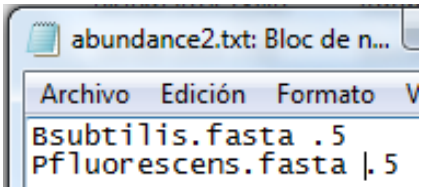


Figura 23: Archivo abundance2.txt para 2 genomas.

Fuente: Elaboración propia tomado del archivo abundance.txt creado

Este nos especifica que cada especie tendrá un porcentaje de 0.5 en la mezcla.

El comando que ejecutamos en Gensim es el siguiente:

gensim$ time ./GemReads.py -R . -a abundance2.txt -n 1000000 -l 100 -p -m models/ill100v5_p.gzip -q 64 -u 400 -s 100 -o **meta2genome**

Este comando nos indica que se utilizará el archivo abundance2.txt para la mezcla y se harán 1´000000 lecturas(500 mil por cada genoma) con 100 nucleótidos cada una.

```
ffojeda@debianffojeda:~/Downloads/gensim$ time ./GemReads.py -R . -a abundance2.txt -n 1000000 -l 100
 -o meta2genome
Bsubtilis
Bsubtilis
Pfluorescens
Pfluorescens

real    59m4.913s
user    58m21.499s
sys     0m22.965s
ffojeda@debianffojeda:~/Downloads/gensim$
```

Figura 24: Ejecución de Gensim-para 2 genomas.

Fuente: Elaboración propia tomado del archivo abundance.txt creado

Al ejecutar el comando anterior se generan los ficheros FASTQ (FIRST and SECOND) que se utilizarán como datos de entrada para el Velveth. **meta2genome_fir.fastq y meta2genome_sec.fastq**

Se realizó todo esto tanto para 3, 6 y 12 genomas, los tiempos de ejecución los resumimos en el siguiente cuadro:

Cuadro 3: Tiempo de ejecución de Gensim

Nro Genomas	Lecturas	Tiempo
2	1000000	59 min 4.9 seg
3	1500000	87 min 9,7 seg
6	3000000	166 min 36 seg
12	6000000	339 min 40 seg

Fuente: Elaboración propia, datos obtenidos

Cada uno de los conjuntos de datos generados anteriormente han sido almacenados para ser posteriormente utilizados por el ensamblador.

2.2. Ejecución del ensamblador

El ensamblador Velvet está dividido dos funciones principales que son. Velveth y Velvetg, las mismas que se ejecutan una a continuación de otra.

Figura 25: Funciones principales de Velvet

Fuente: Elaboración propia.

2.2.1. Ejecución de Velveth

Para la ejecución de Velveth emplearemos los archivos fastq generados anteriormente, siguiendo el siguiente esquema:

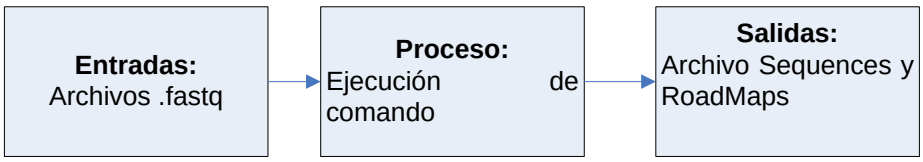


Figura 26: Ejecución de Velveth

Fuente: Elaboración propia

En el código del ensamblador Velvet encontramos una función principal denominada run.c que ejecuta los procesos de Velveth.

Entradas:

Para ejecutar la función run.c necesitamos tener los archivos .fastq que se generó anteriormente con la utilización del Gensim.

Los archivos .fastq utilizados son los siguientes:

Cuadro 4: Archivos fastq generados con Gensim

Archivos	**Número de Genomas**
meta2genome_fir.fastq meta2genome_sec.fastq	2 genomas
meta3genome_fir.fastq meta3genome_sec.fastq	3 genomas
meta6genome_fir.fastq meta6genome_sec.fastq	6 genomas
meta12genome_fir.fastq meta12genome_sec.fastq	12 genomas

Fuente: Elaboración propia

Comando:

>velveth output221/ 21 -fastq -short data/meta2genome_fir.fastq data/meta2genome_sec.fastq

La sintaxis de este comando está detallada en el manual que viene con el ensamblador.

"output221/" Es la carpeta donde se guardará le resultado

"*21*" Valor de k

"*-fastq*" Formato en que están los archivos de las lecturas

"*data/meta2genome_fir.fastq data/meta2genome_sec.fastq* " Directorio donde se leen las lecturas de los genomas.

Se ejecuta este comando con los valores de *k* de *21, 23, 25,27, 29* y *31* para cada conjunto de genomas.

Proceso:

Velveth es el encargado de ayudar a formar la estructura de datos que se va a utilizar para ensamblar todo el conjunto de los genomas. Con Velveth se leen los datos simulados de secuenciación metagenómica(.fastq) y se genera algunos archivos de salida.

Salidas

El resultado de la ejecución de Velveth son dos archivos:

Sequences

RoadMaps

Este proceso (ejecutar Velveth) lo realizamos tanto para 2, 3, 6 y 12 genomas y los diferentes valores de k (21. 23. 25. 27. 29 y 31),, obteniendo con esto los archivos Sequences y Roadmaps los mismos que se almacenar en carpetas diferentes según sea el número de genomas usados y el valor de k utilizado.

Con esto en la carpeta llamada **output221**(**2**-Número de genomas, **21** Valor de k) se guardan los archivos resultantes de la ejecución de Velveth.

Luego de ejecutar para cada conjunto de grupos y valores de k tenemos las siguientes carpetas:

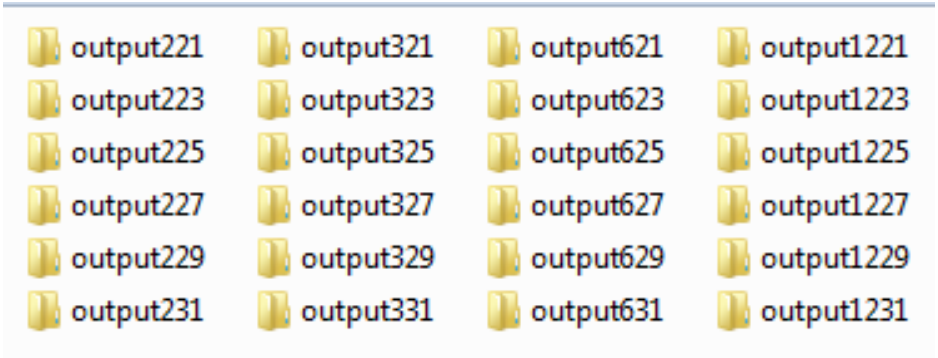


Figura 27: Carpetas creadas para almacenar archivos generados por Velveth

Fuente: Elaboración propia

Hasta ahora se han creado estas 24 carpetas y en cada una de estas está creado el archivo Sequences y RoadMaps correspondiente.

En vista de que el conjunto de datos actual comprende lecturas de dos genomas diferentes, en el archivo de Sequences, cada secuencia que corresponde a cada genoma está etiquetada con un NC diferentes:

NC_000964.3 para las lecturas de un genoma y

NC_0007492.2 para la lecturas del otro genoma como se muestra en la siguiente imagen del archivo Sequences generado

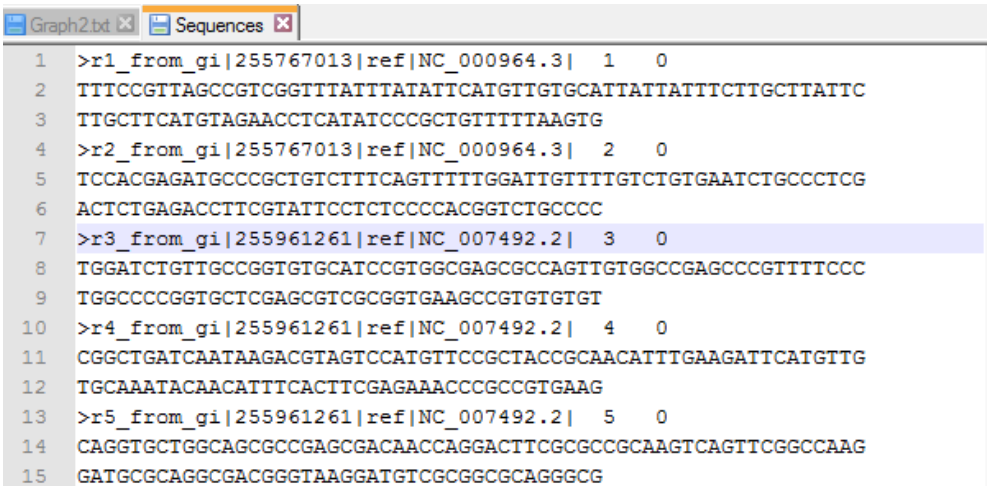


Figura 28: *Archivo Sequences generado*

Fuente: Elaboración propia basado en archivo generado

Esto se da tanto cuando trabajamos con 2, 3, 6 y 12 genomas, cada lectura de genoma está etiquetada con el NC y un número.

2.2.2. Análisis de Velvetg

El ensamblador Velvet ha probado ser uno de los mejores ensambladores al momento de trabajar con lecturas cortas individuales de genomas, pero lo que se tiene en este proyecto es un conjunto de lecturas cortas metagenómicas, es decir las lecturas pertenecen a varios genomas. Por tal razón hemos identificado y suprimido las funciones en las que Velvet elimina los nodos que los detecta como errores ya que estos no necesariamente son errores porque pueden ser nodos que pertenecen a otro genoma. Esto ha dado origen a una nueva versión de Velvet que la hemos denominado VelvetLite.

Las funciones que se han suprimido son las siguientes:

- Función "*clipTipsHard(Graph, lConserveLong)".* Esta Función de Velvet elimina los nodos punta del grafo de Bruijin.
- Función *" tourBus(startingNode)"* Esta Función elimina las burbujas del grafo.

Dichas funciones han sido suprimidas al momento que se manipula el Graph y el LastGrpah. A continuación se presenta como quedaría el diagrama de flujo de VelvetLite

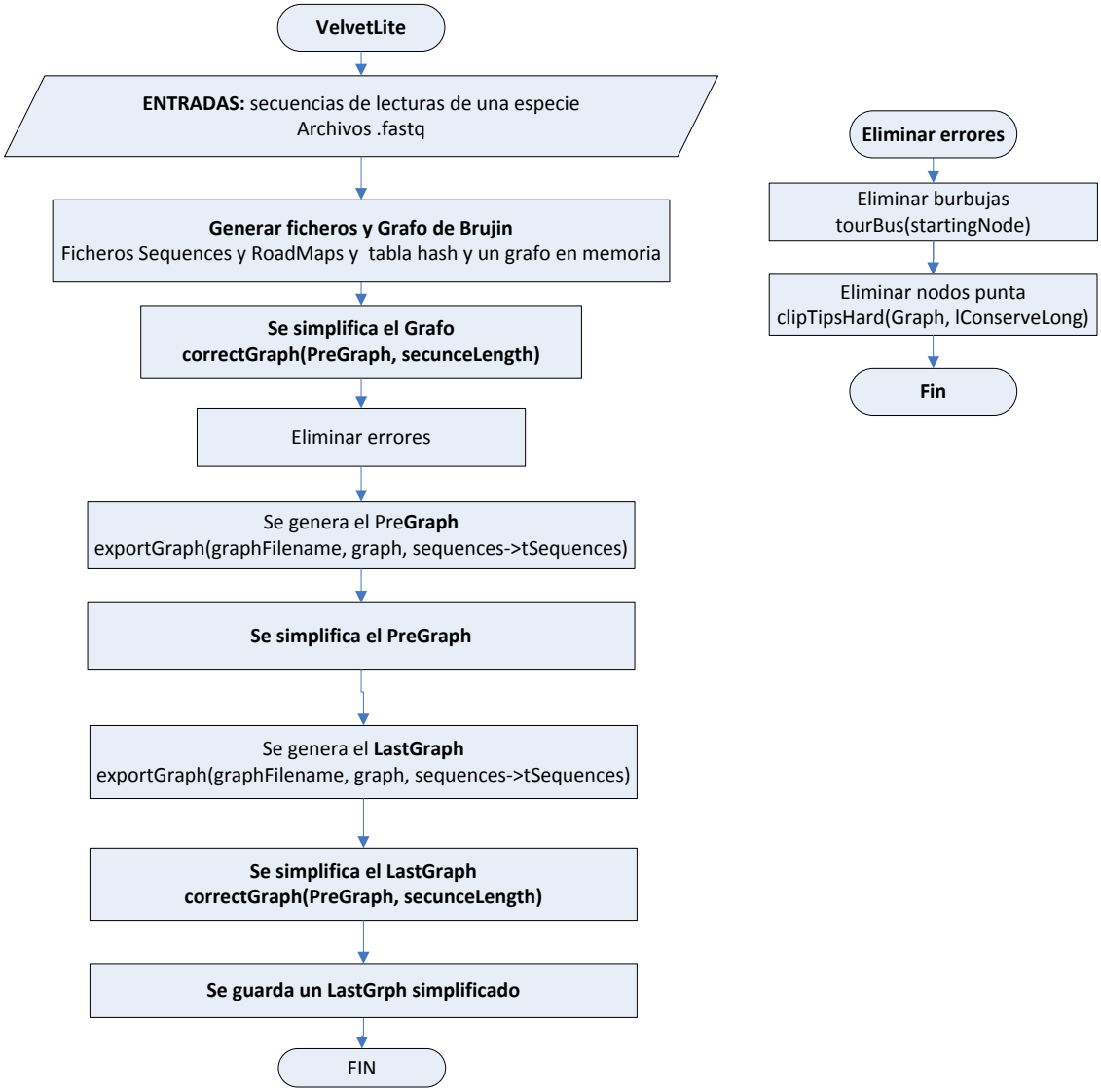


Figura 29: Principales procesos de VelvetLite

Fuente: Elaboración propia

2.2.3. Ejecución de Velvetg-VelvetLite

Con los cambios realizados al archivo run2.c (Velvetg) se ha creado una versión diferente de Velvet a la que denominaremos VelvetLite. VelvetLite nos permitirá generar un grafo con todos los nodos, sin la eliminación de estos.

Para la ejecución de Velveth se realizan los siguientes pasos:

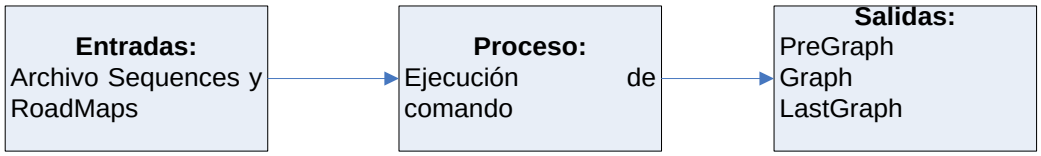


Figura 30: Ejecución de Velvetg.

Fuente: Elaboración Propia

Entradas:

- Sequences

- RoadMaps

Comando:

> *./velvetg / output221 -read_trkg yes*

La sintaxis de este comando está detallada en el manual que viene con el ensamblador.

"*output221/*" Es la carpeta donde se guardará el resultado

"*21*" Valor de k

Se ejecuta este comando con los valores de k de 21, 23, 25,27, 29 y 31 para cada conjunto de genomas.

Proceso: Velvetg genera el archivo Prergraph, y lo simplifica utilizando las funciones de Velvet, dando como resultado un grafo simplifica denominado Graph.

Luego de crear el archivo Graph, suprimimos en el código las funciones que eliminan los nodos que no tienen salida(*clipTipsHard(Graph, lConserveLong)*) o los nodos burbuja (*tourBus(startingNode).* Esto nos permitirá conservar todos los nodos generados para la construcción del archivo LastGraph.

Salidas:

- PreGraph
- Graph
- LastGraph

Una vez ejecutado velvetg cada carpeta de las salidas está conformada por los siguientes archivos:

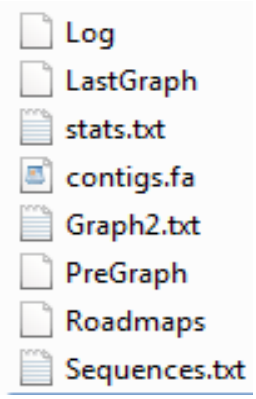


Figura 31: Contenido de la carpeta output luego de ejecutar VelvetLite.

Fuente: Elaboración propia en base a los datos obtenidos

Ahora tenemos el archivo Graph con la información de todos los nodos generados, este archivo nos sirve para de él extraer los datos necesarios para

hacer el análisis topológico del grafo de Bruijn. Una vez extraídos estos datos podemos responder preguntas tales como:

Número de nodos generados._ Es el número total de nodos que contiene el grafo.

Número de nodos Puros._ Son aquellos nodos formados por lecturas de un solo genoma.

Número de nodos Impuros._ Son aquellos nodos formados por lecturas de 2 o más genomas.

Número de nodos Quimera._ Son aquellos cuyo grado de salida es mayor de 2.

CAPÍTULO 3

3. Análisis de grafo de De Bruijn

Para analizar la gran cantidad de datos generados por Velvet se extrae la información de los archivos generados hasta el momento. Los datos serán guardados en una base de datos para que una vez allí tengamos la posibilidad de hacer su análisis a través de consultas a dicha base de datos

3.1. Extracción de la información del grafo de De Bruijn

Para realizar lo descrito se analiza la estructura de los archivos generados. Utilizaremos el archivo Graph que almacena el id de los nodos y de las secuencias que los conforman así como también el número de secuencias por las que está formado cada nodo. Los datos obtenidos serán enlazados con la información del archivo Sequences, ya que en este archivo encontramos el id de las secuencias y la información del genoma la que pertenece cada secuencia.

Del archivo Graph se obtienen los datos:

- Id del Nodo
- Número de secuencias de cada nodo
- Id de las secuencias que conforman un nodo

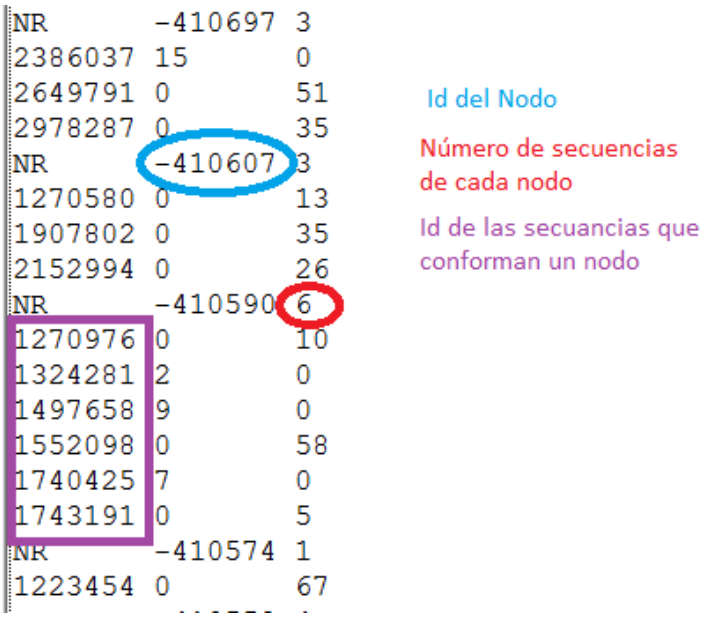


Figura 32: Archivo Graph creado por VelvetLite

Fuente: Elaboración propia en base a los datos generados

Del archivo de secuencias se obtiene:

- Id de la secuencia

Las secuencias son de tamaño 100

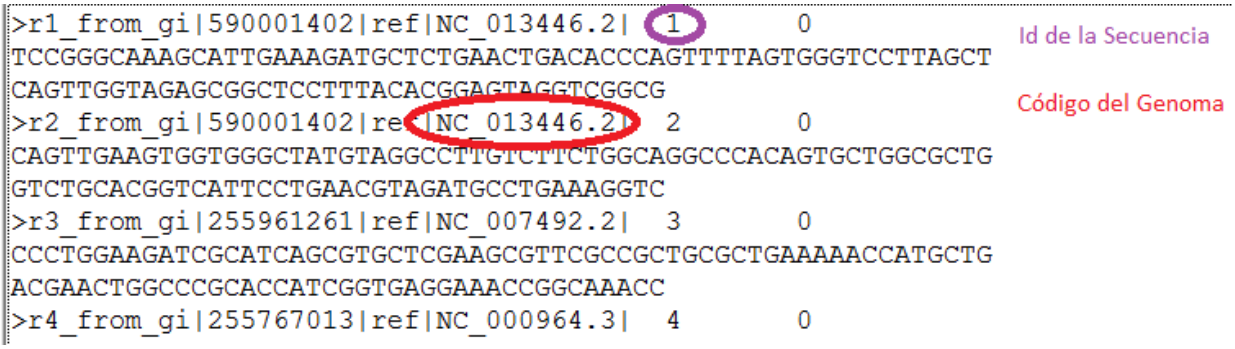


Figura 33: Archivo Sequences creado con Velveth

Fuente: Elaboración propia en base a los datos generados

El archivo Graph es procesado utilizando un script en python, el cual realiza el siguiente proceso:

- Busca en el archivo ***Graph*** la cadena NR.
- En la línea que encuentra NR toma como datos el id del nodo y el número de lecturas que conforman ese nodo
- Con esto genera un archivo en el que cada lectura se identifique con el id del nodo correspondiente
- Guarda los datos de IdNodo y IdSecuencia en el archivo ***GraphLecturas*** y pasa a la siguiente línea donde encuentre NR. 2661

La figura 34 muestra un diagrama de flujo del proceso descrito anteriormente:

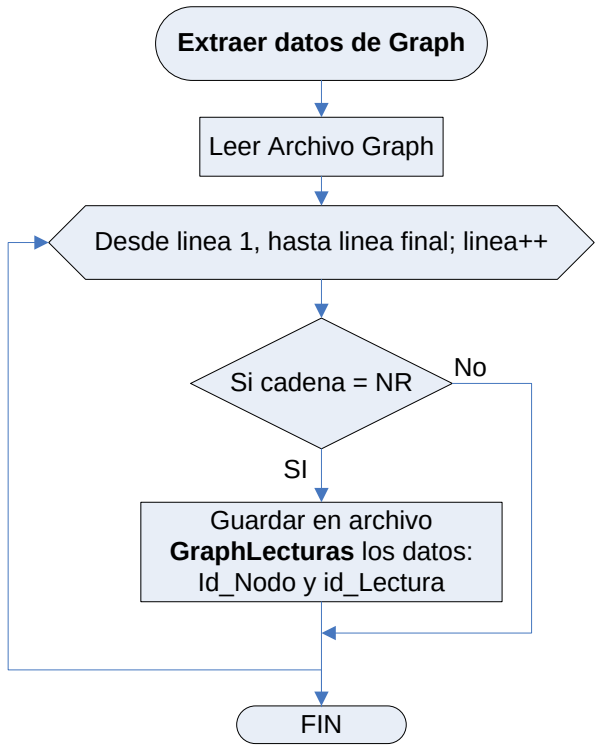


Figura 34: Extracción de datos del archivo Graph

Fuente: Elaboración propia.

Entonces el archivo de salida GraphLecturas queda estructurado como se muestra en la figura 35:

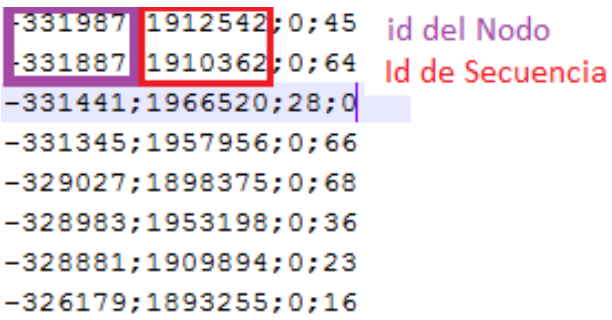


Figura 35: Estructura del archivo GraphLecturas

Fuente: Elaboración propia en base a los datos generados

También necesitamos extraen el id de la secuencia y el código del genoma para esto utilizamos el archivo de secuencias lo procesamos con el script en python el cual realiza lo siguiente:

- Buscar el símbolo > que es como se denota el inicio de una secuencia
- Extraer el código del genoma (NC_007493.2) y el número que se encuentra a continuación que vendría a ser el id de la secuencia

La figura 36 muestra el diagrama de flujo del proceso anterior.

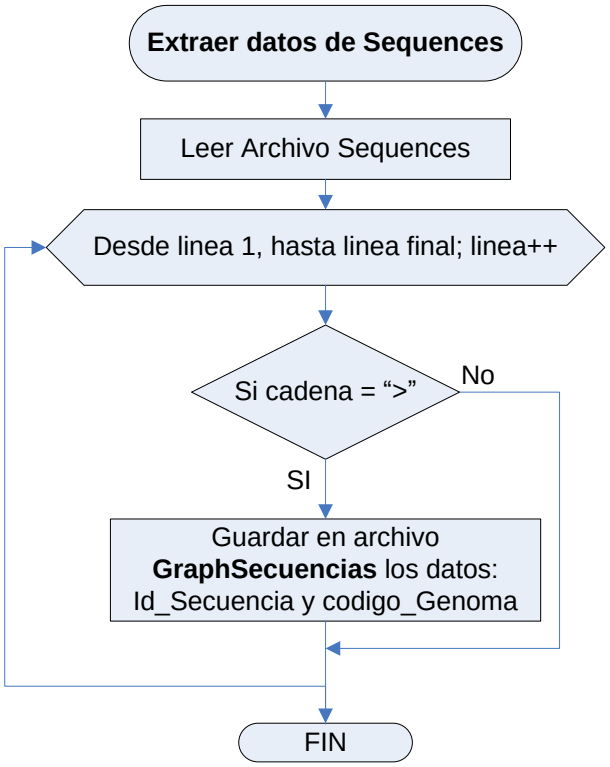


Figura 36: Extracción de datos del archivo Sequences

Fuente: Elaboración propia en base a los datos generados

Dando como resultado el archivo GraphSecuencias que queda de la siguiente manera:

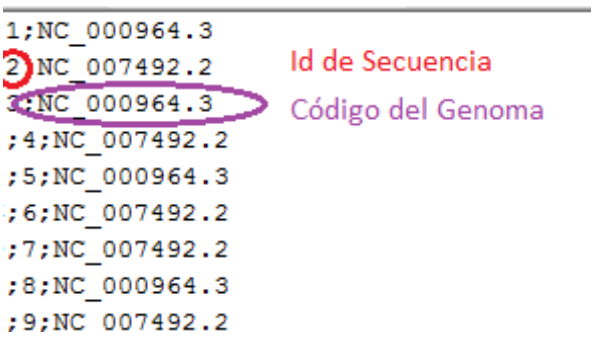


Figura 37: Estructura del archivo GraphSecuencias

Fuente: Elaboración propia en base a los datos generados

Con estos 2 archivos GraphSecuencias y GraphLecturas los cuales contienen la información de los nodos generados, utilizamos la herramienta “TALEND OPEN STUDIO for Big Data”, la cual nos servirá para manipularlos de una mejor manera estos archivos y extraer la información para posteriormente almacenarla en una base de datos.

Creamos los siguientes job:

- Formatear Archivos
- Generar Lecturas
- Generar Secuences
- Generar Resultado

A continuación la descripción de cada job

Job Formatear Archivos

La figura 38 muestra el job Formatear Archivos, en este Job se procesan los archivos Graph y Sequences

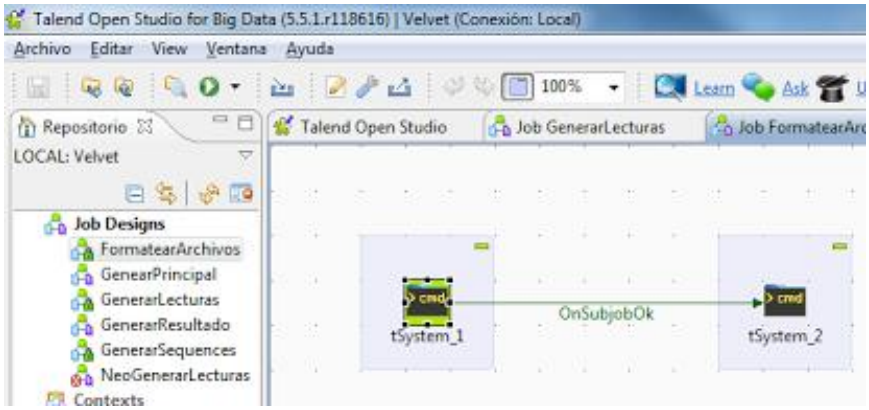


Figura 38: Job procesar Graph y Sequences

Fuente: Elaboración propia captura de pantalla de procesos

Entradas

Archivo Graph

Archvo Sequences

Proceso

A Graph y Sequences se los procesa con los scripts de python

Salidas

Un archivo GraphLecturas

Un Archivo GraphSecuencias

Job Generar Lecturas

La figura 39 muestra el job Generar lecturas:

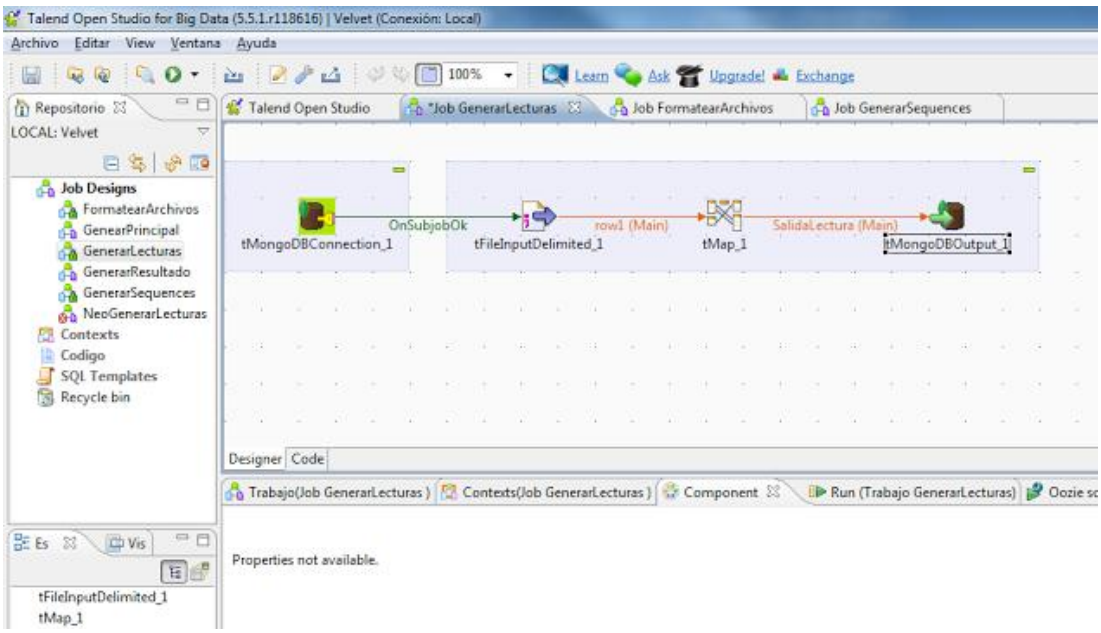


Figura 39: Job generar lecturas

Fuente: Elaboración propia en base a los datos generados

Entradas

Archivo GraphLecturas

Proceso

Se mapean los datos del archivo de entrada y se almacenan en la base de datos MongoDB creando la Tabla(Collection) Lectura que tiene el siguiente esquema:

Cuadro 5: Estructura de la tabla Lectura almacenada en la base de datos

ID_NODO	ID:LECTURA

Fuente: Elaboración propia

Salida

Tabla Lectura almacenada en MongoDB

Job GenerarSecuences

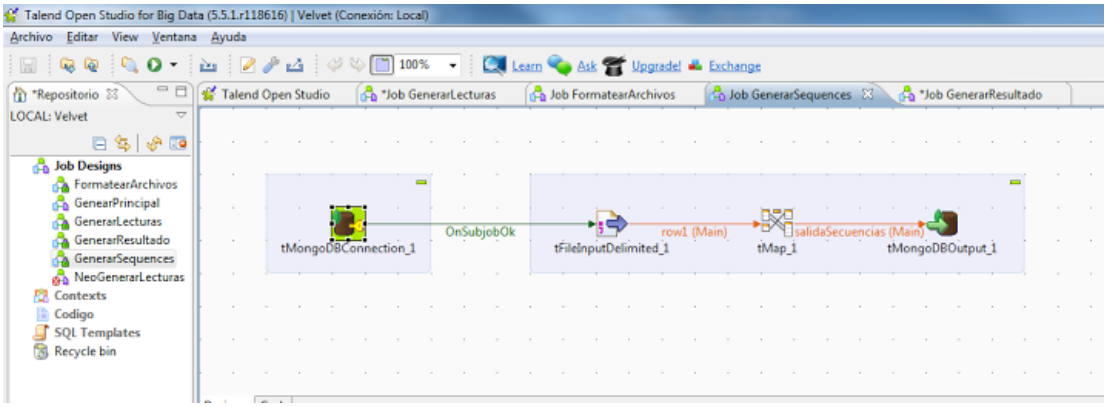


Figura 40: Job Generar Secuencias

Fuente: Elaboración propia en base a los datos generados

Entradas

Archivo Graph2Secuencias

Proceso

Se mapean los datos del archivo de entrada y se almacenan en la MongoDB creando la Tabla(Collection) Secuencias que tiene el siguiente esquema:

Cuadro 6: Estructura de la tabla Secuencias almacenada en la base de datos

ID_LECTURA	CODIGO_GENOMA

Fuente: Elaboración propia

Salidas

Tabla Secuencias almacenada en MongoDB

Job Generar Resultado

Entradas

Tabla Lectura

ID_NODO	ID:LECTURA

Tabla Secuencias

ID_LECTURA	CODIGO_GENOMA

Proceso

Por cada Lectura de la tabla Lectura irá a buscar en la tabla Secuencias el código del genoma al que pertenece dicha lectura.

Salida

Se obtendrá la tabla Resultado que tendrá el siguiente esquema:

Cuadro 7: Estructura de la tabla Resultados almacenada en la base de datos

ID_NODO	ID_LECTURA	CODIGO_GENOMA

Fuente: Elaboración propia

La información de la tabla Resultados se describe a continuación:

Campos	**Descripción**
Id:Nodo	Número del nodo, nos sirve para identificar un nodo en específico.
Id_lectura	Es el id de una de las lecturas por las que está conformado un nodo.
Codigo_Genoma	Es la etiqueta que muestra a que genoma pertenece una lectura en específico.

Una vez se tiene lista la tabla **Resultados**, se procede a ejecutar en la base de datos la siguiente operación la que nos permitirá agrupar los datos de la tabla Resultado.

```
db.Log.aggregate([
{$group:{ _id : {ID_NODO:"$ID_NODO",
CODIGO_GENOMA:"$CODIGO_GENOMA"}, count: {$sum:1}}},
 {$out:'nodos_agrupados'}], {  allowDiskUse : true}).objsLeftInBatch();
```

Ejecutamos también la siguiente orden para agrupar nuevamente:

```
db.nodos_agrupados.aggregate([
```

{$group:{ _id : '$_id.ID_NODO', total: {$sum:1}}}, {$out:'nodos_clasificados'}])

Realizar esta agrupación nos permite tener un acceso más rápido a la base de datos para realizar las consultas

3.2. Visualización de los grafos generados

Cytoscape nos permite crear grafos de muchas maneras, en este caso como ya tenemos los datos generados (LastGraph), necesitamos crear un script para transformar el archivo generado por Velvet en un archivo que pueda ser utilizado por Cytoscape.

Para esto se realizó el siguiente script en lenguaje perl:

buscaSec.pl | LastGraph.pl

```
die "perl $0 LastGraph\n" if ($#ARGV!=0) or (!$ARGV[0]=~/LastGraph/);
#open LastGraph file
open(F,"<$ARGV[0]");
while(<F>){
if (/^ARC/){
chomp;
@t=split(/\t/);
$t1=abs($t[1]);
$t2=abs($t[2]);
$start=$t1<$t2?$t1:$t2;
$end=$t1>$t2?$t1:$t2;
$h{$t1}->{$t2}=$t[3];
$hn{$t1}++;
}
}
close F;
open(O,">LastGraph2.sif");
foreach $x(sort keys %h){
#next if $hn{$x}>3;
foreach $y(sort keys %{$h{$x}}){
#next if $hn{$y}>3;
print O "$x pp $y\n";
}
```

Figura 41: Script en perl transforma archivo Graph a .sif.

Fuente: Elaboración propia.

Este script nos permite utilizar el archivo LastGraph generado por Velvetg y lo transforma a un archivo .sif que es el formato en que trabaja Cytoscape.

Para la utilización este script ejecutamos en la consola lo siguiente:

perl LastGraph.pl LastGraph

En donde LastGraph.pl es el archivo programado en perl y LastGraph es el archivo de Velvetg

Esto transformará el archivo LastGraph en un archivo .sif el cual quedaría de la siguiente manera:

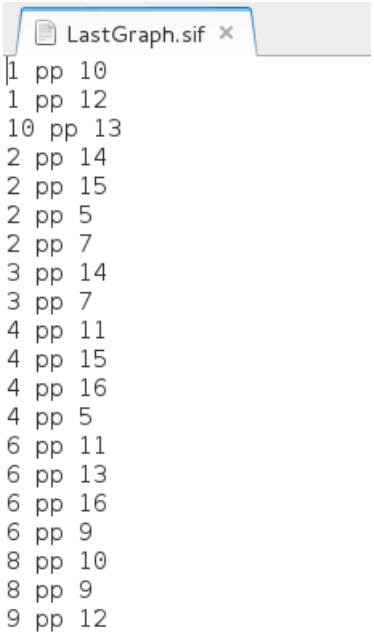


Figura 42: Archivo .sif generado con script en perl

Fuente: Elaboración propia.

Ahora para poder visualizar los datos generados realizamos lo siguiente:

- Ejecutamos el Cytoscape
- Nos vamos a File e importamos el Archivo LastGraph.sif

Cytoscape nos permite visualizar los datos de la siguiente manera:

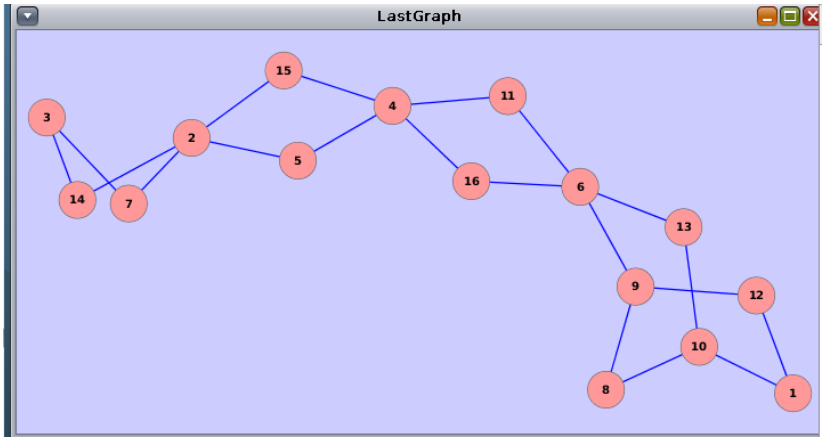


Figura 43: Archivo LastGraph visualizado con Cytoscape

Fuente: Elaboración propia.

Haciendo una comparación entre el archivo LastGraph generado por Velvetg y nuestro grafo generado con Cytoscape podemos observar que el gráfico ha sido construido correctamente ya que las conexiones entre los 16 nodos mostrados en el archivo LastGraph corresponden con la información mostrada con el Cytoscape.

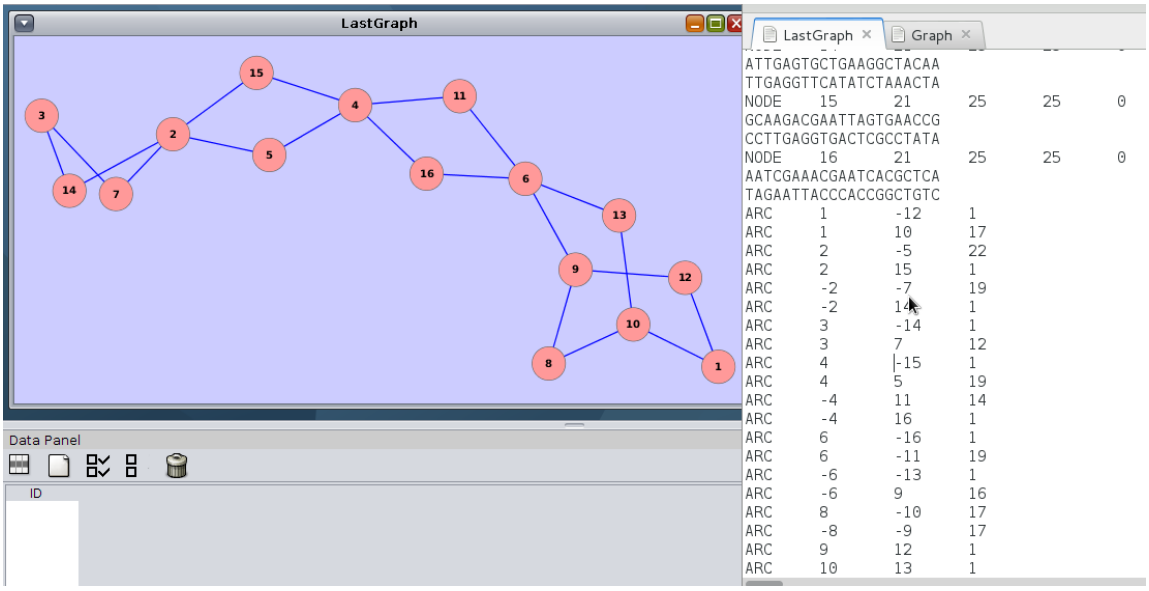


Figura 44: Comparación ente grafo creado y archivo LastGraph

Fuente: Elaboración propia.

El gráfico muestra la representación de de los 16 nodos de este ejemplo, pero los datos con los que trabajamos van en el orden de decenas de miles de nodos, razón

por la cual una representación gráfica de los nodos no nos permite comprender adecuadamente las propiedades del grafo.

Realizamos la visualización de los nodos en el Cytoscape y se obtiene lo siguiente:

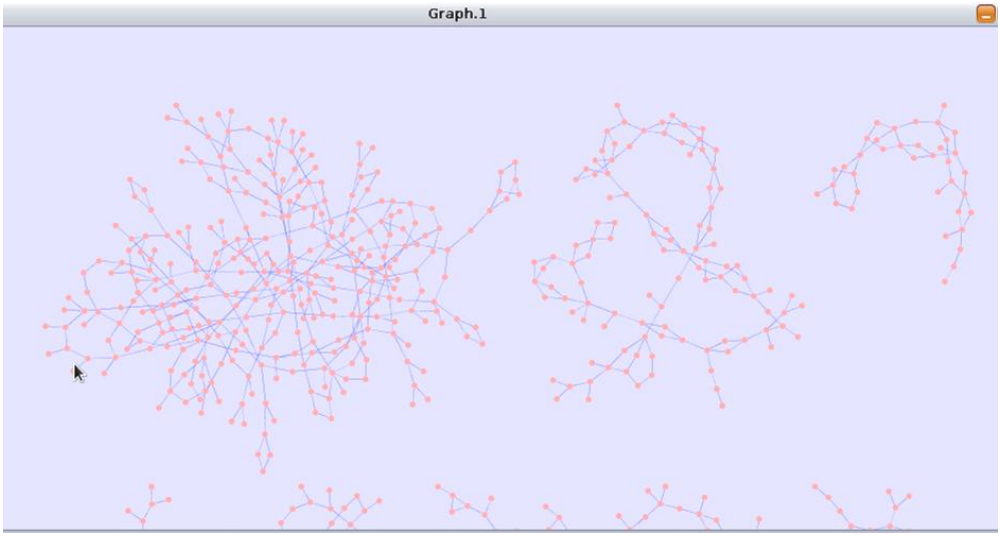


Figura 45: Visualización de una parte del grafo trabajando con 2 nodos

Fuente: Elaboración propia.

Debido a la cantidad de nodos generados, la visualización de los grafos en Cytoscape no nos permite realizar un análisis de los mismos, por tal motivo se utiliza el almacenamiento de los datos en una base de datos.

3.3. Extracción de Propiedades Topológicas

Con los datos almacenados en la base de datos podemos realizar consultas tales como:

Cantidad de nodos: Son todos los generados, para obtenerlos se realiza la siguiente consulta:

db.nodos_clasificados.count()

Cantidad de Nodos Puros: Son los nodos que están formados por secuencias de un solo genoma, para obtenerlos se realiza la siguiente consulta:

db.nodos_clasificados.count({total:1})

Cantidad de Nodos Impuros: Son los nodos que están formados por secuencias de 2 o más genomas, para obtenerlos se realiza la siguiente consulta:

db.nodos_clasificados.count({total:n}) //reemplazar n por número de genomas 1, 2 3 4.

Cantidad de Nodos Quimera: Son los nodos cuyo grado de entrada es mayor a 1.

Para poder hacer las consultas en la base de datos MongoDb utilizamos una aplicación para la administración de este tipo de bases de datos llamada Robomongo, la misma que tiene el siguiente aspecto:

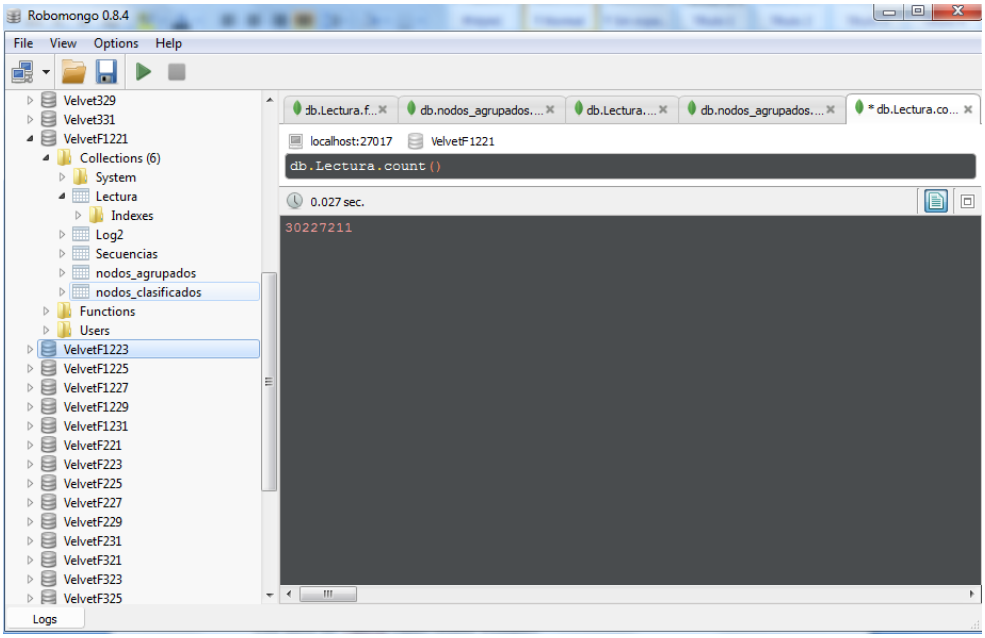


Figura 46: Visualización de base de datos a través de Robomongo

Fuente: Elaboración propia.

En la parte izquierda tenemos las bases de datos creadas a partir de los archivos generados por Velvet.
En la parte derecha podemos escribir las consultas para la base de datos.
En la parte inferior se visualizan los resultados de las consultas.

3.4. Datos obtenidos

Con el almacenamiento de los datos de los grafos generados en la base de datos procedemos a realizar las consultas descritas en el literal anterior y obtenemos los siguientes resultados:

Total de nodos generados

Los cuadros 7 y 8 muestran la cantidad de nodos generados en cada simulación, tanto con la ejecución de Velvet y VelvetLite respectivamente.

Cuadro 8: Nodos generados con Velvet

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	36417	24526	18333	14603	12695	11223
3	76843	46474	34240	27219	23479	19651
6	496034	306203	214921	162643	141192	107723
12	1606877	1158669	889375	718303	643067	529776

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Cuadro 9: Nodos Generados con VelvetLite

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	933379	843281	765164	694036	628558	561191
3	1421957	1279810	1160052	1051185	951492	848853
6	3153401	2766296	2472819	2220657	2007972	1779373
12	6939309	6112259	5427914	4852316	4415548	3916287

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Nodos puros generados

Los cuadros 9 y 10 muestran la cantidad de nodos puros generados en cada simulación, tanto con la ejecución de Velvet y VelvetLite respectivamente.

Cuadro 10: Nodos puros generados con Velvet

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	36044	24312	18185	14504	12611	10158
3	72764	44488	33141	26516	22931	19244
6	396803	239553	169002	129086	112877	86758
12	1059380	732309	556644	448843	401108	328507

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Cuadro 11: Nodos puros generados con VelvetLite

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	932789	842881	764882	693836	628390	561061
3	1417097	1277244	1158522	1050139	848277	950687
6	3023735	2676578	2409462	2174020	1969568	1750867
12	5783902	5150681	4632118	4180291	3816224	3403973

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Nodos Impuros Generados

Los cuadros 11 y 12 muestran la cantidad de nodos impuros generados en cada simulación, tanto con la ejecución de Velvet y VelvetLite.

Cuadro 12: Nodos impuros generados con Velvet

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	373	214	146	99	84	1065
3	4079	1986	1099	703	548	407
6	99231	66650	45919	33557	28315	20965
12	**547497**	**426360**	**332731**	**269460**	**241959**	**201269**

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Cuadro 13: Nodos impuros generados con VelvetLite

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	590	400	282	200	168	130
3	4860	2566	1530	1046	805	576
6	129666	89718	63357	46637	38404	28506
12	1155407	961578	795796	672025	599324	512314

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Nodos Quimera Generados

Los cuadros 13 y 14 muestran la cantidad de nodos quimera generados en cada simulación, tanto con la ejecución de Velvet y VelvetLite.

Cuadro 14: Nodos quimera generados con Velvet

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	9311	6265	4670	3735	3239	5672
3	19635	11780	8729	6972	6023	5084
6	121501	74730	52765	40099	34855	26551
12	390788	281702	217195	176294	157376	129823

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

Cuadro 15: Nodos quimera generados con VelvetLite

N^{ro} de genomas	**Valores de k**					
	21	**23**	**25**	**27**	**29**	**31**
2	243681	223200	205225	188574	172924	156589
3	368472	336609	309263	284019	260223	235384
6	800668	717040	652852	595846	546002	491583
12	1574667	1469333	1385661	1312641	1250746	1179417

Fuente: Elaboración propia en base a las consultas realizadas a la base de datos

CAPÍTULO 4

4. Análisis de resultados

Luego de haber extraído los datos generados por Velvet y por nuestra versión modificada de Velvet (VelvetLite) procedemos a construir las siguientes gráficas que serán analizadas a continuación:

TOTAL DE NODOS GENERADOS

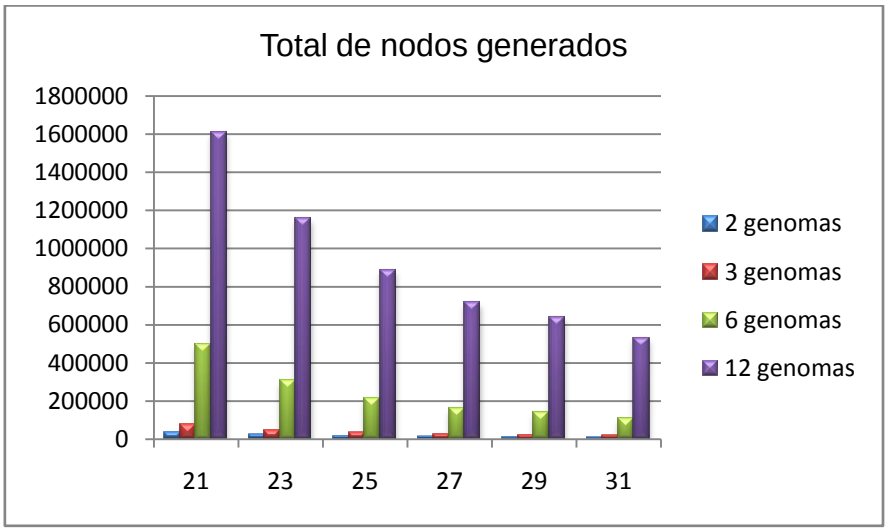


Figura 47: Cantidad de nodos generados con Velvet

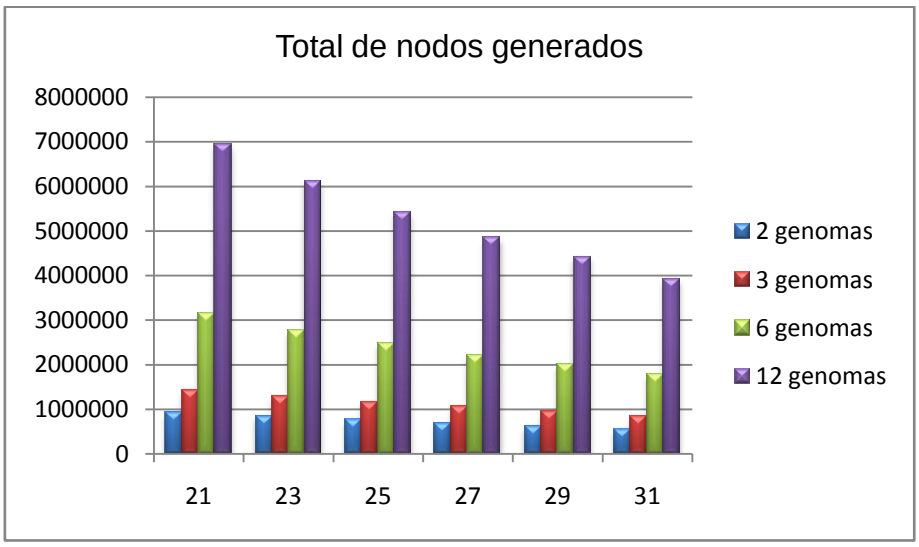


Figura 48: Total de nodos generados con VelvetLite

La cantidad de nodos generados en directamente proporcional al número de genomas e inversamente proporcional al valor de k empleado.

NODOS PUROS GENERADOS

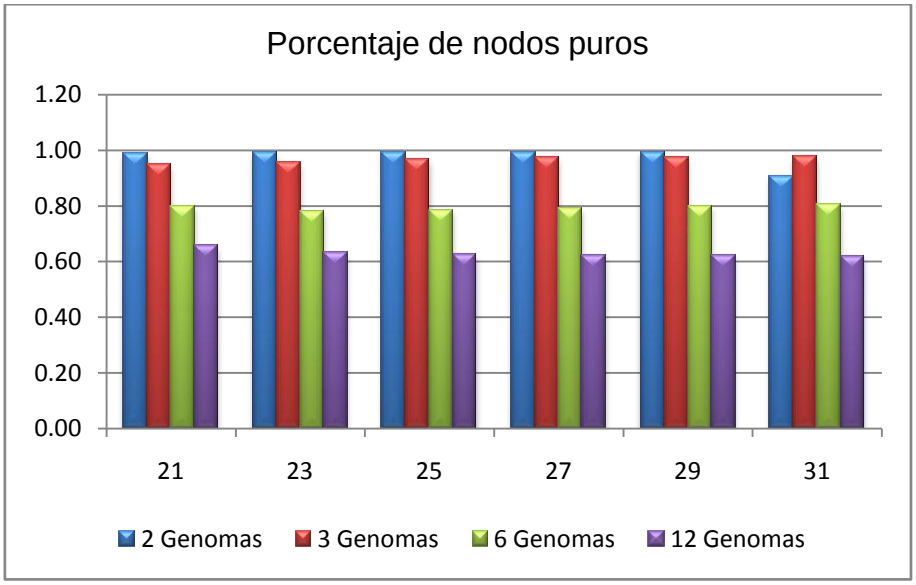


Figura 49: Porcentaje de nodos puros generados con Velvet

Podemos observar que a medida que aumenta el número de genomas presentes, disminuye el porcentaje de nodos puros generados. Por lo tanto la cantidad de nodos puros es inversamente proporcional con el número de genomas con el que se trabajen.

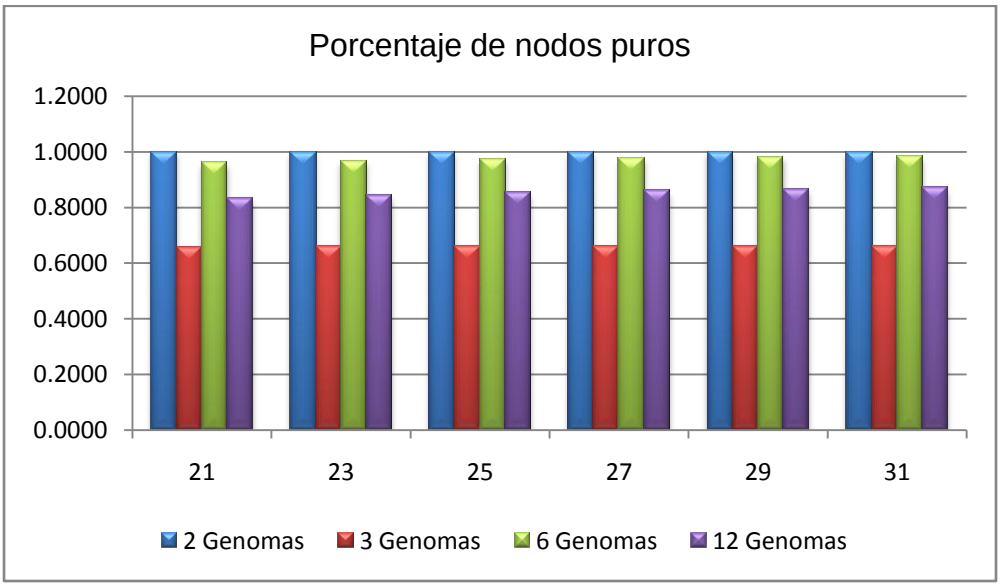


Figura 50: Porcentaje de nodos puros generados con VelvetLite

La figura 50 indica que con el uso de VelvetLite, el mayor porcentaje de nodos puros se dá cuando el número de genomas es 2, y el menor porcentaje se produce cuando el

número de genomas es 3, Tanto para 6 y 12 genomas el porcentaje de nodos puros es sobre el 80%.

NODOS IMPUROS GENERADOS

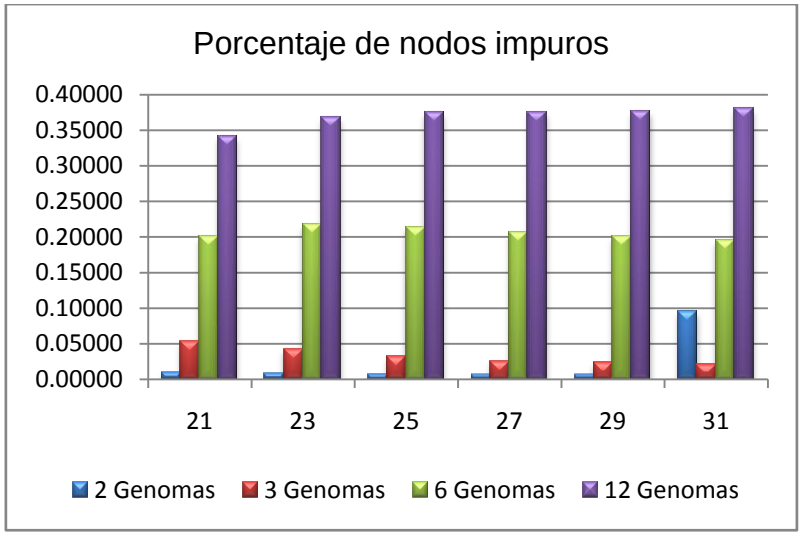


Figura 51: Porcentaje de nodos impuros generados con Velvet

La gráfica 51 nos muestra que el porcentaje de nodos impuros es directamente proporcional al número de genomas, siendo cercanas al 1% cuando se trata de 2 genomas, y llegando al 35% cuando se trata de 12 genomas. Esto ocurre con Velvet.

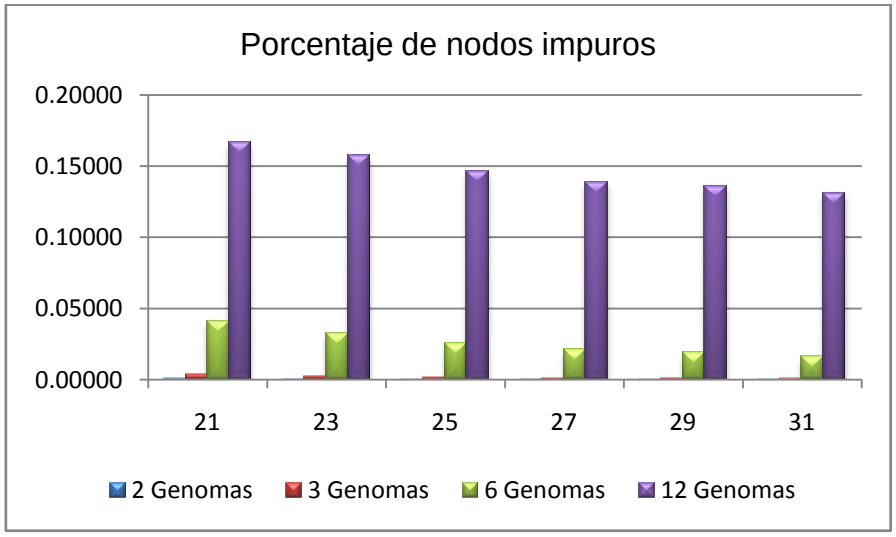


Figura 52: Porcentaje de nodos impuros generados con VelvetLite

Al igual que con la utilización de Velvet, con el uso de VelvetLite el número de nodos impuros generados es directamente proporcional al número de genomas. Pero su

porcentaje con respecto al total de nodos es casi nulo con 2 y 3 genomas, siendo a lo mucho del 16% cuando se trata de 6 y 12 genomas.

NODOS QUIMERA GENERADOS

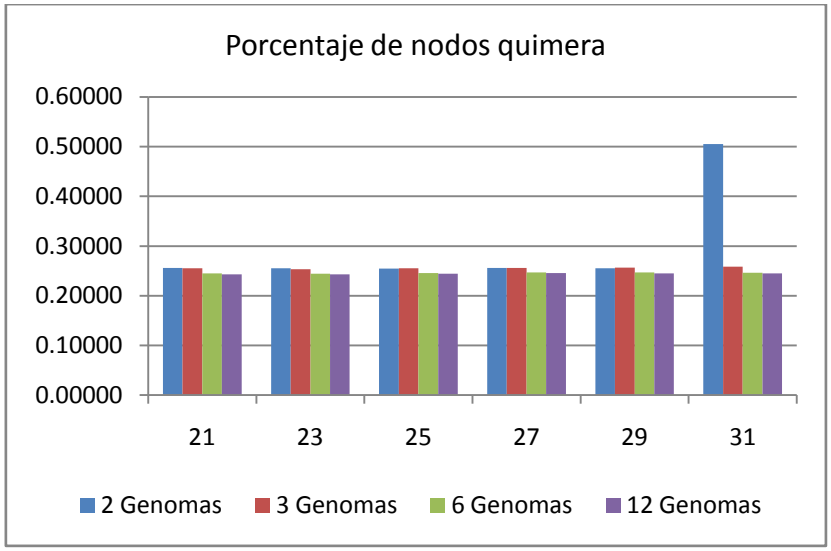


Figura 53: Porcentaje de nodos impuros generados con VelvetLite

El porcentaje de nodos quimera es casi el mismo ya sea que se trabajen con 2, 3. 6 ó 12 genomas. El porcentaje de nodos quimera con Velvet es del 26%

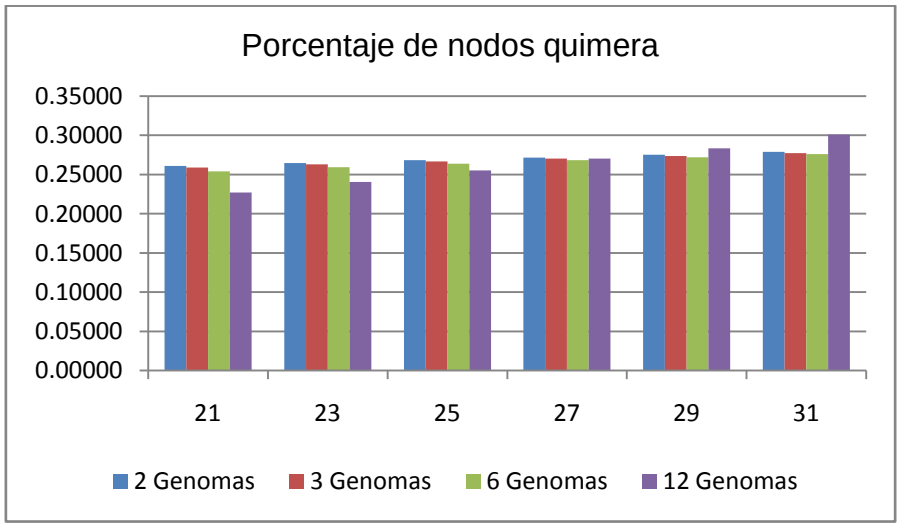


Figura 54: Porcentaje de nodos quimera generados con VelvetLite

El porcentaje de nodos quimera generados con VelvetLite es casi el mismo, sin importar el número de genomas. El porcentaje de nodos quimera es de alrededor de 26%.

Tiempos de ejecución

La ejecución del ensamblador se realizó en un servidor del laboratorio de Química, el cual tiene las siguientes características:

Procesador: Core i5 3470 3.20GHz

Memoria Ram: 4 GB

Núcleos: 4

Disco Duro: 1 TB + disco externo de 2TB

Sistema Operativo: CentOS Release 6.3 Final Kernel Linux 2.6.32-279.14.1 GNOME

Los tiempos de ejecución del ensamblador medidos en minutos son los siguientes:

Cuadro 16: Tiempo (minutos) de ejecución del ensamblador **Velvet**

Nro Genomas	**velvetgh**	**velvetg**
2	6.99	10.02
3	11.91	23.90
6	33.41	59.08
12	493.03	121.19
TOTAL	545.34	214.19

Fuente: Elaboración propia en base a los tiempos tomados en la ejecución del ensamblador.

Cuadro 17 Tiempo (minutos) de ejecución del ensamblador **VelvetLite**

Nro Genomas	velvetgh	velvetg
2	6.99	14.67
3	11.91	25.41
6	33.41	34.19
12	493.03	161.31
TOTAL	545.34	235.58

Fuente: Elaboración propia en base a los tiempos tomados en la ejecución del ensamblador.

Los tiempos son los mismos en el proceso que ejecuta Velveth, pero los tiempos de ejecución de Velvetg varían ya que en este realizamos los cambios en el código.

El tiempo de ejecución del ensamblador con cada conjunto de genomas corresponde a la suma de todos los tiempos con cada valor de k(21,23,25,27,29,31) que se utilizó.

CONCLUSIONES

- El estudio de la estructura y propiedades de los De Bruijn Graphs generados a partir de datos simulados de secuenciación metagenómica nos permitió extraer la información del grafo resultante y con esta lograr identificar los genomas individuales presentes.

- El porcentaje de nodos puros (secuencias ensambladas con información de un solo genoma) por el que está formado el grafo de Bruijn resultante, disminuye, si hay un mayor número de genomas presentes, pero se mantiene constante para cualquier valor que le asignemos a k (tamaño en que se dividen las lecturas iniciales), por lo que fue necesario trabajar con una cantidad menor de genomas si se desea obtener un porcentaje mayor de nodos puros y con esto conseguir que las secuencias hayan sido ensambladas en el orden correcto para cada genoma.

- La extracción de las propiedades topológicas del grafo de Bruijn resultante y su almacenamiento en una base de datos nos permitió clasificar y cuantificar los nodos generados; ya que debido a que la cantidad de nodos va en el orden de los millones, su visualización a través de una gráfica no nos permitía apreciar las propiedades del mismo ni diferenciar a qué genoma pertenecían.

- La construcción de un conjunto de Jobs utilizando la herramienta Talend Open Studio for Big Data sirvió para automatizar todo el proceso de extracción, manipulación y posterior almacenamiento en una base de datos de las propiedades del grafo de Bruijn resultante.

RECOMENDACIONES

- Para las simulaciones utilizar un equipo de escritorio que cuente con los siguientes requisitos mínimos: un procesador i5; una memoria RAM de por lo menos 4 GB para la ejecución del ensamblador y mayor a 6 GB si se desea visualizar los nodos con aplicaciones como Cytoscape o Gephi; también debe tener un disco duro de 1TB para el almacenamiento de los datos generados y un sistema operativo para servidor ya sea Windows o Linux.

- Por la gran cantidad de datos que se debe procesar es mejor trabajar con una base de datos NoSQL como MongoDB, ya que este tipo de base de datos se centran más en almacenar y consultar grandes cantidades de datos y resulta más rápida que trabajar con bases de datos como mySQL.

- Para almacenar la gran cantidad de datos generados se necesitó un disco de almacenamiento adicional, por lo cual se podría revisar con mayor detalle los datos que se guardan en la base de datos y así analizar de cuales se podría prescindir.

- Familiarizarse con conceptos del área biológica como genomas, ADN, Metagenómica, secuenciación etc., ayudará a tener una mejor comprensión de los procesos estudiados y de los resultados obtenidos.

REFERENCIAS

1. Ainhoa Ogiza, A. G. (19 de Abril de 2010). *Ciencia y tecnología de la Fundación Telefónica: Secuenciación de Nueva Generación*. Recuperado el 5 de Noviembre de 2014, de http://blogs.creamoselfuturo.com/bio-tecnologia/2010/04/19/secuenciacion-de-nueva-generacion/

2. Arif Bilgin, J. E. (2014). *Graphviz-graph visualization software.* Recuperado el 21 de Julio de 2014, de http://www.graphviz.org/

3. Blanco Diez, A. (julio de 2013). *Archivo Digital Universidad Politécnica de Madrid: Implementación de algoritmos de ensamblaje de genomas en sistemas de memoria compartida y memoria distribuida.* Recuperado el 14 de junio de 2014, de http://oa.upm.es/21934/1/TESIS_MASTER_ADOLFO_BLANCO_DIEZ.pdf

4. Bonilla-Rosso, G., Souza, V., & Eguiartea, L. E. (1 de junio de 2008). *Red de Revistas Científicas de América Latina y el Caribe, España y Portugal.* Recuperado el Diciembre de 2014, de http://www.redalyc.org/articulo.oa?id=43211943005

5. Cañizares Sales, J., & Forment Millet, J. J. (2014). *Bioinformatics at COMAV: Ensamblaje de secuencias*. Recuperado el 4 de Octubre de 2014

6. Cytoscape Consortium. (2014). *Cytoscape.* Recuperado el 20 de agosto de 2014, de http://www.cytoscape.org/

7. Daniel R. Zerbino, E. B. (marzo de 2008). *Genome Research: Velvet. Algorithms for de novo short read assembly using de Bruijn graphs.* Recuperado el 1 de junio de 2014, de http://genome.cshlp.org/content/18/5/821.full.pdf+html

8. Diez, A. B. (Julio de 2013). *Universidad Politécnica de Madrid, Implementación de algoritmos de ensamblaje de genomas en sistemas de memoria compartida y distribuida.* Recuperado el 30 de Julio de 2014, de http://oa.upm.es/21934/1/TESIS_MASTER_ADOLFO_BLANCO_DIEZ.pdf

9. gephi.org. (2014). *The Open Graph Viz Platform.* Recuperado el 30 de julio de 2014, de https://gephi.github.io/

10. Hernández-León R, I. V.-S.-M. (2010). *REVISTA INTERNACIONAL DE BOTÁNICA EXPERIMENTAL.* Recuperado el 15 de octubre de 2014, de http://www.revistaphyton.fund-romuloraggio.org.ar/vol79/Hernandez-Leon.pdf

11. Informatik, A. A. (s.f.). *aiSee — Graph Visualization.* Recuperado el 2014 de julio de 20, de http://www.absint.com/aisee/

12. Joaquin Cañizares, J. M. (s.f.). *Bioinformatica. Grado de Biotecnologia.* Recuperado el 10 de diciembre de 2014, de http://personales.upv.es/jcanizar/bioinformatica/ensamblaje.html

13. John C. Wooley, A. G. (13 de Diciembre de 2011). *PLOS Computational Biology: A Primer on Metagenomics*. Recuperado el 29 de Julio de 2014, de http://www.ploscompbiol.org/article/info:doi/10.1371/journal.pcbi.1000667

14. Lin Liu, Y. L. (2 de Abril de 2012). *BioMed Research International: Comparison of Next-Generation Sequencing Systems.* Recuperado el 10 de Agosto de 2014, de http://downloads.hindawi.com/journals/bmri/2012/251364.pdf

15. National Human Genome Research Institute. (13 de 10 de 2011). *National Human Genome Research Institute*. Recuperado el 24 de 2 de 2015, de http://www.genome.gov/11510905#al-2

16. Restrepo Restrepo, S., González, A., & Cárdenas, M. (12 de febrero de 2012). *Universidad de los Andes, Facultad de Ciencias.* Recuperado el 25 de julio de 2014, de http://hipotesis.uniandes.edu.co/hipotesis/images/stories/ed12pdf/Metagenomica.pdf

17. Santamaría, R. (13 de Diciembre de 2011). *VisUsal Research Group at the University of Salmanca: Secuenciación.* Recuperado el 14 de octubre de 2014, de http://vis.usal.es/rodrigo/documentos/bioinfo/temas/8_Secuenciaci%C3%B3n.pdf

18. The Computer Architecture Department at the University of Malaga. (18 de Noviembre de 2010). *Bioinfomatics and Information Technologies Laboratory.* Recuperado el 20 de Diciembre de 2014, de http://chirimoyo.ac.uma.es/hpcngs/public/Estad-NGS.pdf

19. Toshiaki N, T. H. (7 de 2012). *National Center for Biotechnology Information: MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads.* Recuperado el 30 de julio de 2014, de http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3488206/pdf/gks678.pdf

20. Universidad de Málaga, Moreno, Rocío Bautista. (Mayo de 2010). *Plataforma Andaluza de Bionformática, Universidad de Málaga.* Recuperado el 20 de Mayo de 2015, de http://www.encuentros.uma.es/encuentros128/comofunciona128.pdf

21. Zerbino, B. (14 de abril de 2011). *Velvet*. Recuperado el 20 de mayo de 2014, de http://www.ebi.ac.uk/~zerbino/velvet/

ANEXOS

INSTALACIÓN DEL ENSAMBLADOR VELVET PARA SU ANÁLISIS CON NETBEANS

Para conseguir el código del ensamblador Velvet debemos ir a la página oficial https://www.ebi.ac.uk/~zerbino/velvet/ en la que podemos encontrar lo siguiente:

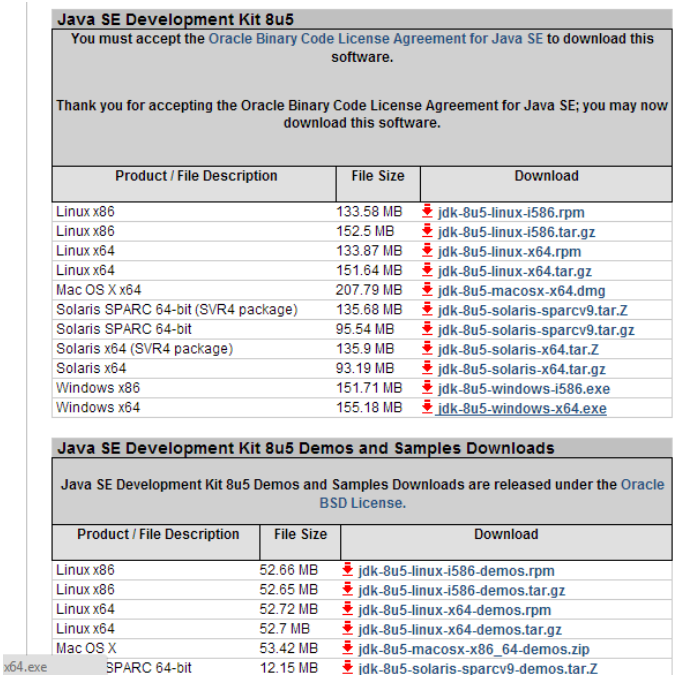
- La última versión que es la 1.2.10 está en un archivo .zip
- Manual en pdf

Descargamos el archivo .zip, en el podemos encontrar la documentación de Velvet, así como también el código en c.

Para revisar el código se utilizó el IDE Netbean en su versión 7.4.

La instalación de Netbeans consiste en los siguientes pasos:

Instalar el JDK (Java SE Development Kit) desde http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html



Java SE Development Kit 8u5

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u5-linux-i586.rpm
Linux x86	152.5 MB	jdk-8u5-linux-i586.tar.gz
Linux x64	133.87 MB	jdk-8u5-linux-x64.rpm
Linux x64	151.64 MB	jdk-8u5-linux-x64.tar.gz
Mac OS X x64	207.79 MB	jdk-8u5-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.68 MB	jdk-8u5-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	95.54 MB	jdk-8u5-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.9 MB	jdk-8u5-solaris-x64.tar.Z
Solaris x64	93.19 MB	jdk-8u5-solaris-x64.tar.gz
Windows x86	151.71 MB	jdk-8u5-windows-i586.exe
Windows x64	155.18 MB	jdk-8u5-windows-x64.exe

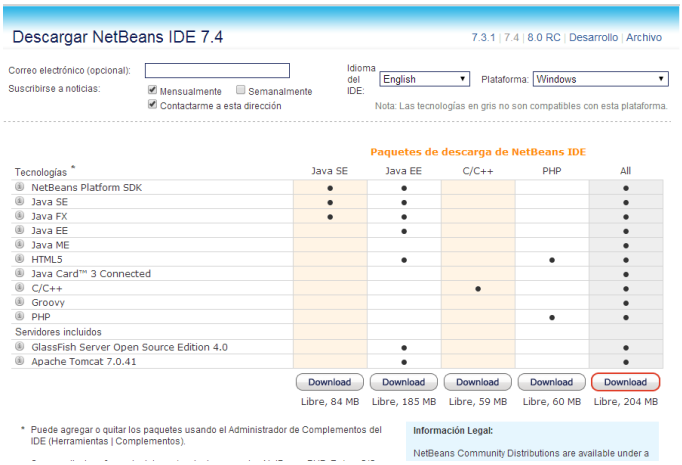
Java SE Development Kit 8u5 Demos and Samples Downloads

Java SE Development Kit 8u5 Demos and Samples Downloads are released under the Oracle BSD License.

Product / File Description	File Size	Download
Linux x86	52.66 MB	jdk-8u5-linux-i586-demos.rpm
Linux x86	52.65 MB	jdk-8u5-linux-i586-demos.tar.gz
Linux x64	52.72 MB	jdk-8u5-linux-x64-demos.rpm
Linux x64	52.7 MB	jdk-8u5-linux-x64-demos.tar.gz
Mac OS X	53.42 MB	jdk-8u5-macosx-x86_64-demos.zip
x64.exe SPARC 64-bit	12.15 MB	jdk-8u5-solaris-sparcv9-demos.tar.Z

Escogemos el jdk según el sistema operativo que utilicemos

- Descargar e instalar el IDE Netbeans 7.4 desde https://netbeans.org/downloads/7.4/



Escogemos el paquete que tiene soporte para C/C++ ya que el código de Velvet está escrito en C

Si el sistema operativo con el que trabajaremos es Windows, debemos instalar la herramienta cygwin desde https://www.cygwin.com/install.html, pero si se usa linux se trabaja con el GNU.

Esto nos permitirá descargar, instalar, y configurar el soporte(compilador, debuger y make) para C/C++. Entonces tendremos los siguientes componentes instalados:

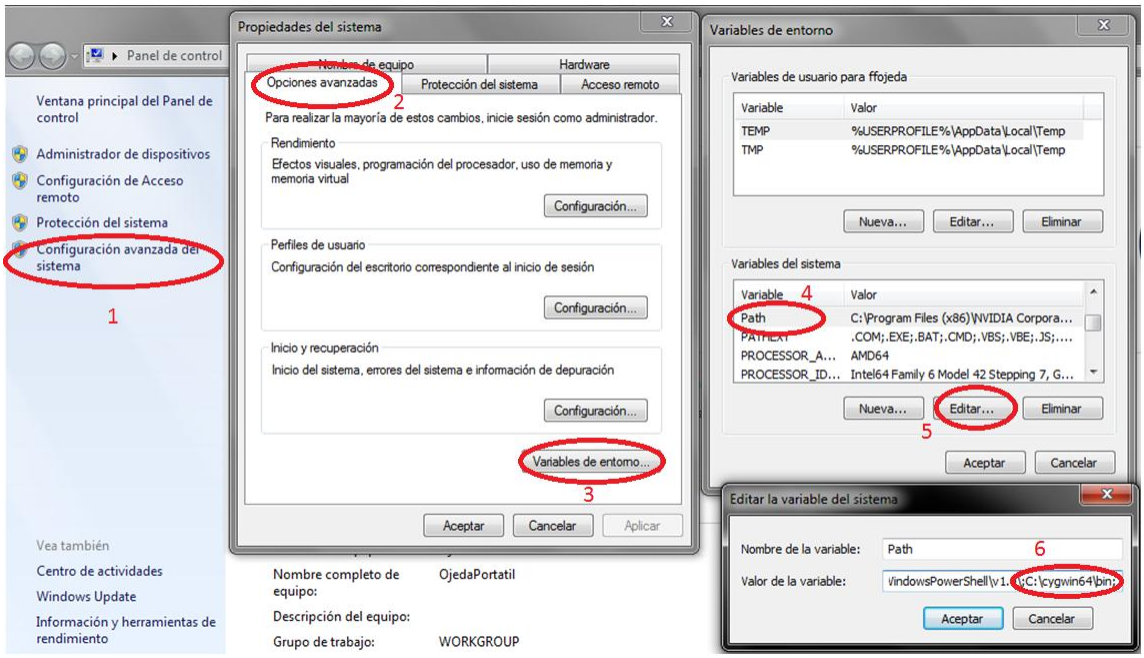
- g++, el GNU C++ compiler,
- gdb, el GNU debugger, y make

Para los usuarios de Windows al momento de instalar el cygwin seleccionamos cada paquete que deseamos descargar haciendo click en la etiqueta Skip al lado de cada paquete. Como mínimo, debemos seleccionar los siguientes componentes:

- gcc-core: el compilador C
- gcc-g++: el compilador C++
- gdb: el depurador GNU
- make: la versión GNU de la utilidad 'make'.

Ahora hay que añadir en el path el directorio donde se instaló el Cygwin para que Netbeans encuentre las herramientas. Para esto hacemos nos vamos a propiedades de equipo y hacemos lo siguiente:

- Clic derechos sobre equipo, vamos a propiedades
- Escogemos configuración avanzada del sistema.
- Vamos a opciones avanzadas, escogemos variables de entorno.
- En variables del sistema escogemos la variable “Path”.
- Presionamos el botón editar
- Añadimos el directorio donde se instaló el Cygwin, en mi caso C:\cygwin64\bin; como se muestra en la imagen a continuación.



Añadir el Path al sistema en Windows 7.

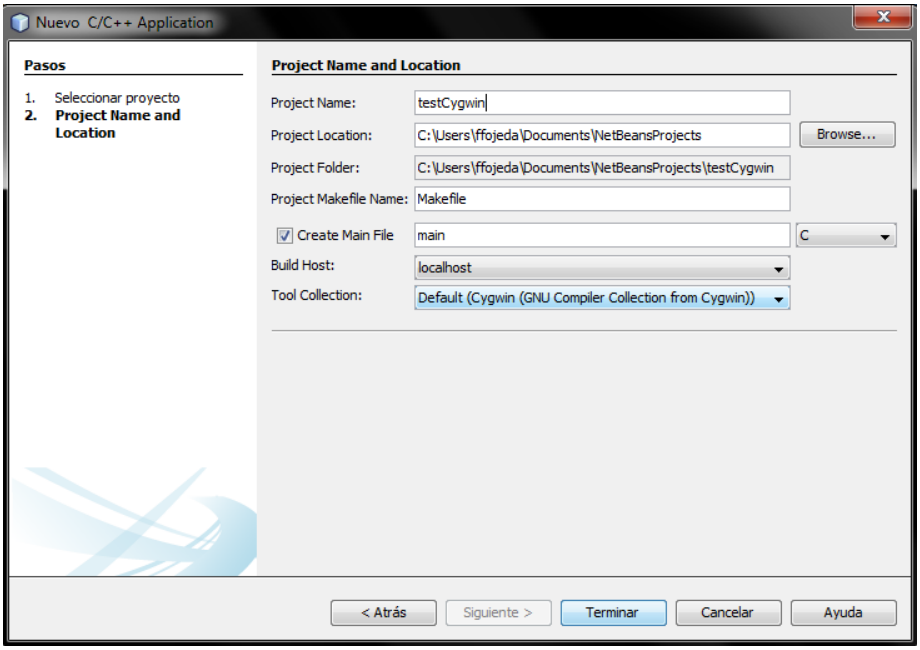
Para verificar que estas herramientas están instaladas correctamente hacemos lo siguiente:

Abrir Netbeans

Crear un nuevo proyecto C/C++

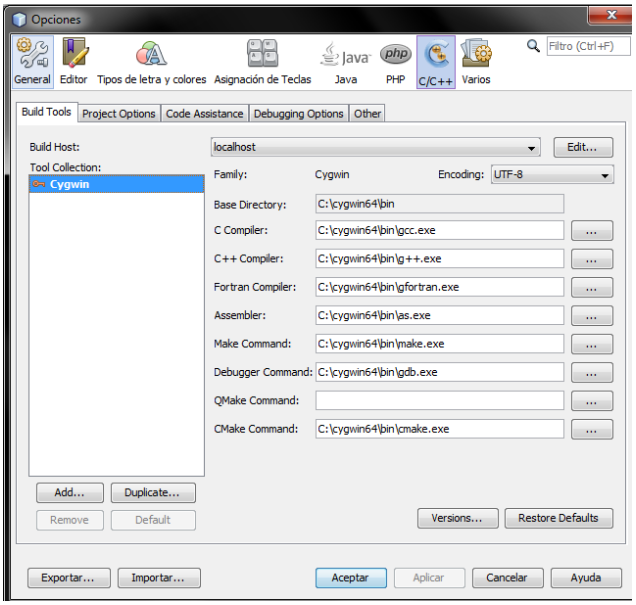
Nos debe aparecer una imagen como la siguiente:

:



En la cual ya podemos observar que se encuentra el compilador GNU de Cygwin, damos clic en terminar y se crea el proyecto.

Cómo último paso vamos a Herramientas, opciones y debemos tener una ventana como esta:



En la que nos muestra que Netbeans está configurado correctamente con el Cygwin

VELVET

Es el núcleo del ensamblador. Es el programa encargado de construir el Grafo de Bruijn y manipularlo para conseguir ensamblar las lecturas que hemos pasado por parámetro

Ya que necesitamos analizar el código del ensamblador en Netbeans creamos un nuevo proyecto para utilizar los archivos de Velvet descargados siguiendo los siguientes pasos:

Creamos un nuevo proyecto en Netbeans.

Escogemos en Categoría C/C++.

Proyecto C/C++ Aplicación.

Le damos un nombre al proyecto en mi caso Velvet_1

Desmarcamos la casilla de “crear main file”.

En tool Collection escogemos “Cygwin” y terminar

En el directorio donde se creó el proyecto de Netbeans copiamos los arhivos .c, .h y las carpetas data y third-party que vienen en la carpeta de velvet

En netbeans nos vamos al proyecto creado y añadimos en la el directorio Header Files todos los archivos .h

También añadimos en el directorio Source Files todos los archivos .c.

Con lo que hemos hecho hasta ahora si vamos a Clean and Build nos va a dar algunos errores, los cuales han sido solucionado con los siguientes pasos.

Abrir el archivo globals.h y definir las variables MAXKMERLENGTH=31 y CATEGORIES=2

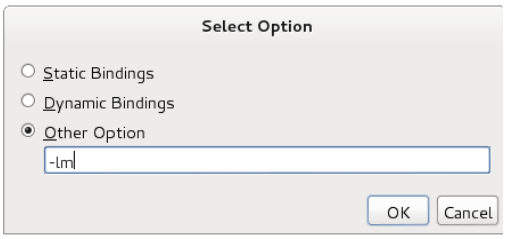
Añadimos al proyecto la carpeta third-party

Dentro del proyecto de Netbeans borramos el archivo example.o y la carpeta contrib.

Vamos al archivo binarySequences.c y en la linea 37 la editamos para que quede como lo siguiente: #include "./third-party/zlib-1.2.3/zlib.h".

Como el proyecto viene con 2 archivos main que son run.c y run2.c. Para velveth se necesita usar el archivo run.c, entonces excluimos del proyecto a run2.c.

El código de velvet utiliza funciones matemáticas, por lo cual debemos incluir la opción -lm en las librerias.



Incluir opción en librerias

Con esto hacemos un clean and build y ya no nos dará ningún error

Al ejecutar Velveth(run.c) empezamos a trabajar con las lecturas(archivos .fasta de la carpeta data) obtenidas a través de la máquina de secuenciación, estas lecturas son divididas en k-mers en las cuales el valor de k es definido por el usuario en el comando de ejecución del ensamblador.

El valor de k determinará la calidad del ensamblaje final, pero no hay una regla que nos pueda indicar que valores de k son aceptables para un genoma en concreto utilizando este ensamblador. Por ello hay que realizar pruebas exhaustivas con varios valores de k, hasta conseguir ver valores más o menos aceptables.

De todas formas, debido a comprobaciones ya realizadas en otros experimentos podemos indicar que los valores de k que son cercanos a la longitud de las lecturas que tenemos pueden producir superposiciones en el ensamblado mientras que valores de k más pequeños pueden producir un mayor número de superposiciones que pueden generar errores y bucles en la generación de la estructura de datos.

Una vez que las lecturas son divididas en k-mers se escanean, se transforman a un formato interno utilizado por Velvet y se guarda en un archivo denominado “Sequences”.

A continuación Velvet crea una tabla hash de n entradas y cada vez que un k-mer es leído se realiza un proceso de búsqueda en la tabla hash. Si el k-mer analizado no se encuentra en la tabla hash correspondiente se almacena el identificador de dicho k-mer y su posición. En cambio, si el k-mer analizado si es encontrado en la tabla hash, una referencia de este k-mer es almacenada en el archivo “RoadMaps”, que como podemos observar en el proceso, el fichero RoadMaps almacena aquellos k-mers que tienen coincidencias con lecturas anteriores. La tabla hash se almacena temporalmente en memoria mientras el archivo RoadMaps se guarda permanentemente.

La sintaxis para ejecutar velveth es la siguiente:

./velveth output_directory hash_length [[-file_format][-read_type] filename], en nuestro caso ejecutams con los siguientes parámetros:

./velvet output 21 -shortPaired data/test_reads.fa

Los parámetros que configuramos nos sirven para indicarle a velveth donde están las lecturas que se utilizarán(data/test_reads.fa), la carpeta(output) donde se generarán los resultados. Para saber que parámetros podemos utilizar revisaremos el Manual que viene en Velvet.

Velveth utiliza los archivos de lecturas para generar el archivo de;

Sequences y

Roadmaps,

Velvetg utiliza los archivos generados por Velveth para crear estos archivos:

Graph

Pregraph

Lastgraph

Se ha realizado las respectivas pruebas con un conjunto de datos de prueba que vienen en la carpeta del proyecto, como lo que a nosotros nos interesa es analizar cómo velvetg genera los grafos resultantes, empezamos por analizar el archivo run2.c(velvetg) en donde en forma general se puede identificar que realiza lo siguiente:

Velvetg lee el archivo de Roadmaps y el archivo de Sequences

Con estos datos se genera el Pregraph el cual está formado por 106331 prenodos.

Se realiza una concatenación de este Pregraph.

Se realiza una "limpieza" de estos prenodos recortando los prenodos puntas.

Al finalizar todas estas acciones el grafo queda con 205 nodos.

Luego de esto velvet importa el pregraph al archivo graph y lo escanea para encontrar las tuplas o k-mers en total son unas 101403 y clasifica la tabla de ocurrencia k-mers.

Las funciones que se han suprimido son las siguientes:

- Función "*clipTipsHard(Graph, lConserveLong)".* Esta Función de Velvet elimina los nodos punta del grafo de Bruijin.
- Función *" tourBus(startingNode)"* Esta Función elimina las burbujas del grafo.

Continuando con la revisión del código se analiza que en la línea 526 de run2.c se tiene el procedimiento correctgraph dentro del cual se ejecutan los procesos antes mencionados.

Después de esto velvetg utiliza el procedimiento concatenar para eliminar los nodos nulos, dando como resultado de este procedimientos la reducción del número de nodos a 16, esto a nivel de código es suprimido ya que al trabajar con varios genomas no se desea que se eliminen nodos que velvetg los toma como nodos inválidos.

Haciendo las modificaciones descritas se crea una nueva versión de Velvet a la que se ha denominado **VelvelLite** la cual servirá para nuestros fines ya que no elimina los nodos que si eliminaba velvetg, y de esta manera nos permitirá poder trabajar con el análisis de todos los nodos.

CYTOSCAPE

Cytoscape es una plataforma de programación de código abierto para bioinformática, permite visualizar redes moleculares, sus interacciones, etc.

A pesar de que Cytoscape fue diseñado originalmente para la investigación biológica, ahora es una plataforma general para el análisis y la visualización de redes complejas.

Para instalar Cytoscape, primero descargamos el programa desde la página oficial: http://www.cytoscape.org/download.html, llenamos el formulario de registro y descargamos las versión Cytoscape_2_8_3 disponible para Linux.

Para la instalación del programa desde la ventana de comandos en linux ejecutamos los siguientes comandos:

```
chmod a+x Cytoscape_2_8_3_unix.sh
sudo sh Cytoscape_2_8_3_unix.sh
```

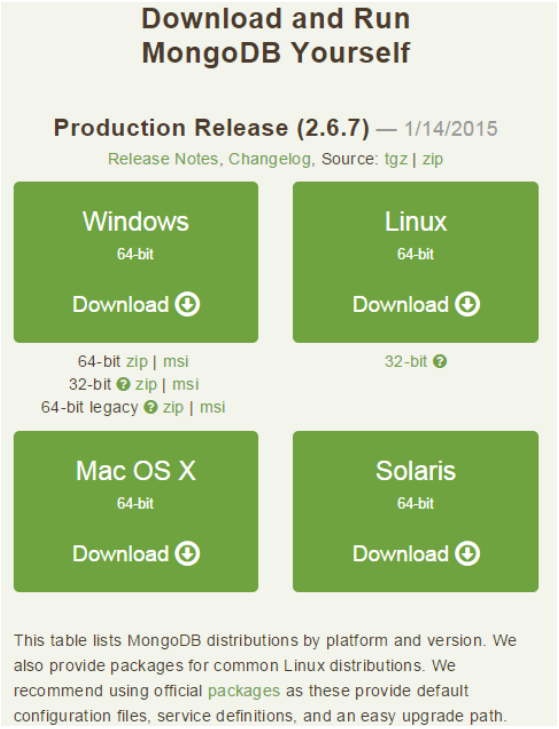
MONGODB

MongoDB (de la palabra en inglés “humongous” que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

El desarrollo de MongoDB empezó en octubre de 2007 por la compañía de software 10gen. Ahora MongoDB es una base de datos lista para la producción de uso y con muchas características. Esta base de datos es altamente utilizada en las industrias y MTV Network, Craiglist y Foursquare, son algunas de las empresas que utilizan esta base de datos.

El código binario está disponible para los sistemas operativos Windows, Linux, OS X y Solaris.



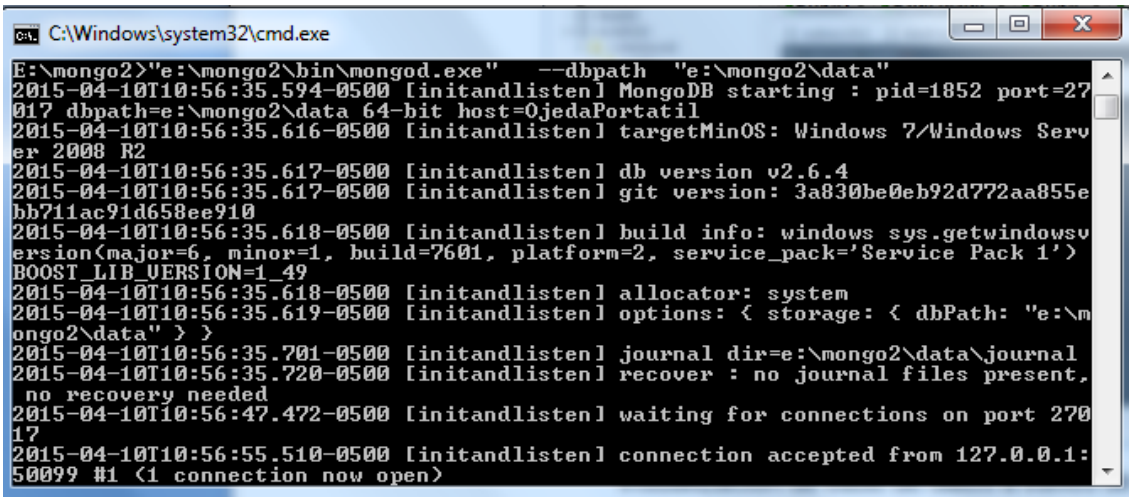
MongoDB

Uso de MongoDb

Vamos al directorio donde está instalado mongodb y ejecutamos lo siguiente:

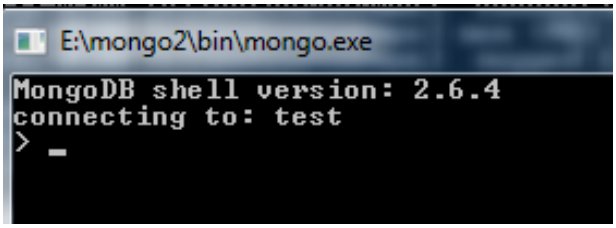
./bin/mongod --dbpath data

Nos mostrara esto:

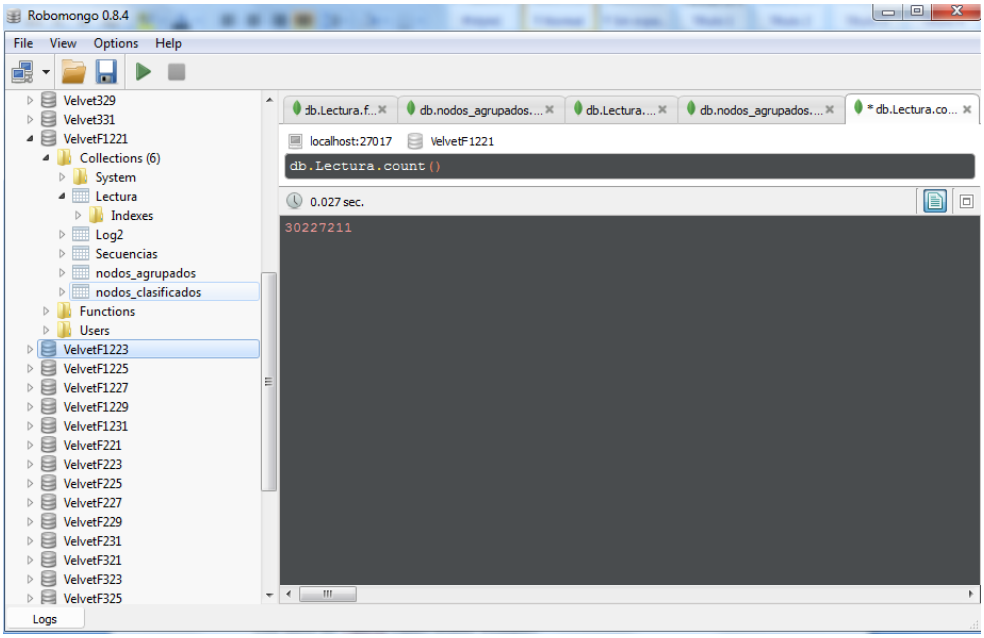


En ese mismo terminal ejecutamos:

./mongo



Ahora ya tenemos levantada la base de datos al cual accederemos mediante Robomongo y tiene este aspecto:



Visualización de base de datos a través de Robomongo

En la parte izquierda tenemos las bases de datos creadas a partir de los archivos generados por Velvet.

En la parte derecha podemos escribir las consultas para la base de datos.

En la parte inferior se visualizan los resultados de las consultas.

TALEND OPEN STUDIO

Es una herramienta open-source (licencia GPL) que permite la integración y gestión de datos, así como integración de aplicaciones empresariales. Se basa en Java, por lo tanto puede ser ejecutado en entornos Linux o Windows con tan solo descomprimir su instalador disponible en la web.

Talend es una herramienta ETL (*Extract, Transform and Load)*, es el conjunto de procesos que permite a las organizaciones mover datos desde múltiples orígenes y modificar su formato, para luego enviarlos en otra base de datos, para su análisis posterior, o enviar los datos hacia otro sistema operacional para apoyar un proceso de negocio.

Ofrece un diseñador visual (basado en Eclipse RCP) que permite definir todo el flujo de transformaciones en base a componentes predefinidos (más de 400, también se pueden crear y ampliar).

Este diseñador proporciona una vista gráfica de los procesos, con componentes arrastrables que permiten: mappings, transformaciones, etc; funciones especializadas como data filtering, data multiplexing, o ELT; y soporte para RDBMS, file formats, LDAP directories, etc.

Niveles de Talend

El diseño de Talend se basa en 3 niveles:

- Business Models: Nivel diseñado para modelar de manera teórica la aplicación, para lo cual se realizan diagramas de flujo básicos con actores de los procesos.
- Job Designs: El nivel más importante, ya que en este se diseña el trabajo en sí, el código que será ejecutado.
- Contexts: Nivel que contiene los contextos, los cuales pueden ser definidos como variables globales de ejecución del programa, como la carpeta donde se ejecutará la aplicación final o variables iniciales de entrada.

Job Designs

Es el nivel más utilizado en Talend Open Studio. Está formado por el conjunto de *Jobs*, o tareas a realizar. Cada Job inicialmente de una pestaña en blanco, donde se arrastran elementos de una paleta ubicada en la parte derecha del diseñador.

En dicha paleta se encuentran varios elementos configurables, llamados *Subjobs*, los cuales se encargan de ejecutar tareas predeterminadas pero configurables como conexiones, consultas, código personalizado, etc.

Dichos Subjobs se encuentran separados en la paleta en diversas categorías: *BI, Cloud, Custom Code, Data Quality, ETL, Orchestation, System*, etcétera.
Algunos Subjobs que pueden resultar interesantes son:

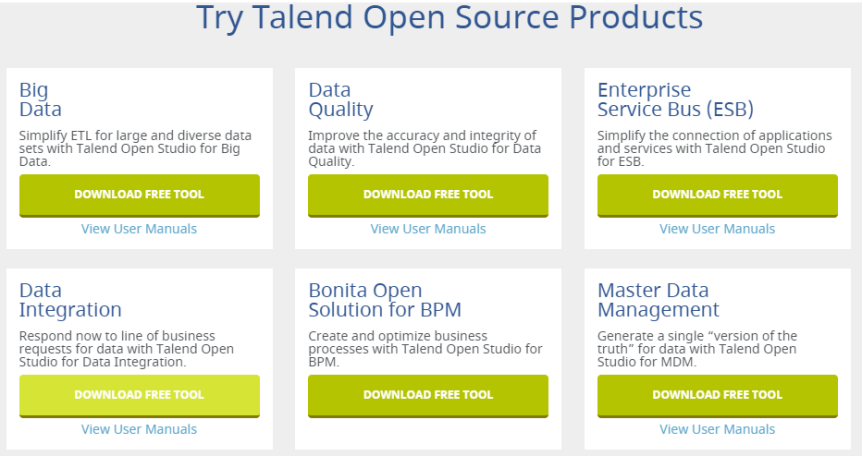
- Conexiones estandarizadas y personalizables a bases de datos, incluye soporte a gran cantidad de las bases de datos Open Source (MySQL, PostgreSQL, SLQLite) e incluso Sybase y Oracle.
- Ejecutores de consultas y procedimientos almacenados en las mencionadas bases de datos.
- Código Java personalizable.
- Iteradores, repetidores de subtareas.
- Inscripción de variables globales, muy usado para mantener una variable global, ya que los subjobs solo crean variables locales.
- Extractores e insertores de datos de archivos (xml, properties), también configurables.
- Conexión a FTP para envío y descarga de archivos.
- Compresión de archivos.
- Modificadores de carpetas (crear, eliminar, modificar).
- Filtros de información.

Los elementos mencionados se unen mediante un flujo secuencial, guiado por flechas extraídas de cada uno de ellos que hacen referencias a eventos de dos clases:

- row: evento que transmite data del elemento al elemento apuntado.
- trigger: conjunto de eventos, activados por posibles modos de ejecución.
    - OnSubjobOk: cuando el elemento ha sido ejecutado sin errores.
    - OnSubjobError: cuando el elemento ha tenido algún error en la ejecución.
    - otros eventos, dependiendo del tipo de Subjob.

Adicionalmente, un *Job* puede contener a uno o varios otros *Jobs* e invocarlos como parte de un nuevo flujo, ejecutando los *Subjobs* contenidos dentro de cada uno.
Finalmente, cada *Job* es exportable a un archivo comprimido en formato ZIP, que contiene el ejecutable: un archivo .jar llamado a partir de un archivo .bat (para Windows) o un archivo .sh (Linux).

Otro punto en el que destaca Talend Open Studio es en las capacidades de depuración, que permite seguir en tiempo real los datos que fluyen a través de los procesos de transformación.



Productos de Talend Open

Existen varias versiones de este software, las mismas se las puede descargar de la página web http://www.talend.com/download/talend-open-studio. Como se trabajará con gran cantidad de datos, se utilizó la Talend Open Studio for Big Data ya que este permite trabajar con bases de datos NoSQL

GEPHI

Para la instalación de Gephi nos vamos a la página web http://gephi.github.io/, y descargamos el instalador, se usó al versión 0.8.2-Beta.

Esta herramienta es open source y está disponible para todos los sistemas operativos de Linux, Windows y Mac.

Distribuciones de Gephi

Se procede con la instalación de esta herramienta, la misma que nos servirá para calcular la distribución de grado de cada grafo, así como para el cálculo de los nodos quimera.