
CAMPUS
DE EXCELENCIA
INTERNACIONAL

Máster Universitario en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TESIS FIN DE MASTER

# Implementación de algoritmos de ensamblaje de genomas en sistemas de memoria compartida y memoria distribuida

Autor: Adolfo Blanco Diez

Directores: Vicente Martín Ayuso

MADRID, JULIO DE 2013

# RESUMEN

El desarrollo de algoritmos ensambladores de genes y la utilización de estos está viviendo un aumento muy espectacular en los últimos años.

Debido a las mejoras ofrecidas en los dispositivos hardware de los numerosos supercomputadores que existen hoy en día se pueden realizar experimentos científicos de una manera más asequible que hace unos años.

Este proyecto servirá como introducción en el complejo mundo de algoritmos científicos, más concretamente en algoritmos ensambladores de genomas. Veremos de primera mano cómo utilizar estas nuevas tecnologías, con ejemplos sencillos, pero con un desarrollo lo bastante importante para darnos una idea del funcionamiento de todas las fases de experimentación que engloban los algoritmos ensambladores y la utilización de la programación paralela en supercomputadores.

Concretamente en este proyecto se van a analizar exhaustivamente una serie de algoritmos ensambladores que serán probados en uno de los supercomputadores más potentes de España, el Magerit 2.

En estas pruebas vamos a proceder al ensamblado de genomas de tres tipos de organismos como bacterias (*Staphylococcus Aureus*, y *Rhodobacter Sphaeroides*) y una prueba gran escala con el genoma del Cromosoma 14 del *Homo Sapiens Sapiens* (Ser humano).

Después procederemos a la comparación de todos los resultados obtenidos para poder comprobar que algoritmos realizan mejor su trabajo y ajustar dicha decisión a las necesidades que tenemos actualmente para buscar un algoritmo eficaz.

# ABSTRACT

The development of gene assemblers and the use of supercomputers for scientific purposes are experiencing an increase in recent years.

The improvements in hardware devices of many supercomputers are allowing to the scientists get important results much easier than a few years ago.

This project presents an introduction to the complex world of scientific algorithms, more specifically gene assembly algorithms. We will see, firsthand, how to use these new technologies, with simple examples but sophisticated enough to give us an idea about all phases of testing these algorithms.

Specifically in this project we will analyze a set of gene assembly algorithms that will be tested in one of the most powerful supercomputers in Spain, Magerit 2.

In these tests we will assemble genome sequence of several organisms like bacteria (*Sthapylococcus Aureus* and *Rhodobacter Sphaeroides*) and a big test with Chromosome 14 of *Homo Sapiens Sapiens*. After that, we will compare the results obtained to get statistics of performance.

# TABLA DE CONTENIDOS


1. INTRODUCCIÓN Y OBJETIVOS ..... 1
2. ESTADO DEL ARTE ..... 5
   2.1 ¿Qué es un Genoma? ..... 5
   2.2 Procesos de Ensamblado de Secuencias ..... 6
      2.2.1 Proceso de Secuenciación ..... 6
      2.2.2 Proceso de Ensamblado ..... 7
      2.2.3 Calidad del Ensamblaje ..... 8
   2.3 Algoritmos de Ensamblaje de Genomas ..... 9
      2.3.1 ¿Qué es un grafo? ..... 9
      2.3.2 Grafos de Bruijn ..... 11
      2.3.3 Grafos en Cadena ..... 13
      2.3.4 Grafos de Bruijn vs. Grafos en Cadena ..... 14
      2.3.5 Algoritmos Voraces ..... 16
   2.4 Clasificación de Ensambladores ..... 17
      2.4.1 Clasificación por Categorización ..... 17
      2.4.2 Clasificación por Características de Ensamblado ..... 18
      2.4.3 Clasificación por Tipo de Algoritmo ..... 18
      2.4.4 Clasificación por Tipo de Memoria ..... 19
   2.5 Tecnologías Utilizadas ..... 21
   2.6 Formatos de Representación de Secuencias de Genes ..... 23
      2.6.1 FASTA ..... 23
      2.6.2 FASTQ ..... 23
      2.6.3 GenBank ..... 24
      2.6.4 EMBL ..... 26
   2.7 Hardware y Software Utilizado en las Pruebas ..... 28
      2.7.1 Magerit 2 ..... 28
      2.7.2 SLURM (Simple Linux Utility for Resource Management) ..... 30
      2.7.3 Ejecutar Tareas en Magerit 2. ..... 31
      2.7.4 Ensamblador Minia ..... 33
      2.7.5 Ensamblador Velvet ..... 36
      2.7.6 Ensamblador Ray ..... 39
      2.7.7 QUAST-(Quality Assessment Tool for Genome Assemblies) ..... 40
3. EVALUACIÓN DE RIESGOS ..... 43
4. DESARROLLO ..... 45

4.1 Introducción ... 45
4.1.1 Datasets de Genomas Utilizados ... 45
4.2 Ensamblador Minia ... 47
4.2.1 Descarga e Instalación ... 47
4.2.2 Preparación de las Lecturas ... 48
4.2.3 Ejecución ... 49
4.2.4 Extraer Resultados ... 49
4.3 Ensamblador Velvet ... 51
4.3.1 Descarga e Instalación ... 51
4.3.2 Ejecución ... 52
4.3.3 Extraer Resultados ... 53
4.4 Ensamblador Ray ... 54
4.4.1 Descarga e Instalación ... 54
4.4.2 Ejecución ... 54
4.4.3 Extraer Resultados ... 55
5. RESULTADOS ... 57
5.1 Resultados Ensamblador Minia ... 57
5.1.1 Sthapylococcus Aureus ... 57
*5.1.2 Rhodobacter Sphaeroides* ... 59
5.1.3 *Homo Sapiens Sapiens* Cromosoma 14 ... 61
5.2 Resultados Velvet ... 63
5.2.1 Sthapylococcus Aureus ... 63
*5.2.2 Rhodobacter Sphaeroides* ... 65
5.2.3 Home Sapiens Sapiens Cromosoma 14 ... 67
5.3 Resultados Ray ... 68
5.3.1 Sthapylococcus Aureus ... 68
*5.3.2 Rhodobacter Sphaeroides* ... 70
5.3.3 Home Sapiens Sapiens Cromosoma 14 ... 72
5.4 Resumen de Resultados ... 75
5.4.1 Sthapylococcus Aureus ... 75
*5.4.2 Rhodobacter Sphaeroides* ... 78
5.4.3 *Homo Sapiens Sapiens* Cromosoma 14 ... 81
6. CONCLUSIONES ... 85
7. LÍNEAS FUTURAS ... 87
8. BIBLIOGRAFÍA ... 89
ANEXOS ... 91
Conexión remota a Magerit 2. ... 91

Conexión remota desde Windows ..... 91
Conexión remota desde Linux ..... 95
Descarga de Datos en Magerit ..... 98

# Listado de Figuras

Ilustración 1 Esquema del DNA ..... 5
Ilustración 2 Pipeline proceso de ensamblado ..... 6
Ilustración 3 Proceso de Ensamblado ..... 7
Ilustración 4 Ejemplo Grafo de Bruijn Lectura Simple ..... 11
Ilustración 5 Ejemplo Grafo de Bruijn (lectura Múltiple) ..... 11
Ilustración 6 Ejemplo Grafo de Bruijn (lectura Múltiple con Errores) ..... 12
Ilustración 7 Ejemplo Grafo de Bruijn Múltiples (lecturas con Repeticiones) ..... 12
Ilustración 8 Ejemplo de Grafo en Cadena (lectura simple) ..... 13
Ilustración 9 Grafo de Superposición ..... 13
Ilustración 10 Grafo en Cadena ..... 13
Ilustración 11 Grafo de Bruijn vs Grafo en Cadena (Grafo de Bruijn) ..... 14
Ilustración 12 Grafo de Bruijn vs Grafo en Cadena (Grafo en Cadena) ..... 14
Ilustración 13 Grafo de Bruijn vs Grafo en Cadena (Grado de Bruijn con error en la lectura) ..... 15
Ilustración 14 Grafo de Bruijn vs Grafo en Cadena (Grado en Cadena con error en la lectura) ..... 15
Ilustración 15 Esquema Memoria Compartida ..... 19
Ilustración 16 Esquema Memoria Distribuida ..... 19
Ilustración 17 Cloud Computing ..... 20
Ilustración 18 Ejemplo Formato Fasta ..... 23
Ilustración 19 Ejemplo Formato FASTQ ..... 24
Ilustración 20 Primera Parte Formato GenBank (Introducción) ..... 24
Ilustración 21 Segunda Parte Formato GenBank (Comentarios) ..... 25
Ilustración 22 Tercera Parte Formato GenBank (Secuencia) ..... 25
Ilustración 23 Primera Parte Formato EMBL (Introducción) ..... 26
Ilustración 24 Segunda Parte Formato EMBL (Comentarios) ..... 26
Ilustración 25 Tercera Parte Formato EMBL (Secuencia) ..... 27
Ilustración 26 Supercomputador Magerit 2 ..... 28
Ilustración 27 Estructura Interna Magerit 2 ..... 29
Ilustración 28 Esquema arquitectura Magerit 2 ..... 29
Ilustración 29 Estructura Script de Tareas Magerit ..... 32
Ilustración 30 Ejemplo de la Estructura de Datos del Ensamblador Minia ..... 34
Ilustración 31 Ejemplo del Grafo de Bruijn Velvet para k = 5 ..... 37
Ilustración 32 Ejemplo de corrección de errores mediante el algoritmo Tour Bus ..... 38
Ilustración 33 Compilación Ensamblador Minia ..... 47
Ilustración 34 Test Funcionamiento Minia ..... 48
Ilustración 35 Ejemplo de Script de Ejecución de Tareas Minia ..... 49
Ilustración 36 Ejemplo Script de Ejecución Ensamblador Velvet ..... 52
Ilustración 37 Configuración Putty ..... 92
Ilustración 38 Pantalla de bienvenida Magerit en Windows. ..... 92
Ilustración 39 Configuración WinSCP ..... 93
Ilustración 40 Proceso de Login WinSCP ..... 94
Ilustración 41 Pantalla principal WinSCP ..... 94
Ilustración 42 Terminal Linux ..... 95
Ilustración 43 Comando SSH Linux ..... 96
Ilustración 44 Pantalla de Bienvenida Magerit en Linux ..... 96
Ilustración 45 Aplicación Connect Server Ubuntu ..... 97
Ilustración 46 Carpeta Home de Magerit en Linux ..... 98

# Listado de Tablas

Tabla 1 Comparación ensambladores De-Novo vs. Mapping ..... 17
Tabla 2 Tecnologías Utilizadas en Ensambladores (I) ..... 21
Tabla 3 Tecnologías Utilizadas en Ensambladores (II) ..... 22
Tabla 4 Niveles de Calidad de Servicio Magerit 2 ..... 31
Tabla 5 Características Librerías *Staphylococcus Aureus* ..... 46
Tabla 6 Características Librerías *Rhodobacter Sphaeroides* ..... 46
Tabla 7Características Librerías *Homo Sapiens Sapiens* Cromosoma 14 ..... 46
Tabla 8 Recopilación de Datos Ensamblador Minia ..... 50
Tabla 9 Script de Ejecución Ensamblador Ray ..... 54
Tabla 10 Gráfico del Tiempo de Ejecución Ensamblador Minia para el genoma Sthapylococcus Aureus .... 57
Tabla 11 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Staphylococcus Aureus* ..... 58
Tabla 12 Estadísticas de Ensamblaje para Ensamblador Minia Genoma *Staphylococcus Aureus* ..... 58
Tabla 13 Gráfico del tiempo de Ejecución Ensamblador Minia Genoma *Rhodobacter Sphaeroides* ..... 59
Tabla 14 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Rhodobacter Sphaeroides* ..... 59
Tabla 15 Estadísticas Ensamblador Minia Genoma *Rhodobacter Sphaeroides* ..... 60
Tabla 16 Tiempo de Ejecución Ensamblador Minia Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 61
Tabla 17 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 61
Tabla 18 Estadísticas Ejecución Ensamblador Minia para el genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 62
Tabla 19 Tiempo de Ejecución Ensamblador Velvet Genoma *Staphylococcus Aureus* ..... 63
Tabla 20 Gráfico de consumo teórico de Memoria RAM Ejecución Ensamblador Velvet para el genoma *Staphylococcus Aureus* ..... 64
Tabla 21 Estadísticas Ensamblador Velvet 166 CPU Genoma *Staphylococcus Aureus* ..... 64
Tabla 22 Tiempo de Ejecución Ensamblador Velvet Genoma *Rhodobacter Sphaeroides* ..... 65
Tabla 23 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Velvet para el genoma *Rhodobacter Sphaeroides* ..... 66
Tabla 24 Estadísticas Ensamblador Velvet 16CPU Cromosoma *Rhodobacter Sphaeroides* ..... 66
Tabla 25 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Staphylococcus Aureus* ..... 68
Tabla 26 Consumo de Memoria RAM Ensamblador Ray Genoma *Staphylococcus Aureus* ..... 68
Tabla 27 Estadísticas sobre Contigs Ensamblador Ray Genoma *Staphylococcus Aureus* ..... 69
Tabla 28 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Staphylococcus Aureus* ..... 69
Tabla 29 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Rhodobacter Sphaeroides* ..... 70
Tabla 30 Consumo de Memoria RAM Ensamblador Ray Genoma *Rhodobacter Sphaeroides* ..... 70
Tabla 31 Estadísticas sobre Contigs Ensamblador Ray Genoma *Rhodobacter Sphaeroides* ..... 71
Tabla 32 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Rhodobacter Sphaeroides* ..... 71
Tabla 33 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 72
Tabla 34 Consumo de Memoria RAM Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14 . 73
Tabla 35 Estadísticas sobre Contigs Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 73
Tabla 36 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14... 74
Tabla 37 Resumen del Tiempo de Ejecución Genoma *Staphylococcus Aureus* ..... 75
Tabla 38 Resumen Consumo de Memoria RAM *Staphylococcus Aureus* ..... 76
Tabla 39 N50 Genoma *Staphylococcus Aureus* ..... 76

Tabla 40 Número de Contigs y Scaffolds Genoma *Staphylococcus Aureus* ..... 77
Tabla 41 Tiempo de Ejecución Genoma *Rhodobacter Sphaeroides* ..... 78
Tabla 42 Consumo de Memoria RAM Genoma *Rhodobacter Sphaeroides* ..... 78
Tabla 43 N50 Genoma *Rhodobacter Sphaeroides* ..... 79
Tabla 44 Número de Contigs y Scaffolds Genoma *Rhodobacter Sphaeroides* ..... 79
Tabla 45 Tiempo de Ejecución Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 81
Tabla 46 Consumo de Memoria RAM Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 81
Tabla 47 Resumen Valores N50 Genoma *Homo Sapiens Sapiens* Cromosoma 14 ..... 82
Tabla 48 Número de Contigs y Scaffolds Genoma *Rhodobacter Sphaeroides* ..... 82

# 1. INTRODUCCIÓN Y OBJETIVOS

Hoy en día, en la era de la información, es muy común utilizar las tecnologías computacionales para intentar mejorar y facilitar la resolución de múltiples problemas de todo tipo.

Como bien sabemos podemos presentar soluciones más o menos aceptables a diversos problemas por medio de medios computacionales.

Esta utilización "masiva" de soluciones computacionales, ha permitido la unión y convivencia de campos científicos que a priori podría parecer que no tienen mucho que ver entre sí.

En este proyecto en concreto nos vamos a centrar en la unión de esos dos campos: El campo de la Biología-Biotecnología y el campo de la Ingeniería informática. La unión de estos dos campos ha dado lugar a un nuevo campo de investigación denominado Bioinformática.

La Bioinformática es un campo de la ciencia que intenta utilizar los campos citados anteriormente con el objetivo de almacenar, analizar, mantener e interpretar la sucesión de diversos datos biológicos para su posterior utilización en distintas áreas del campo de la Biología y de la Biotecnología.

Debido a que la utilización de la computación en el campo de la biología se puede extender a varias áreas, en este proyecto nos vamos a centrar en el área biológica de la genómica.

En el área de la genómica, siempre ha sido muy importante la clasificación y la anotación de genes que están presentes en una secuencia de DNA de cualquier ser vivo.

El proceso de anotación de genes, es un proceso de gran complejidad que antes de ser automatizado era un proceso tedioso, largo y laborioso.

Con la llegada de las nuevas técnicas de anotación y secuenciación, es decir, con la utilización de métodos computacionales, se ha llegado a conseguir que este proceso se simplifique y permita la realización del proceso en un periodo de tiempo mucho menor al anterior. Aunque el proceso se haya simplificado al utilizar nuevas técnicas, se han creados nuevos problemas relacionados con la complejidad de estas tareas y la utilización de métodos computacionales. Para realizar el proceso de ensamblado de genomas lo más común es utilizar tecnologías de computación paralela debido al gran número de datos que se generan y que hay que procesar, dando lugar a bases de datos con cantidades ingentes de información que puede llegar a tener una tamaño de varios Terabytes de información que hay que procesar de forma más o menos simultánea para poder llegar a conseguir un ensamblado de secuencias de DNA aceptable.

La utilización de métodos computacionales ha permitido un enorme avance en la ciencia y un ejemplo claro de este avance es la finalización del proceso de secuenciación del genoma humano en el año 2006, proyecto que comenzó en el año 1990.

Pese a que la computación ha ganado un puesto importante en el proceso de secuenciación y de anotación de genes, es importante destacar que es necesaria la presencia de expertos en los campos de Biología-Biotecnología que, en diversas partes del proceso de secuenciación, puedan determinar,

analizar y comprender la veracidad de los datos para poder definir si el proceso se está realizando de forma correcta.

En este aspecto se puede destacar que en el campo de bioinformática expertos de Biología-Biotecnología y expertos en Ingeniería Informática tienen que conseguir trabajar en equipo con una comunicación adecuada, aunque a veces puede que no sea fácil conseguirla ya que se trata de llegar a un entendimiento entre dos campos muy distintos del mundo científico.

Aunque, en el presente proyecto, se van a explicar conceptos que pertenecen de forma muy clara al campo de la Biología-Biotecnología, el proyecto será centrado en el aspecto computacional de las pruebas que se van a realizar sobre ensambladores, secuenciación y ensamblado de genomas.

El proyecto consiste principalmente en la comprobación de manera práctica del funcionamiento de ensambladores de genomas en un entorno de pruebas real como es el supercomputador Magerit 2. Para ello vamos a realizar varias pruebas y vamos a estudiar todos los resultados obtenidos para observar si el proceso de ensamblado de genomas por una serie de ensambladores se realiza de forma correcta y el gasto computacional que produce la utilización de este tipo de programas.

Antes de concluir esta introducción, vamos a realizar una pequeña vista hacia atrás para ver la evolución de los ensambladores de genomas a lo largo del tiempo de la forma más resumida posible.

Los primeros ensambladores se desarrollaron en la década de 1980. Estos ensambladores eran bastante limitados y diferentes a los que nos podemos encontrar hoy en día. Utilizaban secuencias largas que permitían una mejor identificación de los fragmentos que se solapaban. Estos primeros ensambladores tenían un uso bastante limitado ya que utilizaban algoritmos poco depurados que podían dar lugar a una complejidad casi exponencial que aumentaba el coste de manera alarmante y el consumo de recursos.

A medida que el tiempo pasaba, los ensambladores iban mejorando poco a poco. Los objetivos de conseguir una reducción de costes se iban cumpliendo poco a poco. Esta mejora en la reducción de costes permitía la producción de lecturas más cortas, estas nuevas lecturas dan lugar a una nueva generación de ensambladores denominada “Next Generation Sequencing”.

El desarrollo de lecturas más cortas es lo que ha dado a la denominación de nueva generación de ensambladores, esto es debido a que cuanto menor es la lectura más rápido se pueden alinear las secuencias. Aunque, y como es lógico, esta evolución no era la perfección. Esta evolución trajo problemas consigo, se dio lugar a nuevos problemas como repeticiones y lazos dentro del proceso de secuenciación que podrían dar lugar a errores en el resultado final del proceso de ensamblaje.

Dadas las características de las dos generaciones de ensambladores vistas, podemos decir que hoy en día, los científicos están tratando de encontrar un camino intermedio entre ensambladores de genomas de primera generación y los de la segunda generación.

Los objetivos planteados para la realización de este proyecto son los siguientes:

- Aprendizaje y comprensión sobre el funcionamiento de los distintos algoritmos de ensamblaje de genomas.
- Entender los resultados que muestran dichos algoritmos y comprender su utilización en el mundo real.
- Mejora de la comprensión de la utilización de tecnologías de procesamiento de grandes cantidades de datos mediante tecnologías de programación en paralelo.
- Comprender el funcionamiento de un supercomputador y como estos elementos mejoran la comprensión de grandes cantidades de datos.
- Implementación adecuada de distintos algoritmos de ensamblado de genomas y analizar los resultados expresados por los mismos y compararlos con otros algoritmos que tienen el mismo fin.
- Comprobación "in-situ" de la integración de campos muy contrapuestos como son la Biología-Biotecnología y la Ingeniería Informática.
- Experimentación en la unión del campo de investigación con al campo comercial-empresa en la Ingeniería Informática.

Los objetivos citados anteriormente son los objetivos principales para la realización de este Proyecto de Fin de Máster. Debido a que en el Máster en Ingeniería Informática se han impartido nociones de procesamiento de grandes cantidades de datos con tecnologías paralelas este proyecto es muy adecuado para complementar, ampliar y mejorar los conocimientos adquiridos durante el máster en un entorno real de trabajo.

El proyecto está dividido en varios capítulos:

- **Capítulo 2** (ESTADO DEL ARTE): En este capítulo se van a especificar, identificar y definir todos los aspectos de la situación actual del proceso, tecnologías, formatos etc… Que se utilizan en la secuenciación, anotación y ensamblaje de genes.
- **Capítulo 3** (EVALUACIÓN DE RIESGOS): En este capítulo se van a especificar los riesgos que pueden aparecer en el proyecto y la manera de evaluarlos y limitarlos.
- **Capítulo 4** (DESARROLLO): En este capítulo se van a definir todas las pruebas realizadas con ensambladores en el supercomputador Magerit 2.
- **Capítulo 5** (RESULTADOS): En este capítulo se van a estudiar y comentar todos los resultados obtenidos durante el desarrollo del proyecto
- **Capítulo 6** (CONCLUSIONES): En este capítulo vamos a comentar las conclusiones más relevantes que podemos extraer gracias a los resultados obtenidos en el desarrollo del proyecto.
- **Capítulo 7** (LINEAS FUTURAS): En este capítulo vamos a definir las posibles líneas futuras de este tipo de tecnologías.
- **Capítulo 8** (BIBLIOGRAFÍA): En este capítulo especificamos toda la documentación utilizada para la realización del proyecto.

# 2. ESTADO DEL ARTE

Para el desarrollo del presente proyecto tenemos que tener claros una serie de conceptos básicos. Entre estos conceptos básicos, el más importante de todos podrías considerarse el término Genoma. Pero, y como dice el título de esta sección, ¿Qué es un genoma?

## 2.1 ¿Qué es un Genoma?

El genoma es el conjunto de información hereditaria que posee un organismo o una especie particular de organismos.

Esta información hereditaria está codificada, o bien en el DNA para organismos unicelulares y organismos multicelulares, o bien en el RNA para algunos tipos de virus y está dividido en un conjunto de elementos discretos denominados genes.

La molécula de DNA está formada por 4 diferentes nucleótidos que están formados a su vez por un azúcar (desoxirribosa), una base nitrogenada (que puede ser Citosina (C), Adenina (A), Timina (T) y Guanina (G)) y un grupo de fosfato que actúa como unión entre un elemento y el siguiente en la molécula de ADN.

Lo que distingue un nucleótido de otro es la base nitrogenada, por lo que la secuencia de ADN se nombra solo utilizando la secuencia de sus bases. La disposición secuencial de estas cuatro bases a lo largo de la cadena es la que codifica la información genética.

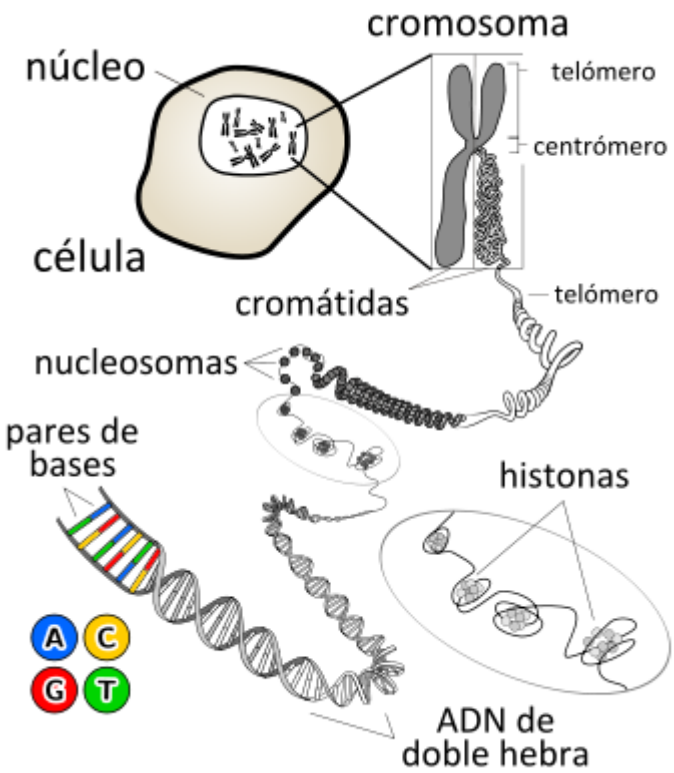



Ilustración 1 Esquema del DNA

En los organismos vivos la secuencia de DNA se representa como una doble cadena de nucleótidos, también denominadas hebras y que se representa en tres dimensiones. Se encuentran en el núcleo de cada célula del organismo.

Dentro de las células el DNA está organizado en estructuras denominadas cromosomas que, durante el ciclo celular, se duplican antes de que la célula se divida. Dependiendo del tipo del organismo podemos encontrar unas pequeñas variaciones. Por ejemplo en los organismo procariotas almacenan el DNA en el citoplasma de la célula, los organismos eucariotas almacenan la mayor parte de su DNA en el núcleo celular y una parte en las mitocondrias, en los plastos y los centros organizadores microtúbulos o centriolos, en caso de tenerlos.

Dependiendo de su tamaño, podemos dividir el DNA en tres clases distintas:

- **Genomas pequeños:** Genomas de bacterias que contienen algunas “Megabases”
- **Genomas medianos:** Genomas de pequeñas plantas que contienen cientos de “Megabases”.
- **Genomas grandes:** Plantas y mamíferos que contienen “Gigabases”.

## 2.2 Procesos de Ensamblado de Secuencias

El ensamblado de genomas es un proceso con el que se busca encontrar una representación exacta del genoma de un organismo vivo.

Debido a que esta tarea es una tarea muy complicada nació un sub-campo de la biología que se encargaba de dicha tarea. Este campo está centrado en la utilización de diferentes algoritmos para la construcción de las secuencias de genes y su ensamblado posterior.

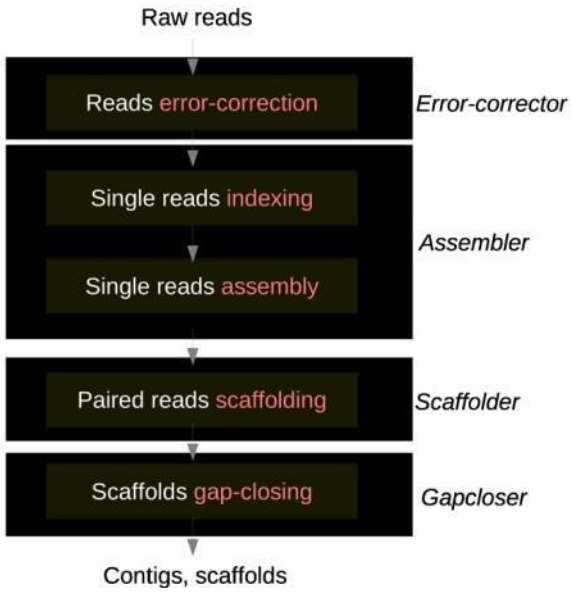


Ilustración 2 Pipeline proceso de ensamblado

En el proceso de ensamblado se divide en dos procesos bien diferenciados:

El primero de ellos es el proceso de secuenciación que es un proceso previo muy importante para realizar el ensamblado de forma correcta.

El segundo proceso es el encargado de realizar el ensamblado propiamente dicho, por medio de Ensambladores que son los programas encargados de realizar lecturas cortas de una secuencia. El proceso comienza con lecturas cortas debido a que la tecnología actual no permite leer una secuencia entera de una sola vez.

### 2.2.1 Proceso de Secuenciación

El secuenciador es el encargado de realizar la producción de los datos que posteriormente serán unidos por el ensamblador. Para producir estos datos se realiza el proceso de secuenciación mediante un secuenciador.

Este proceso consiste en la lectura y descodificación de los nucleótidos de DNA que compone el genoma de un organismo vivo.

Actualmente el proceso de secuenciación es un proceso automatizado y digital gracias a las numerosas mejoras que han ido apareciendo en el mundo de la computación de alto rendimiento. Pero anteriormente este proceso era un proceso muy complicado que podría conllevar una gran cantidad de recursos y tiempo para conseguir una secuenciación más o menos aceptable de genomas no demasiado grandes.

Cómo es lógico, para la temática que tratamos en este proyecto, no nos vamos a centrar en la historia completa de todos los procesos de secuenciación que podemos denominar "pre-computacionales" ya que no tienen demasiada relevancia. Solo destacar que era un proceso de gran complejidad que se ha ido simplificando con la utilización de la computación.

Actualmente existen muchos métodos distintos para la secuenciación de DNA por medio de la computación de alto rendimiento. A continuación vamos a destacar los procedimientos, que quizás, que son los más importantes, y que son utilizados por tecnologías que también participan en el proceso de ensamblaje que comentaremos posteriormente.

Aunque hay varios métodos para realizar el proceso de secuenciación podemos destacar que en la actualidad todos ellos se basan en la idea de paralelizar las tareas de secuenciación que permiten la producción de miles, o incluso millones, de secuencias de DNA de forma simultánea. Dentro de los métodos más actuales podemos destacar:

- **Método de Amplificación clonal in vitro:** Este método se basa en la idea de que la mayoría de los métodos de detección molecular no son los suficientemente sensibles para la secuenciación de una sola molécula. Para evitar este problema se utiliza la clonación in vitro de dichas moléculas para generar muchas copias para facilitar el proceso de secuenciación. Este método es utilizado en tecnologías como: 454, SOLID o Solexa (Ver Capítulo 2.5).

- **Secuenciación paralelizada:** Una vez que las secuencias clonadas de ADN se localizan físicamente en posiciones separadas de la superficie, se pueden utilizar varios métodos de secuenciación para determinar las secuencias de ADN de todas las localizaciones en paralelo. La "secuenciación por síntesis", como en la popular secuenciación electroforética con terminador marcado con colorante, usa el proceso de síntesis de ADN por ADN polimerasa para identificar las bases presentes en la molécula complementaria de ADN. Este método es utilizado en tecnologías como: 454 o SOLID (Ver Capítulo 2.5).

### 2.2.2 Proceso de Ensamblado

Una vez que el secuenciador ha realizado el proceso de secuenciación se manda la información recopilada al ensamblador para que proceda a realizar el ensamblado completo.

El proceso de ensamblado es una tarea jerarquizada en la que podremos distinguir los siguientes elementos:

- **Lectura proporcionada por el secuenciador:** Es el elemento más sencillo de la jerarquía que podremos encontrar en el proceso de ensamblado.
- **Contig:** Podemos definir un contig como el siguiente nivel en la jerarquía del proceso de ensamblado. Un contig no es más que el alineamiento de múltiples lecturas. La característica principal de los contigs es que es un

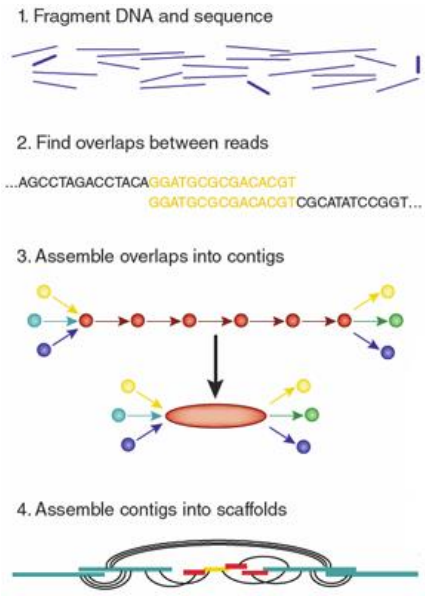



Ilustración 3 Proceso de Ensamblado

alineamiento sin espacios y sin un orden definido.

- **Scaffold:** Nivel más alto en la jerarquía del proceso. Se trata de una evolución de los Contigs ya que, un Scaffold es la suma de dos o más Contigs de forma ordenada y orientada según la estructura del genoma original. En este proceso sí que se pueden producir espacios o errores producidos por un error en el proceso de ensamblado

Explicada la jerarquía principal del proceso de ensamblado, podemos definir este proceso de la siguiente manera:

Dado un conjunto de fragmentos de secuencia, el objetivo es encontrar la supersecuencia común (o secuencia origen de los fragmentos) más corta:

1. Calcular alineamientos por pares de todos los fragmentos.
2. Elegir los dos fragmentos con el mayor solapamiento.
3. Mezclar los fragmentos elegidos.
4. Repetir los pasos 2. y 3. hasta que sólo quede un fragmento.

El resultado es una solución subóptima al problema.

Como podemos ver es un proceso que va "ensamblando" los elementos menores de la jerarquía, las lecturas, en contigs, y posteriormente estos contigs en Scaffolds o supersecuencias. Finalmente conseguiremos un único fragmento que se tratará del genoma totalmente alineado correctamente.

El resultado de los ensambladores se puede medir por el tamaño y la precisión de sus contigs y supercontigs.

El proceso de secuenciación, aparte de producir los datos necesarios para el ensamblado de genomas, también nos proporciona datos acerca de la orientación del extremo par final.

- **Izquierda Derecha (I/D)** Esta orientación indica que las lecturas han sido extraídas de soportes DNA opuestos por lo que la lectura es a la inversa.
- **Izquierda Izquierda / Derecha Derecha (II/DD)** Esta orientación indica que las lecturas han sido extraídas de la misma cadena DNA y por lo tanto tienen la misma orientación.
- **Derecha Izquierda (D/I)** Esta orientación indica que las lecturas han sido extraídas de hebras diferentes pero en el sentido contrario de la orientación de lectura Izquierda-Derecha.

### 2.2.3 Calidad del Ensamblaje

La calidad del ensamblaje viene dada normalmente por datos estadísticos donde se incluye la longitud máxima, la longitud media, la longitud combinada total y el N50. (Ver Capítulo 2.4.2)

El proceso que hemos ido describiendo con anterioridad tiene una serie de problemas que complican bastante su implementación y puede conllevar a errores y, por lo tanto, alterar su calidad.

Los problemas vienen definidos por el número de secuencias repetidas que puede contener un genoma en su interior. Aproximadamente un 30% de las secuencias de un genoma son repetidas. Estas secuencias hacen más difícil el proceso de ensamblado y aumenta la probabilidad de errores. Los errores que se pueden provocar por esta característica de los genomas son los siguientes: Eliminación de parte de las secuencias, secuencias colocadas en un orden que no les corresponden…

Después de realizar el proceso de ensamblado, podemos considerar un genoma terminado cuando tiene un único error por cada 10.000 bases procesadas.

Para evitar la proliferación de errores y mejorar el proceso de ensamblado de genes podemos destacar que en el proceso de secuenciación, un secuenciador puede producir múltiples lecturas del DNA al realizar varias rondas de secuenciación y de segmentación.

## 2.3 Algoritmos de Ensamblaje de Genomas

Los ensambladores de genomas utilizan una serie de algoritmos matemáticos de gran complejidad para generar los contigs necesarios para ensamblar la secuencia completa de DNA.

Cómo hemos dicho anteriormente, en las secuencias de DNA es común encontrarse con fragmentos repetidos, que pueden complicar el proceso, debido a este detalle importante tenemos que plantearnos la utilización de grafos para generar los contigs, debido a que los grafos por su estructura permiten la superposición de dichos fragmentos.

En la actualidad dos tipos de grafos son utilizados en el proceso de secuenciación de DNA. Estos grafos son: Grafos de Bruijn (utilizados por tecnología Illumina) y Grafos en Cadena (utilizados por tecnología 454) (Ver Capítulo 2.5).

A continuación vamos a describir el funcionamiento de los distintos tipos de grafos. Pero antes tenemos que responder a la siguiente cuestión. ¿Qué es un grafo?

### 2.3.1 ¿Qué es un grafo?

Un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos que permiten relaciones entre los elementos de un conjunto.

Típicamente un grafo se representa gráficamente con un conjunto de puntos (nodos) unidos por líneas (aristas).

Un grafo $G$ es un par ordenado $G = (V, E)$, donde:

- $V$ es un conjunto de vértices o nodos, y
- $E$ es un conjunto de aristas o arcos, que relacionan estos nodos.

Normalmente $V$ suele ser finito. Muchos resultados importantes sobre grafos no son aplicables para *grafos infinitos*.

Se llama orden del grafo $G$ a su número de vértices$|V|$.

El grado de un vértice o nodo $V$es igual al número de arcos $E$que se encuentran en él.

Un bucle es una arista que relaciona al mismo nodo; es decir, una arista donde el nodo inicial y el nodo final coinciden.

Dentro de los grafos podemos encontrar grafos dirigidos y grafos no dirigidos. Las expresiones matemáticas generales que definen ambos tipos de grafos son las siguientes:

- **Grafos No Dirigidos:**

Un grafo no dirigido o grafo propiamente dicho es un grafo $G = (V, E)$ donde:

$V \neq 0$

$E \subseteq \{x \in P(V): |x| = 2\}$ es un conjunto de *pares no ordenados* de elementos de $V$.

Un par no ordenado es un conjunto de la forma $\{a, b\}$, de manera que $\{a, b\} = \{b, a\}$. Para los grafos, estos conjuntos pertenecen al conjunto potencia de $V$ de cardinalidad 2, el cual se denota por $P(V)$ .

- **Grafos Dirigidos**

Un grafo dirigido o es un grafo $G = (V, E)$ donde:

$V \neq 0$

$E \subseteq \{(a, b) \in P(V): a \neq b\}$ es un conjunto de pares ordenados de elementos de $V$.

Dada una arista $\{a, b\}$, $a$ es su *nodo inicial* y $b$ su *nodo final*.

Por definición, los grafos dirigidos no contienen *bucles*.

Un grafo mixto es aquel que se define con la capacidad de poder contener aristas dirigidas y no dirigidas. Tanto los grafos dirigidos como los no dirigidos son casos particulares de este.

### 2.3.2 Grafos 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.

Un grafo de Bruijn para un entero fijo de valor k:

1- **Nodos:** Todos los k-mers (sub secuencias de longitud k) presentes en las lecturas.
2- 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.

**Ejemplo 1:**

- Una lectura simple:
  - ACTG
- Valor de k =3

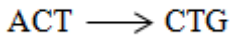



Ilustración 4 Ejemplo Grafo de Bruijn Lectura Simple

**Ejemplo 2:**

- Mantenemos valor de k = 3
- Múltiples lecturas:
  - ACTG
    - CTGC
      - TGCT

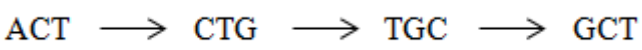



Ilustración 5 Ejemplo Grafo de Bruijn (lectura Múltiple)

En el caso de que se añadieran redundancias a la lectura múltiple anterior el grafo no se vería afectado.

**Ejemplo 3 (Errores):**

- Valor de k = 3
- Múltiples lecturas:
  - ACTG
    - CTGC
      - CTGA
        - TGCT


Ilustración 6 Ejemplo Grafo de Bruijn (lectura Múltiple con Errores)

**Ejemplo 4 (Repeticiones):**

- Valor de k =3
- Múltiples lecturas con repeticiones:
    - ACTG
        - CTGC
            - TGCT
                - GCTG
                    - CTGA
                        - TGAC

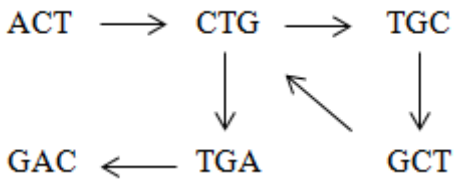



Ilustración 7 Ejemplo Grafo de Bruijn Múltiples (lecturas con Repeticiones)

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

- **Primera Fase:** La primera fase del desarrollo tarta 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. El primer paso trata de encontrar la definición de todas las longitudes de los k-mers únicos.
  Una vez clasificados dichas longitud se generan los nodos del grafo a partir de los k-mers anteriores y se genera las aristas que relacionan dichos nodos.

- **Segunda Fase:** Una vez creado el Grafo de Bruijn al 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 ha producido errores de este tipo puede dar lugar a un grafo erróneo y posiblemente con mayor longitud de lo deseado.

- **Tercera Fase:** En esta fase, y como veremos en el resto de algoritmos, tenemos que ir creando la jerarquía que hemos visto en la Sección 2.2.2. En esta fase se intenta agrupar todos los nodos (lecturas) para conseguir los Scaffolds. Dicho proceso debe dar lugar a una supersecuencia única que será considerada como la solución óptima del genoma buscado.

En este punto podríamos considerar a 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.

### 2.3.3 Grafos en Cadena

Podemos definir un grafo en cadena como un grado de superposición. Podemos hacer esta afirmación debido a que si tenemos un valor de k>0, podemos decir que r y r' es una superposición de lecturas si el sufijo r de longitud l>k, es exactamente un prefijo de r' de longitud similar.

Definición de un grafo de superposición:

1- **Nodos:** Lecturas
2- Dos nodos están conectados por una arista si se superponen dos lecturas

**Ejemplo 1:**

- Valor de k = 3
- Una lectura simple:
  - ACTG


Ilustración 8 Ejemplo de Grafo en Cadena (lectura simple)

Un grafo en cadena es obtenido a partir de un grafo de superposición mediante la eliminación de la redundancia:

- Lecturas redundantes (las que están enteramente contenidas en otras lecturas)
- Transitivamente redundantes:
  (si a → c y a → b → c, entonces eliminamos a → c)

**Ejemplo 2: (De grafo de Superposición a Grafo en cadena)**

- Grafo de superposición para k = 3

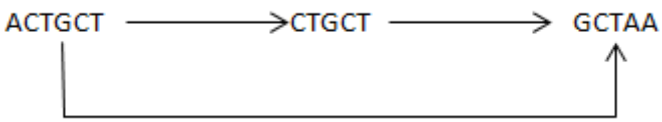
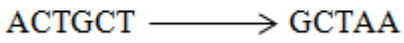



Ilustración 9 Grafo de Superposición

- Grafo en cadena asociado




[15]
Ilustración 10 Grafo en Cadena

La lectura CTGCT está contenida en ACTGCT, por lo que es redundante.

Los ensambladores que utilizan estos tipos de algoritmos, generalmente, tienen tres fases de funcionamiento. Dichas fases son las siguientes:

- **Primera Fase:** Es la primera fase del algoritmo, se trata de la etapa de superposición que consiste en comparar cada lectura con el resto de lecturas para ver si existe superposición. El objetivo final es encontrar los pares de superposición para construir el Grafo en Cadena. Cada lectura la almacenamos en la estructura de datos como un nodo, y si hay superposición entre dos lecturas asociamos las mismas con un vértice.

- **Segunda Fase:** Es la segunda fase del algoritmo, y podemos considerarla como la fase de diseño. El objetivo principal de esta fase es encontrar un único camino que atraviese todo el grafo en cadena y que pase por todos los nodos una sola vez. Durante este proceso podemos comenzar a realizar la configuración de la jerarquía utilizada por todos los ensambladores (Cómo hemos visto en la Sección 2.2.2). Gracias al camino generado podemos construir los contigs y estas a su vez irán formando los Scaffolds para concluir con el proceso de ensamblado.

- **Tercera Fase:** Última fase del desarrollo. En esta fase se intenta realizar el consenso final para ensamblar la secuencia del genoma de forma completa. El grafo es reducido a una sucesión de Scaffolds. La alineación correcta de estos Scaffolds produce la representación del genoma completo.

### 2.3.4 Grafos de Bruijn vs. Grafos en Cadena

**Ejemplo 1:**

En el siguiente ejemplo vamos a comparar el Grafo de Bruijn con los Grafos en Cadena para una misma lectura:

- Valor de k=3
- Múltiples lecturas:
  - ACTGCT
    - CTGCTA
      - GCTAA

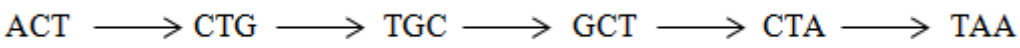



Ilustración 11 Grafo de Bruijn vs Grafo en Cadena (Grafo de Bruijn)

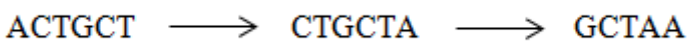



Ilustración 12 Grafo de Bruijn vs Grafo en Cadena (Grafo en Cadena)

**Ejemplo 2: (Lectura con errores)**

- Valor de k =3
- Múltiples lecturas:
    - ACTGCT
        - CTGATA
            - GCTAA

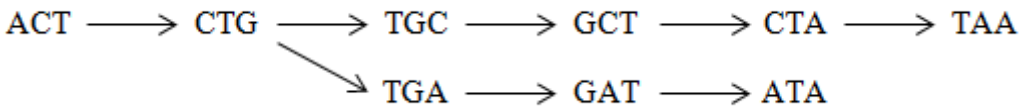



Ilustración 13 Grafo de Bruijn vs Grafo en Cadena (Grado de Bruijn con error en la lectura)

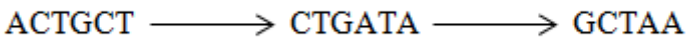



Ilustración 14 Grafo de Bruijn vs Grafo en Cadena (Grado en Cadena con error en la lectura)

Como se puede observar en el ejemplo el grafo en cadena pasa por alto el error en la lectura.

Después de todos estos ejemplos nos podemos realizar la siguiente pregunta ¿Cuál de los dos es mejor? ¿Cuál tenemos que utilizar en nuestro ensamblador?

Ambos tienen sus ventajas:

Por ejemplo los grafos de Bruijn son conceptualmente más simples, tienen una longitud de nodo más simple y la definición de superposiciones también lo es, por lo que podemos conseguir la estructura de datos final del genoma que queremos ensamblar con un menor consumo de memoria que el que podemos encontrar en los grafos en cadena.

En cambio los grafos en cadena permiten capturar toda la información leída y tienen tres fases bien distinguidas de implementación. Gracias a este último detalle podríamos decir que la implementación y optimización de este tipo de ensambladores, a priori, debería ser más fácil que si utilizamos Grafos de Bruijn

Estas pequeñas diferencias hacen que históricamente los grafos de Bruijn hayan sido usados para lecturas cortas y los grafos en Cadena para Lecturas Short Jump.

### 2.3.5 Algoritmos Voraces

Los algoritmos voraces son el tercer tipo de los algoritmos que principalmente se utilizan para realizar ensambladores. Se podría decir que este tipo de algoritmos es de los menos conocidos y de los menos utilizados pero no por ello no deberíamos dar una pequeña referencia sobre ellos.

El funcionamiento principal de estos algoritmos es que se encargan de comprobar que si diferentes lecturas son similares entre sí. Si se da el caso de que unos conjuntos de lecturas son similares se unen entre sí.

Dada una lectura que todavía no se ha ensamblado en un contig, medida por otra lectura o por un contig ya formado que se superpone. Este proceso continua iterativamente comparando todas las posibles superposiciones entre las lecturas. A estas comparaciones se les asigna una puntuación que será utilizada más adelante.

Considerando la superposición con mayor puntuación que se pudiera lograr, las lecturas se combinan entre sí (hemos conseguido encontrar dos lecturas muy similares entre sí).

Este proceso continúa hasta que no es posible realizar más superposiciones ni asignar más puntuaciones a las comparaciones.

Se considera a los algoritmos voraces son considerados como algoritmos de grafos implícitos ya que simplifican el grafo al pasar solo por los extremos con mayor puntuación.

Este tipo de algoritmos para evitar errores tienen un mecanismo que se encarga de parar el proceso cuando se encuentra información contradictoria dentro de las lecturas.

## 2.4 Clasificación de Ensambladores

### 2.4.1 Clasificación por Categorización

En el ensamblado de genes podemos distinguir dos clases principales para su categorización. Estas clases principales son:

- **De-Novo:** Ensamblado de lecturas cortas para crear secuencias de larga duración. Esta categoría tiene la particularidad de que puede ensamblar las secuencias no conocidas a priori sin la ayuda de un genoma de referencia.
  En aspectos matemáticos este tipo de algoritmos se pueden categorizar como problemas del tipo NP completo. Este tipo de problemas matemáticos no tienen una solución computacional eficiente por lo que tienen una complejidad mayor que otro tipo de problemas.
  Aún que este tipo de algoritmos tienen una complejidad más elevada son más óptimos para el ensamblado de secuencias de genes que no tienen un genoma de referencia.

- **Mapping:** Consiste en el ensamblado de secuencias basadas en una secuencia de un genoma pre-existente. Esta secuencia pre-existente se utiliza para alinear la lectura de un nuevo genoma evitando el proceso de creación de las estructuras de datos como utiliza en la categoría anterior.
  La secuencia generada es muy similar a la original pero no idénticas.
  Este tipo de ensamblados son procesos más rápidos y tienen un menor gasto computacional que los anteriores.

| De-Novo | Mapping |
|---|---|
| • No existe información previa del genoma. | • La lectura se alinea contra una referencia. |
| • Necesario para genomas nuevos. | • Usado ampliamente para el estudio del genoma humano. |
| • Las lecturas se asignan en su posición correspondiente. | • Generan contigs que se ensamblan en supercontigs. |
| • Se buscan similitudes entre las lecturas y su referencia. | • Se buscan solapamientos en las lecturas. |
| • Mayor consumo de CPU y memoria. | • Se generan más repeticiones. |

**Tabla 1 Comparación ensambladores De-Novo vs. Mapping**

En el proyecto actual nos centraremos en dos ensambladores del tipo De-Novo para realizar todos los experimentos y comprobaciones necesarias.

### 2.4.2 Clasificación por Características de Ensamblado

La clasificación de ensambladores no es un tema trivial ya que pueden existir muchos puntos a tener en cuenta para realizar dicha clasificación. A parte de una clasificación por categorización podemos considerar entre otras las diferentes características para una clasificación:

- **Número de contigs.**
- **Longitud total del ensamblaje.**
- **Precisión.**
- **Consistencia interna:** Porcentaje de pares correctamente alineados con el ensamblado general. Permite localizar errores en las uniones.
  Para calcular este parámetro podemos utilizar aplicaciones como: REAPR y, FRCurve.
- **Cobertura:** Porcentaje de bases de referencia que están cubiertas por la alineación.
- **N50 de los contigs:**
  - **N50:** El parámetro N50 es una medida de la longitud media de un conjunto de secuencias y se define como el valor X, tal que, al menos la mitad del ensamblado está contenido en contigs de tamaño menor o igual X.
  - **NG50:** El parámetro NG50 es una medida análoga a la anterior pero en vez de considerar la mitad del ensamblado, consideramos le mitad del genoma que estamos ensamblando.

El cálculo del parámetro N50 es un proceso exhaustivo que tiene los siguientes pasos:

1- Clasificamos los contigs en orden descendente por longitud.
2- Cogemos el primer contig (el más grande) y si dicho contig cubre el 50% del ensamblaje es el parámetro N50. Si no se cumple esta restricción seguimos comparando todos los contigs hasta encontrar uno que cubra el 50% del ensamblaje.

### 2.4.3 Clasificación por Tipo de Algoritmo

Como hemos visto en el capítulo 2.3, podemos encontrarnos con varios tipos de algoritmos. Si nos centramos en los tres tipos de algoritmos presentados en este proyecto podemos dar los siguientes ejemplos para cada uno de ellos:

- **Ensambladores de Bruijn:** Estos ensambladores utilizan los algoritmos que realizan Grafos de Bruijn para representar la estructura de datos de todo el proceso de ensamblado.
  Entre los ensambladores que podemos categorizar en este grupo podemos encontrar: EulerSR, Velvet, ABySS, SOAPdenovo, SOAPdenovo2, ALLPATHSLG, Ray y Minia entre otros.

- **Ensambladores en Cadena:** Dentro de este grupo podemos encontrar aquellos ensambladores que utilizan Grafos en Cadena o de Superposición para representar la estructura de datos.
  Entre los más conocidos destacamos: Newbler, Edena, CABOG, Shorty y Forge

- **Ensambladores Voraces:** Ultimo grupo de ensambladores. Como su propio nombre indica en este grupo categorizamos los ensambladores que utilizan algoritmos voraces para construir la estructura de datos.
  Podemos destacar: SSAKE, SHARCGS, VCAKE y QSRA

### 2.4.4 Clasificación por Tipo de Memoria

Como hemos podido ver, hay diversos tipos de clasificación para los ensambladores. Otro de los tipos de clasificación, y puede que el más importantes en aspectos computacionales, es la clasificación por tipo de memoria utilizada para realizar el ensamblaje de genes.

Dentro del tipo de memoria utilizada para realizar el proceso de ensamblaje de genes encontramos los siguientes tipos: memoria compartida, memoria distribuida y computación cloud.

- **Memoria compartida:** Dentro de los diferentes tipos de ensambladores podemos encontrar ensambladores que funcionan en sistemas de memoria compartida bajo interfaces de programación como OpenMP.
  Los sistemas de memoria compartida son aquellos que comparten todas sus tablas de páginas, su memoria virtual y todo lo que forma parte de la memoria RAM.
  Dentro de este grupo podemos encontrar ensambladores como Velvet, ALLPATHS-LG o Minia.

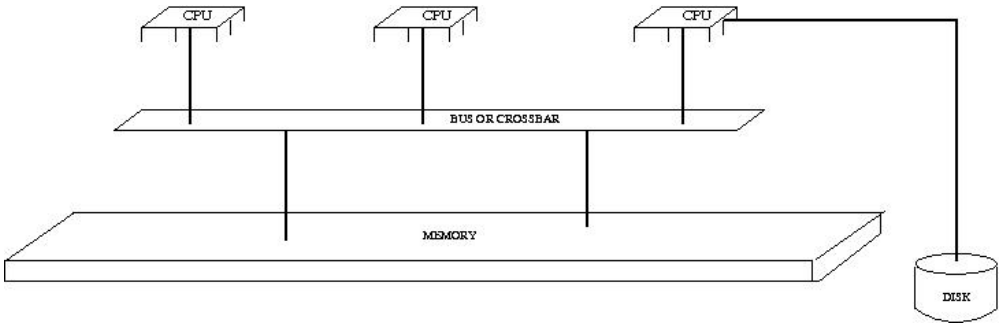



Ilustración 15 Esquema Memoria Compartida

- **Memoria distribuida:** También podemos encontrar ensambladores que utilizan memoria distribuida para realizar sus tareas. La principal característica de los sistemas de memoria distribuida es que son sistemas que utilizan hardware independiente, conectado con redes de comunicaciones de alta capacidad, para conseguir un sistema con mayores prestaciones y haciendo creer al usuario que se encuentra

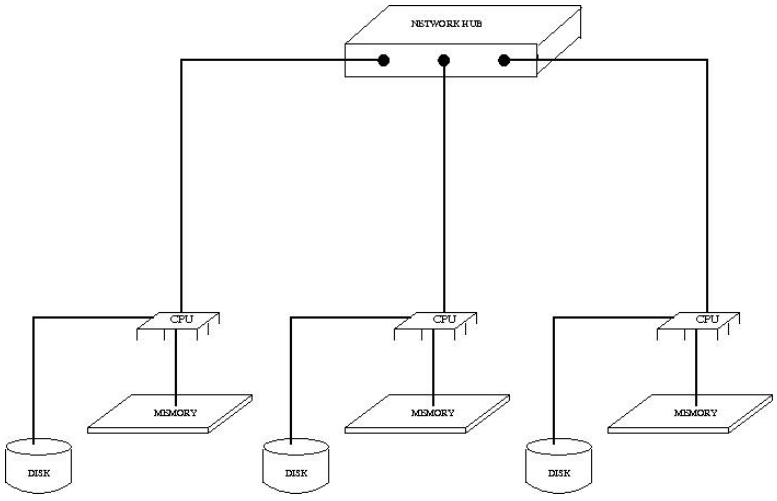



Ilustración 16 Esquema Memoria Distribuida

trabajando con un solo sistema que.
La programación para memoria distribuida se realiza bajo interfaces de programación como MPI.
Dentro de este grupo podemos encontrar ensambladores como ABySS o Ray.

- **Cloud computing:** Finalmente, como último grupo dentro de la clasificación por tipo de memoria, podemos encontrar los sistemas cloud computing. Técnicamente podemos considerar los sistemas de cloud computing como sistemas de memoria distribuida un poco especiales.
  El cloud computing es un nuevo modelo que permite realizar procesamiento masivo de datos de una forma totalmente distinta a la convencional. Anteriormente se tenía que disponer de un supercomputador o un clúster que permitiera realizar este procesamiento de datos. El cloud computing, actualmente, se presenta como un servicio de procesamiento masivo de datos a través de internet que permite realizar todas las operaciones en diversos servidores, de empresas externas como Amazon, que podemos encontrar en la red. La principal ventaja de este sistema es que cualquier usuario sin conocimientos avanzados de programación ni el hardware adecuado puede enviar su información para que otros realicen el procesamiento masivo y almacene sus datos en los servidores de la compañía, por lo que el usuario podrá acceder a sus datos a través de internet.

  Ilustración 17 Cloud Computing

  Para este tipo de paradigma se utilizan tecnologías como hadoop que ofrece un sistema de archivos que permite a las aplicaciones trabajar con miles de nodos y petabytes de datos.
  Ensambladores que utilicen este tipo de tecnología son Contrail y Ray.

## 2.5 Tecnologías Utilizadas

Para realizar el proceso de ensamblado de genomas disponemos de una serie de tecnologías a nuestra disposición. La descripción de las tecnologías más comunes es la siguiente:

| Tecnología | Sanger | Illumina Solexa | Roche 454 |
|---|---|---|---|
| **Máquina de Secuenciación** | 3730xl | HiSeq 2000 | GS FLX Titanium XL+ |
| **Método de Secuenciación** | Terminación de la cadena "Dideoxy" | Secuenciación por Síntesis | Pyrosequencing |
| **Tiempo por Ejecución** | ~23 Horas | ~11 Días | ~23 Horas |
| **Mb (Mega Bases) por Ejecución** | 1.9~84Kb | 600Gb | 700Mb |
| **Longitud de Lectura** | 400-900 Pares de bases | 100 Pares de Bases | 700-1000 Pares de Bases |
| **Coste Por MB** | 1850€ | 0.02€ | 65.12€ |
| **Precisión** | 99.999% | 98% | 99.997% |
| **Coste Instrumentación** | 73315€ | 532510€ | 385860€ |
| **Ventajas** | • Longitud de Lectura<br>• Rendimiento de Alta Calidad | • El mejor Rendimiento<br>• Menor coste por Base | • Lecturas más grandes<br>• Rapidez en Ejecuciones de Secuenciación |
| **Inconvenientes** | • Rendimiento de Baja Calidad<br>• Mayor Coste por Base | • Tasa de Error del 1% | • Rendimiento de Baja Calidad<br>• Tasa de Error del 1% |

Tabla 2 Tecnologías Utilizadas en Ensambladores (I)

| Tecnología | Solid ABI | Ion Torrent | HeliScope |
|---|---|---|---|
| **Máquina de Secuenciación** | 5500 Series | Secuenciador Ion Protón | tSmS |
| **Método de Secuenciación** | Ligadura Basada en Secuencia | Secuenciación de Ion Semiconductor | Secuenciación de Moléculas Individuales |
| **Tiempo por Ejecución** | ~9 Días | ~2 Horas | ~7 Días |
| **Mb (Mega Bases) por Ejecución** | 170Gb | Más de 10 Gb | 21-35Gb |
| **Longitud de Lectura** | 35-75 Pares de Bases | Más de 200 Pares de Bases | 25-55 Pares de Bases |

| | | | |
|---|---|---|---|
| **Coste Por MB** | 0.002€ | 2.5€ | 0.003€ Por Base |
| **Precisión** | Más de 99.99% | 99.6% | 99.995% |
| **Coste Instrumentación** | 460000€ | 115000€ | 770000€ |
| **Ventajas** | • Menor Coste por Base<br>• Rendimiento de Alta Calidad | • Rendimiento de Alta Calidad<br>• Menor Coste de Instrumentación | • Fácil preparación de las Librerías |
| **Inconvenientes** | • Gran Tiempo de Secuenciación | • Gran Coste por Base | • Grandes Tasas de Error |

**Tabla 3 Tecnologías Utilizadas en Ensambladores (II)**

## 2.6 Formatos de Representación de Secuencias de Genes

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 con este fin.

Los principales formatos son: FASTA, FASTQ, GenBank y EMBL. Aunque principalmente se utilizan los dos primeros en la mayoría de experimentos con ensambladores.

### 2.6.1 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 distinta 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.

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

Ilustración 18 Ejemplo Formato Fasta.

### 2.6.2 FASTQ

Es un formato de texto que permite almacenar la secuencia biológica (secuencia de nucleótidos generalmente) 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
```

Ilustración 19 Ejemplo Formato FASTQ

### 2.6.3 GenBank

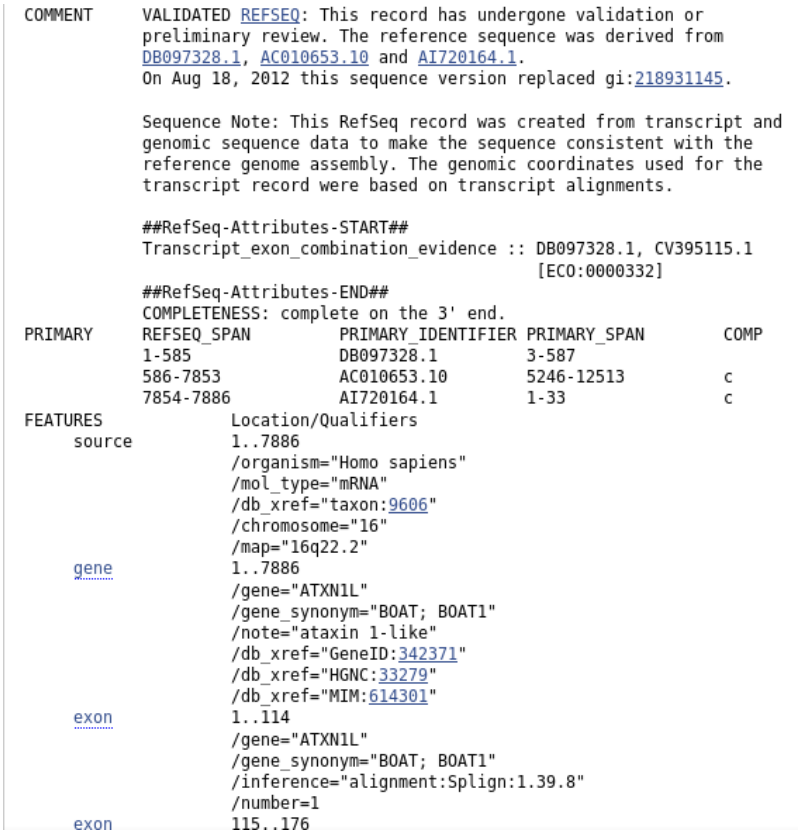
El archivo GenBank al igual que el archivo FASTA o FASTQ almacena información de un secuencia. A diferencia de los dos anteriores este si almacena muchos de los datos correspondientes a un secuencia, por ejemplo, la descripción, publicaciones que se han hecho estudiando la secuencia, el organismo al cual pertenece y muchos otros datos.

La representación de las secuencias con este formato tiene tres partes bien diferenciadas:

```
LOCUS       NM_001137675           7886 bp    mRNA    linear   PRI 13-JAN-2013
DEFINITION  Homo sapiens ataxin 1-like (ATXN1L), mRNA.
ACCESSION   NM_001137675 XM_928464
VERSION     NM_001137675.3  GI:401709969
KEYWORDS    .
SOURCE      Homo sapiens (human)
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini;
            Catarrhini; Hominidae; Homo.
REFERENCE   1  (bases 1 to 7886)
  AUTHORS   Oshikawa,M., Tsutsui,C., Ikegami,T., Fuchida,Y., Matsubara,M.,
            Toyama,S., Usami,R., Ohtoko,K. and Kato,S.
  TITLE     Full-length transcriptome analysis of human retina-derived cell
            lines ARPE-19 and Y79 using the vector-capping method
  JOURNAL   Invest. Ophthalmol. Vis. Sci. 52 (9), 6662-6670 (2011)
   PUBMED   21697133
  REMARK    Publication Status: Online-Only
REFERENCE   2  (bases 1 to 7886)
  AUTHORS   Tong,X., Gui,H., Jin,F., Heck,B.W., Lin,P., Ma,J., Fondell,J.D. and
            Tsai,C.C.
  TITLE     Ataxin-1 and Brother of ataxin-1 are components of the Notch
            signalling pathway
  JOURNAL   EMBO Rep. 12 (5), 428-435 (2011)
   PUBMED   21475249
  REMARK    GeneRIF: Both ATXN1 and BOAT1 bind to the promoter region of Hey1
            and inhibit the transcriptional output of Notch through direct
            interactions with CBF1, a transcription factor that is crucial for
            the Notch pathway.
```

Ilustración 20 Primera Parte Formato GenBank (Introducción)

En esta parte del archivo se muestra el número de acceso a la secuencia el cual es un identificador único en diferentes bases de datos, también se puede ver que muestra el organismo del cual proviene el gen, en este caso, Homo Sapiens, luego muestra los autores de estas secuencias, el título de la misma y algunas publicaciones de PubMed que hablan o hacen referencia a esta secuencia.



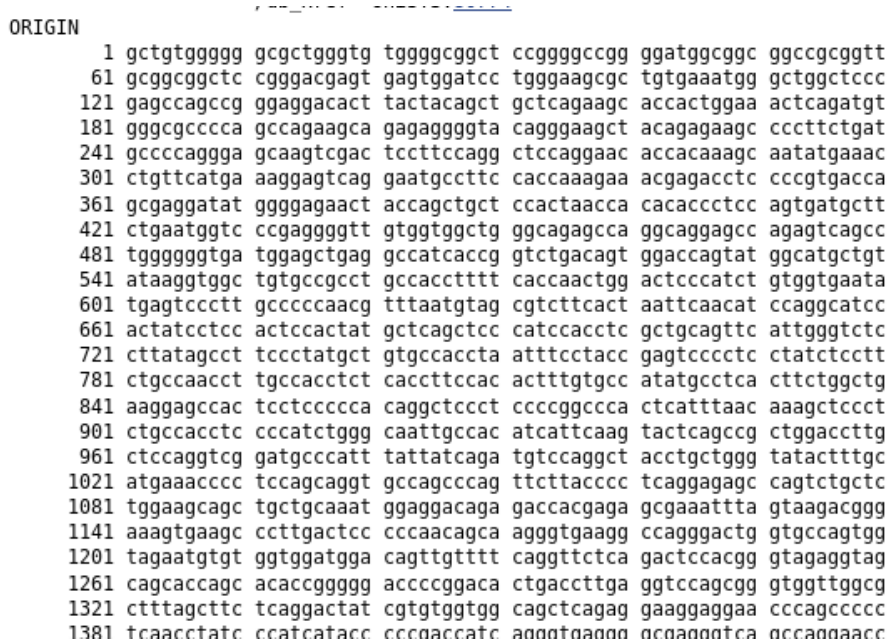
```
COMMENT     VALIDATED REFSEQ: This record has undergone validation or
            preliminary review. The reference sequence was derived from
            DB097328.1, AC010653.10 and AI720164.1.
            On Aug 18, 2012 this sequence version replaced gi:218931145.

            Sequence Note: This RefSeq record was created from transcript and
            genomic sequence data to make the sequence consistent with the
            reference genome assembly. The genomic coordinates used for the
            transcript record were based on transcript alignments.

            ##RefSeq-Attributes-START##
            Transcript_exon_combination_evidence :: DB097328.1, CV395115.1
                                                    [ECO:0000332]
            ##RefSeq-Attributes-END##
            COMPLETENESS: complete on the 3' end.
PRIMARY     REFSEQ_SPAN         PRIMARY_IDENTIFIER PRIMARY_SPAN        COMP
            1-585               DB097328.1         3-587
            586-7853            AC010653.10        5246-12513          c
            7854-7886           AI720164.1         1-33                c
FEATURES             Location/Qualifiers
     source          1..7886
                     /organism="Homo sapiens"
                     /mol_type="mRNA"
                     /db_xref="taxon:9606"
                     /chromosome="16"
                     /map="16q22.2"
     gene            1..7886
                     /gene="ATXN1L"
                     /gene_synonym="BOAT; BOAT1"
                     /note="ataxin 1-like"
                     /db_xref="GeneID:342371"
                     /db_xref="HGNC:33279"
                     /db_xref="MIM:614301"
     exon            1..114
                     /gene="ATXN1L"
                     /gene_synonym="BOAT; BOAT1"
                     /inference="alignment:Splign:1.39.8"
                     /number=1
     exon            115..176
```

**Ilustración 21 Segunda Parte Formato GenBank (Comentarios)**

Esta es la segunda parte del archivo tenemos comentarios acerca de la secuencia y algunas de sus características.



```
ORIGIN
        1 gctgtggggg gcgctgggtg tggggcggct ccggggccgg ggatggcggc ggccgcggtt
       61 gcggcggctc cgggacgagt gagtggatcc tgggaagcgc tgtgaaatgg gctggctccc
      121 gagccagccg ggaggacact tactacagct gctcagaagc accactggaa actcagatgt
      181 gggcgcccca gccagaagca gagaggggta cagggaagct acagagaagc cccttctgat
      241 gccccaggga gcaagtcgac tccttccagg ctccaggaac accacaaagc aatatgaaac
      301 ctgttcatga aaggagtcag gaatgccttc caccaaagaa acgagacctc cccgtgacca
      361 gcgaggatat ggggagaact accagctgct ccactaacca cacaccctcc agtgatgctt
      421 ctgaatggtc ccgaggggtt gtggtggctg ggcagagcca ggcaggagcc agagtcagcc
      481 tgggggggtga tggagctgag gccatcaccg gtctgacagt ggaccagtat ggcatgctgt
      541 ataaggtggc tgtgccgcct gccacctttt caccaactgg actcccatct gtggtgaata
      601 tgagtccctt gcccccaacg tttaatgtag cgtcttcact aattcaacat ccaggcatcc
      661 actatcctcc actccactat gctcagctcc catccacctc gctgcagttc attgggtctc
      721 cttatagcct tccctatgct gtgccaccta atttcctacc gagtcccctc ctatctcctt
      781 ctgccaacct tgccacctct caccttccac actttgtgcc atatgcctca cttctggctg
      841 aaggagccac tcctccccca caggctccct ccccggccca ctcatttaac aaagctccct
      901 ctgccacctc cccatctggg caattgccac atcattcaag tactcagccg ctggaccttg
      961 ctccaggtcg gatgcccatt tattatcaga tgtccaggct acctgctggg tatactttgc
     1021 atgaaacccc tccagcaggt gccagcccag ttcttacccc tcaggagagc cagtctgctc
     1081 tggaagcagc tgctgcaaat ggaggacaga gaccacgaga gcgaaattta gtaagacggg
     1141 aaagtgaagc ccttgactcc cccaacagca agggtgaagg ccagggactg gtgccagtgg
     1201 tagaatgtgt ggtggatgga cagttgtttt caggttctca gactccacgg gtagaggtag
     1261 cagcaccagc acaccggggg accccggaca ctgaccttga ggtccagcgg gtggttggcg
     1321 ctttagcttc tcaggactat cgtgtggtgg cagctcagag gaaggaggaa cccagccccc
     1381 tcaacctatc ccatcatacc cccgaccatc agggtgaggg gcgagggtca gccaggaacc
```

**Ilustración 22 Tercera Parte Formato GenBank (Secuencia)**

La tercera, y última parte de un fichero con formato GenBank contiene la secuencia propiamente dicha.

### 2.6.4 EMBL

EMBL es un formato creado por "European Molecular Biology Laboratory" este formato al igual que los anteriores contiene información de una secuencia. A continuación se describen algunos de sus campos.

```
ID   16    standard; DNA; HTG; 11338 BP.
XX
AC   chromosome:GRCh37:16:71879894:71891231:1
XX
SV   chromosome:GRCh37:16:71879894:71891231:1
XX
DT   3-FEB-2013
XX
DE   Homo sapiens chromosome 16 GRCh37 partial sequence 71879894..71891231
DE   annotated by Ensembl
XX
KW   .
XX
OS   Homo sapiens (human)
OC   Eukaryota; Opisthokonta; Metazoa; Eumetazoa; Bilateria; Coelomata;
OC   Deuterostomia; Chordata; Craniata; Vertebrata; Gnathostomata; Teleostomi;
OC   Euteleostomi; Sarcopterygii; Tetrapoda; Amniota; Mammalia; Theria;
OC   Eutheria; Euarchontoglires; Primates; Haplorrhini; Simiiformes; Catarrhini;
OC   Hominoidea; Hominidae; Homininae.
XX
```

**Ilustración 23 Primera Parte Formato EMBL (Introducción)**

En esta parte se destaca la línea OS seguida de las líneas OC. La línea OS corresponde a (Organism Species), esta especifica el nombre preferido del organismo que tiene esta secuencia, las líneas OC (Organism Classification) corresponde a la clasificación del organismo, como se pude notar en la imagen se ven algunas de las clasificaciones en las que está el Homo Sapiens, por ejemplo, primate, homínido, vertebrado.

```
CC   All the exons and transcripts in Ensembl are confirmed by similarity to
CC   either protein or cDNA sequences.
XX
FH   Key             Location/Qualifiers
FT   source          1..11338
FT                   /organism="Homo sapiens"
FT                   /db_xref="taxon:9606"
FT   gene            1..11338
FT                   /gene=ENSG00000224470
FT                   /locus_tag="ATXN1L"
FT                   /note="ataxin 1-like [Source:HGNC Symbol;Acc:33279]"
FT   mRNA            join(1..114,2020..2081,3634..11338)
FT                   /gene="ENSG00000224470"
FT                   /note="transcript_id=ENST00000427980"
FT   CDS             3751..5820
FT                   /gene="ENSG00000224470"
FT                   /protein_id="ENSP00000415822"
FT                   /note="transcript_id=ENST00000427980"
FT                   /db_xref="CCDS:CCDS45523.1"
FT                   /db_xref="Uniprot/SWISSPROT:ATX1L_HUMAN"
FT                   /db_xref="RefSeq_peptide:NP_001131147.1"
FT                   /db_xref="RefSeq_mRNA:NM_001137675.3"
FT                   /db_xref="Vega_transcript:ATXN1L-001"
FT                   /db_xref="Vega_transcript:OTTHUMT00000434171"
FT                   /db_xref="Uniprot/SPTREMBL:G1UI23_HUMAN"
FT                   /db_xref="HGNC:ATXN1L"
```

**Ilustración 24 Segunda Parte Formato EMBL (Comentarios)**

De esta parte se destaca el CC que son comentarios relevantes de la secuencia, también las líneas FT (Feature Table) proveen algunos datos de anotación de la secuencia, en este caso como encontrarlas en otras bases de datos de secuencias, por ejemplo, RefSeq y Uniprot.

```
FT                   /note= trf repeat: matches 1..58(1) of consensus
FT   repeat_region   complement(11283..11323)
FT                   /note="trf repeat: matches 1..41(1) of consensus"
FT   STS             complement(11184..11283)
FT                   /standard_name="NIB1158"
FT                   /db_xref="UniSTS_NUM:80774"
FT                   /db_xref="UniSTS:NIB1158"
FT                   /db_xref="UniSTS:T16346"
FT                   /note="map_weight=1"
FT   misc_feature    1..11338
FT                   /note="contig AC010653.10 5218..16555(-1)"
XX
SQ   Sequence 11338 BP; 2700 A; 2698 C; 2904 G; 3036 T; 0 other;
     GCTGTGGGGG GCGCTGGGTG TGGGGCGGCT CCGGGGCCGG GGATGGCGGC GGCCGCGGTT        60
     GCGGCGGCTC CGGGACGAGT GAGTGGATCC TGGGAAGCGC TGTGAAATGG GCTGGTGAGT       120
     GTCCCTGGAA GTGGTGGCCG CCGGGGCCAG GCAGGCCTGG GGTCGGTGCT CCGGAGGAAG       180
     TGGCCCTCAG GAGCCCGTTG CTCCGGCGGG CCGGGGCCCA TGAACCGTGG ACAGACGGAG       240
     GGGAGAGGCC GCACCTAGAC CCCCTCGCGC TTCTCTCTGG CCTGTCCTGG AAGGTGGCCT       300
     CAGGCCCCGG CGCCCCGGAG GCTGTGTGCC TGCGAGCCAA GGTGTCCGAG TCTGTGGGGC       360
     GGGAAGGGCC CCCAAACAAG GAAGCGTCGC CTGCAGGGCC TCGCCCCTGC CCCCTTGGTG       420
     AAGGGTGGAC CCCCTACTCT CTAAACGCAG AATTCCCAAT TTCGAGCCCT ACCCTTGTTG       480
     GATCAAGGAT TGCGGGACCA CTGGGAGCCA TAACCGTAGT CTCTGACAAG CTTCTCCCTT       540
     TCTGTCGGGC GCACCTCTCG CCCGCAGAGA ATTTGGGTCT CCAGCTGGGT TCAGCCCCTT       600
     GATTCCGACC CAAGCGTGAT GAGGAGCTTG GTTCCGGGCT CTGCTAGCAG CGTTGGCAAC       660
     TCTCCTGAAC AGCCTGTTAG CTCGGTTGGG GTCACAAGCT GCGTTTTCCC TTTGAGATGC       720
```

**Ilustración 25 Tercera Parte Formato EMBL (Secuencia)**

La línea SQ como tal corresponde a la secuencia que está contenida en el registro.

## 2.7 Hardware y Software Utilizado en las Pruebas

En este apartado vamos a especificar todos los detalles acerca del hardware y el software utilizado en los test del presente proyecto.

### 2.7.1 Magerit 2

Para la realización de los test vamos a utilizar el supercomputador de la Universidad Politécnica de Madrid Magerit2.

Actualmente el supercomputador Magerit2 es uno de los más potentes de España y del mundo. De este supercomputador podemos destacar los siguientes datos:


Ilustración 26 Supercomputador Magerit 2

- **Configuración Hardware:**

Magerit está compuesto por 245 nodos eServer BladeCenter PS702 cada uno de los cuales dispone de 16 procesadores PPC de 3'3 GHz (294 GFlops) con 32 GB de RAM. Para su interconexión se utiliza una red Infiniband de fibra Óptica de altas prestaciones junto con redes auxiliares Gigabit para su control y gestión.

El sistema dispone de una capacidad de almacenamiento local de unos 192 TB, proporcionado por 256 discos de 750 GB, que utiliza un sistema distribuido y tolerante a fallos (GPFS).

Aunque todos los nodos tienen una configuración hardware y software idéntica, se dividen en dos funcionalidades básicas:

- **Interactivos o de login:** Tienen habilitado el acceso al exterior y son utilizados como punto de entrada al sistema. En ellos se realizan labores de edición, compilación, gestión de trabajos e intercambio de ficheros.
  En estos nodos no se permite la ejecución de servicios o cálculos, que son abortados de forma automática.

- **Cómputo:** Tienen como única misión ejecutar los trabajos de usuario. Estos nodos están completamente aislados del exterior y sólo son accesibles desde el gestor de trabajos Ejecución de trabajos.

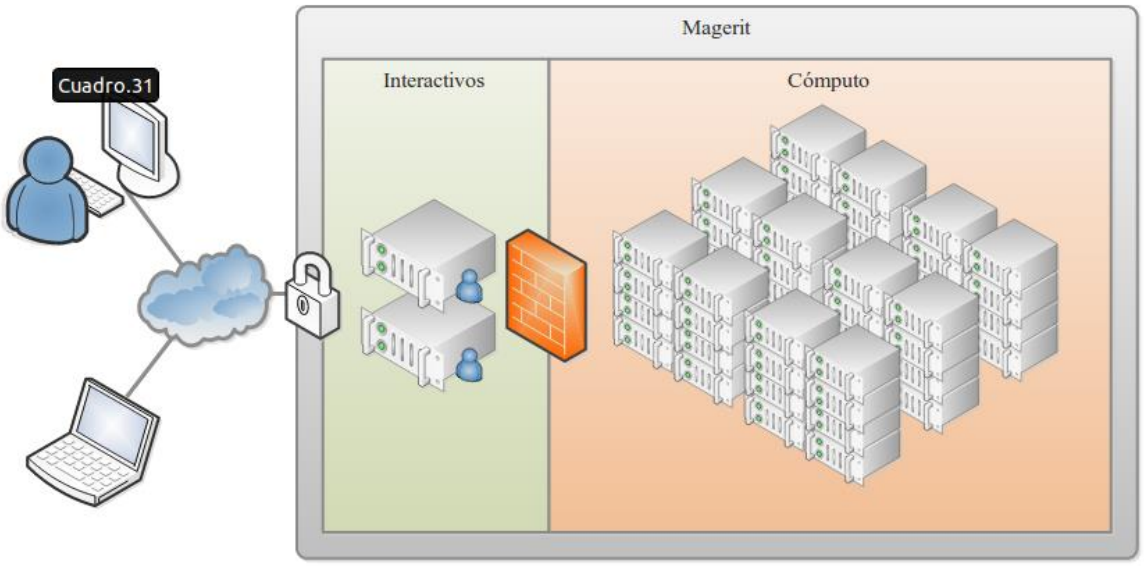



Ilustración 27 Estructura Interna Magerit 2

La conexión exterior se realiza a través de RedIRIS mediante enlaces de 1Gb y 10 Gb

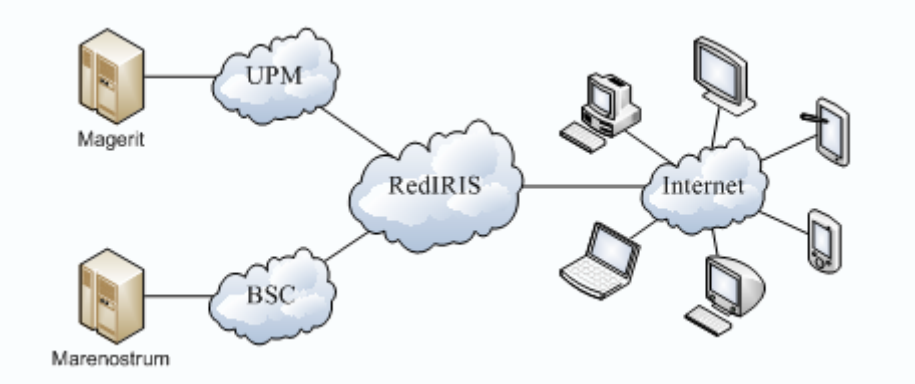



Ilustración 28 Esquema arquitectura Magerit 2

- **Configuración Software**

  Todos los nodos tienen instalado un sistema operativo SLES11SP1 (Kernel 2.6.32). Además de todo el entorno de trabajo estándar de un sistema Linux, se dispone de numerosas aplicaciones y librerías de cálculo científico.

  Es posible solicitar la instalación de un nuevo software en el sistema contactando con el centro de atención a usuarios.

  Todos los nodos tienen acceso a un espacio de almacenamiento compartido y distribuido con una capacidad bruta total de casi 190 TB, proporcionado por 256 discos de 750 GB, que utiliza un sistema distribuido y tolerante a fallos denominado GPFS [IBM-GPFS].

- **Comunicaciones**
  Para la interconexión de todos los elementos se dispone de dos enrutadores para redes Ethernet y un tercero de Infiniband. Estos enrutadores dan soporte a dos redes diferentes accesibles desde los nodos:

- **Red Infiniband:** Es una red de alto rendimiento utilizada para las comunicaciones de las aplicaciones paralelas. Proporciona un ancho de banda de 40 Gbps con latencias inferiores a 1 microsegundo.
- **Red Ethernet:** Se trata de dos redes que se utilizan para cargar el sistema operativo y acceso al sistema de ficheros GPFS, respectivamente.

Los nodos interactivos disponen de una tercera conexión de red Ethernet hacia el exterior

### 2.7.2 SLURM (Simple Linux Utility for Resource Management)

SLURM es un sistema de código abierto para la gestión de clústeres Linux. Este sistema proporciona todas las funciones básicas para la gestión de un clúster. Como por ejemplo:

- Proporciona funciones de asignación exclusiva o no exclusiva de los recursos disponibles por el clúster en un periodo de tiempo.
- Proporciona un marco para el arranque, ejecución y seguimiento de las acciones realizadas por cada usuario.
- Gestiona las tareas pendientes que pueden crear conflictos organizándolas en una cola.
  La prioridad que se asigna a cada tarea dentro del a cola viene determinada por una serie de parámetros definidos por los recursos que necesita la tarea, el modo en el que el usuario utiliza los recursos que tiene a sus disposición para ejecutar las tareas pendientes. Por ejemplo los usuarios que necesiten menor tiempo de ejecución tienen prioridad sobre usuarios que necesitan más tiempo de ejecución. Cabe de estacar que este tipo de soluciones solo se utilizan si no hay recursos disponibles para satisfacer las peticiones realizadas por todos los usuarios.

La gestión que proporciona SLURM es una gestión centralizada, es decir todos los nodos y recursos están gestionados por un solo centro de gestión centralizado. Aún que cada nodo ejecuto un demonio interno de SLURM denominado slurmd que controla las tareas que se van a ejecutar en dicho nodo.

Dentro de las posibilidades que proporciona SLURM podemos destacar la siguiente serie de comandos:

- **Jobsubmit:** Presentar una tarea.
- **Jobcheck:** Comprobar el estado de una tarea.
- **Jobhold:** Bloquear / desbloquear una tarea.
- **Jobq:** Lista de tareas en ejecución.
- **Jobcancel:** Cancelar un trabajo.
- **Sbatch:** Enviar script de configuración al sistema de colas.

### 2.7.3 Ejecutar Tareas en Magerit 2.

Para añadir tareas de ejecución de Magerit2 hay que realizar una configuración previa de todos los requisitos que queramos configurar de nuestra tarea. Por ejemplo podemos modificar el número de procesadores que queremos utilizar en la ejecución, límite de tiempo de trabajo…

Una vez configuradas todos aspectos previos tenemos que mandar nuestra tarea al gestor de colas de Magerit2.

El gestor de colas de Magerit2 es el encargado de gestionar todas las peticiones, de todos los usuarios que quieran ejecutar sus tareas en Magerit2 y ofrecer la mejor solución posible a cada uno de ellos.

Los pasos que el usuario debe hacer para ejecutar un trabajo en Magerit2 son los siguientes:

- Conectarse a un nodo.
- Preparar el archivo ejecutable que se quiere mandar al supercomputador.
- Preparar la definición de trabajo.
- Enviar el trabajo al gestor de colas.
- Esperar a que el sistema ejecute la tarea.
- Recuperar los resultados.

Dentro de Magerit2 cada usuario, dependiendo del proyecto o de la tarea que esté realizando, podrá configurar una serie de parámetros de calidad de servicio dentro de los límites de autorización que tiene pre-establecidos con anterioridad.

En la siguiente tabla podemos observar los distintos niveles de calidad de servicio disponibles en el sistema:

| Calidad de Servicio | Cpus | Tiempo | Prioridad |
|---|---|---|---|
| Debug | 16 | 00:10:00 | - |
| Class_a | 1024 | 72:00:00 | Baja |
| Class_b | 1024 | 36:00:00 | Media |
| Class_c | 1024 | 24:00:00 | Alta |
| Standard | 512 | 72:00:00 | Alta |
| Premium | 1024 | 72:00:00 | Media |
| Partner | 512 | 24:00:00 | Baja |

Tabla 4 Niveles de Calidad de Servicio Magerit 2

Para la presentación de las tareas en Magerit2 hay que generar unos scripts de ejecución con una estructura bien definida. Las declaraciones óptimas vienen definidas a continuación.

- **#!/bin/bash:** Indica que terminal se encuentra instalada por defecto en el sistema. Por defecto todos los sistemas que tienen instalada el terminal bash lo tienen en el directorio /bin/
- **#@ class:** Indica la calidad de servicio.

- **#@ initialdir:** Indica el directorio de trabajo del script. Todas las rutas especificadas (salida, error…) son relativas a este directorio. Si no se especifica se considera el directorio de trabajo el directorio actual.
- **#@ output:** Indica el directorio de salida de todos los procesos ejecutados.
- **#@ error:** Indica el directorio de error de los procesos ejecutados.
- **#@ total_tasks:** Indica el número de CPUs que son necesarias. El número máximo viene determinado por la calidad de servicio. Este campo es importante para plataformas MPI de memoria distribuida
- **#@ cpus_per_task:** Indica el número de Indica el número de threads OpenMP que deben lanzarse por cada tarea (todos los hilos de cada tarea ejecutan en el mismo nodo). Esta opción es importante para sistemas de memoria compartida
- **#@ Wall_clock_limit:** Indica el tiempo límite de una tarea. El valor máximo viene determinado por la calidad de servicio.
- **export OMP_NUM_THREAD=:** Número de threads que pueden ser creados es análogo a la opción cpus_per_task.
- **srun./[miprograma]:** Comando que permite la ejecución del programa generado.

```
#!/bin/bash
#---------------------- Start job description ----------------------
#@ class            = [qos_name]
#@ initialdir       = /gpfs/projects/[project_id]/[data_dir]
#@ output           = res/out-%j.log
#@ error            = res/err-%j.log
#@ total_tasks      = [number of tasks]
#@ wall_clock_limit = [hh:mm:ss]
#----------------------- End job description -----------------------

#-------------------------- Start execution --------------------------

# Run our program
srun ./[myprogram]

#--------------------------- End execution ---------------------------
```

Ilustración 29 Estructura Script de Tareas Magerit

Una vez realizado el script con toda la información necesaria para ejecutar una tarea o serie de tareas, hay que enviar dicho script al gestor de colas SLURM para que lo procese y asigne recursos cuando se cumplan las restricciones de prioridad.

Para enviar el script al sistema hay que ejecutar el siguiente comando:

- **sbatch "nombre_fichero"**

Una vez enviado el script de ejecución solo hay que esperar que la tarea concluya. El sistema nos avisará con un correo email a la cuenta de correo electrónico con la que hemos completado el registro para poder utilizar Magerit.

También, en todo momento, podemos comprobar el estado del a ejecución de la tarea con el siguiente comando:

- **jobcheck “identificador de trabajo”**

Una vez completada la ejecución de todas las tareas especificadas en el script de configuración se generará un fichero con el siguiente formato:

- **slurm-“identificador”.out**

En dicho fichero podemos observar toda la información relevante de la ejecución de la tarea incluidos los errores en la ejecución, si los hubiera.

### 2.7.4 Ensamblador Minia

El ensamblador Minia es un ejemplo bastante peculiar de los ensambladores del tipo de Novo.

Este ensamblador utiliza una nueva codificación para el Grafo de Bruijn que permite realizar su tarea con la utilización de menos memoria RAM y teóricamente con un tiempo igual o menor que otros ensambladores del mismo tipo.

Según podemos observar en la documentación oficial de este ensamblador (Ensamblador Minia) nos indica que la nueva codificación se basa en un filtro de Bloom que permite, con una estructura de datos adicional, eliminar los falsos positivos críticos del Grafo de Bruijn original.

Gracias a esto es posible simplificar al máximo el consumo de memoria RAM, y al tener menos errores en la estructuras de datos es posible realizar el proceso de ensamblaje más rápido.

Cómo hemos visto en secciones anteriores la corrección de errores muy importantes para evitar datos innecesarios, bucles en el Grafo, etc.

El filtro de Bloom es un array de bits, basado en una estructura de tabla hash, que representa cualquier conjunto de datos con una precisión $\epsilon$.

En este filtro una proporción de $\epsilon$ se consideran falsos positivos que hay que tener en cuenta y tratarlos y/o eliminarlos para optimizar los recursos.

Para representar un conjunto de datos de n elementos se requiere aproximadamente $1.44 \log_2(\frac{1}{\epsilon})$. n bits utilizando esta codificación con optimización de errores.

El consumo de memoria que utiliza la tecnología Minia se puede simplificar de la siguiente manera:

- **Lista explicita:** 2k * n bits
- **Auto-información de n nodos:** $\mathbf{\log_2((\frac{4^k}{n}))}\ \boldsymbol{bits}$

**Ejemplo:**

En el siguiente ejemplo podemos observar un Grafo de Bruijn para un valor de K = 3.

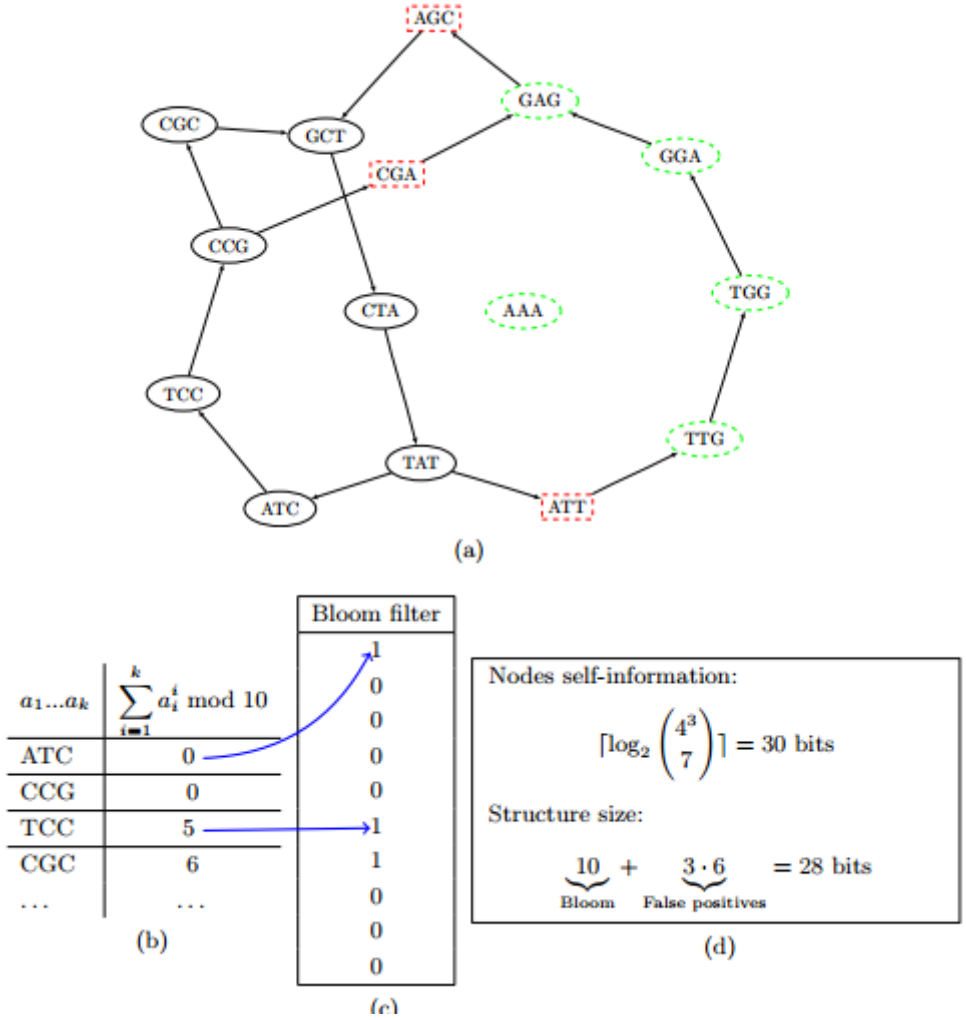



Ilustración 30 Ejemplo de la Estructura de Datos del Ensamblador Minia

En el ejemplo de la imagen anterior se puede ver la resolución del problema de falsos positivos por parte del ensamblador Minia.

En la imagen a) podemos observar el Grafo de Bruijn para una secuencia dada. Cómo podemos observar en principio 7 nodos no discontinuos (nodos de color negro), 3 nodos considerados como falsos positivos (nodos de color rojo) y 5 nodos circulares discontinuos que completan el Grafo de Bruijn (nodos de color verde).

En la siguiente imagen se muestra una asociación ficticia de los nodos en una tabla hash para proporcionar los valores necesarios al filtro de Bloom

En la imagen c) podemos ver el filtro de Bloom, en el filtro de Bloom se marcan como positivos aquellos nodos no discontinuos que vimos en la imagen a).

Finalmente en la imagen d) podemos observar el límite inferior para la codificación de los nodos y el espacio requerido para codificar la estructura de Bloom 10 bits para la estructura del filtro de Bloom y 18 bits para falsos positivos.

- **Parámetros de Entrada:**

    - **Fichero de Entrada:** Es el parámetro donde indicamos el fichero de entrada de las lecturas. Puede ser un fichero del tipo FASTA/FASTQ un archivo comprimido en formato gzip o un archivo de texto donde en cada línea podemos encontrar un archivo de lectura distinto.

    - **Tamaño Kmer:** La longitud k-mer es la longitud de los nodos en el grafo de Bruijn. Este valor depende en gran medida del conjunto de datos de entrada.

    - **Mínima Abundancia:** La mínima abundancia es usada para eliminar kmers de baja repetición (corresponde con la cantidad más pequeña de veces que una lectura kmer correcta aparece en el conjunto de datos). Este parámetro depende de forma muy importante del conjunto de datos.

    - **Tamaño de Genoma Estimado:** Este parámetro solo controla el uso de memoria durante la primera fase de Minia (construcción del grafo de Bruijn). No tiene ningún impacto en el conjunto de ensamblaje.

    - **Prefijo:** Es un nombre arbitrario utilizado para nombrar los ficheros de salida.

- **Salida:**

La salida que devuelve el ensamblador minia es un conjunto de contigs codificadas en formato FASTA. El nombre del fichero es: [prefijo].contigs.fa

### 2.7.5 Ensamblador Velvet

Velvet es un conjunto de algoritmos escritos en lenguaje de programación C creado para manipular Grafos de Bruijn para el ensamblado de genes.

El ensamblador Velvet puede ensamblar cualquier tipo de lecturas, pero en realidad está diseñado para el ensamblaje de lecturas cortas que van desde 25 a 50pb.

La principal ventaja de este ensamblador es que, al utilizar Grafos de Bruijn, puede eliminar errores producidos por la máquina de secuenciación y resuelve repeticiones causadas por la complejidad del genoma. Estas dos características se desarrollan en dos etapas distintas del proceso de ensamblaje.

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.

A continuación vamos a especificar las etapas que sigue el ensamblador Velvet para producir el ensamblaje de genes. Al ser un ensamblador que utiliza Grafos de Bruijn sus etapas son muy similares a las etapas genéricas que pudimos observar en la Sección 2.3.2.

- **Primera Etapa:** Etapa en la que desarrollaremos la estructura de datos general que va a utilizar el ensamblador en el resto de etapas para realizar su tarea. Cómo hemos comentado antes en esta etapa se puede producir un cuello de botella.
  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 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 le 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.

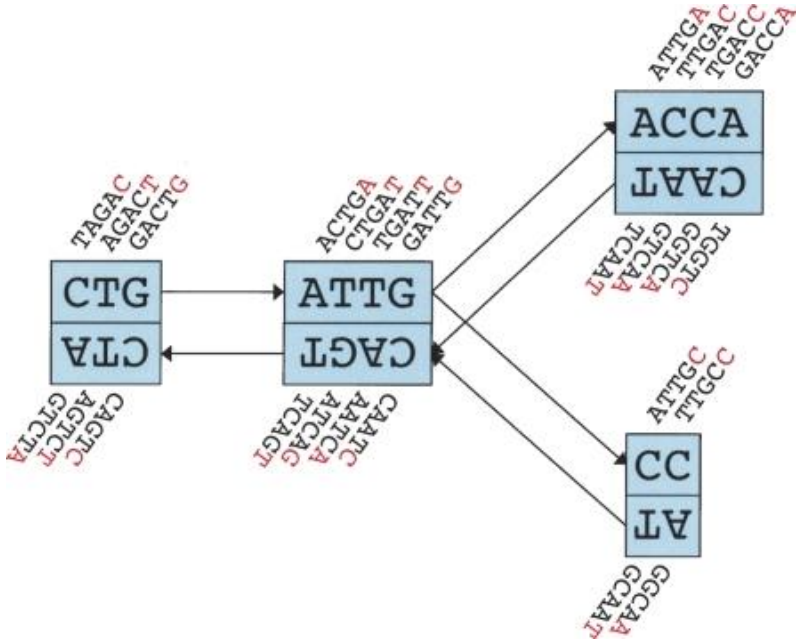



Ilustración 31 Ejemplo del Grafo de Bruijn Velvet para k = 5

- **Segunda Etapa:** En esta segunda 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 tenemos que considerar lo siguiente. Si tenemos un nodo N1 que tiene solo una conexión saliente con un nodo N2 y dicho nodo N2 solo tiene una conexión entrante. Podemos simplificar ambos nodos en un solo nodo N3 que combine la información de los nodos anteriores.

- **Tercera Etapa:** En esta tercera etapa ya disponemos de un Grafo de Bruijn simplificado. Ahora tenemos que corregir todos los errores que podamos encontrar en dicho grafo.
  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 parámetro a considerar para eliminar dichos caminos es la longitud de los mismos. Si encontramos alguno con longitud menor que 2k pb podemos asegurar que es un error y no tenemos por qué tenerlo en cuenta. Si se da el caso de que hay caminos de longitud mayor que son errores deberíamos reconsiderar cambiar el parámetro K.
  Otro de los errores que podemos encontrar en un Grafo de Bruijn, es el denominado "error burbuja". Para tener este tipo de errores tenemos que tener conexiones en el grafo que tienen

el mismo nodo de inicio y mismo nodo final pero en medio tienen distinta información. Esto puede ser debido a errores en el medio de las lecturas o de los k-mers. Para solucionar este tipo de errores se utiliza el algoritmo "Tour Bus". Dicho algoritmo tiene el siguiente funcionamiento:

El algoritmo realiza una primera búsqueda en el grafo utilizando el algoritmo de Dijkstra (Algoritmo de Dijkstra). Debido al realizar este tipo de búsquedas podemos dar prioridad a las rutas más. Cuando el procedimiento encuentra un nodo por el que se ha pasado antes se retrocede al ancestro más cercano. Una vez hecho esto, los dos caminos que acceden al nodo que causa la marcha atrás se extraen alineados y si son similares se fusionan.

Tras la fusión de los dos nodos encontrados hay que volver a reorganizar las conexiones con los nodos que tenían conexión con los nodos antes de fusionarse. En rutas lineales (rutas que no tienen nodos con más de una visita del algoritmo) es fácil ya que solo hay que cambiar la posición en la secuencia resultante tras la fusión. En cambio, si hay nodos visitados por el algoritmo de Dijkstra, tenemos el algoritmo Tour Bus realiza una tarea mucho más compleja que consiste en marcar todos los nodos contenidos en los dos caminos y comienza un proceso de fusión desde un extremo a otro visitando todos los nodos de manera consecutiva. Cada nodo de la rama minoritaria (rama con menos prioridad) se compara con el nodo correspondiente de la secuencia de consenso y se transfiere toda la información del nodo al nuevo nodo de consenso.

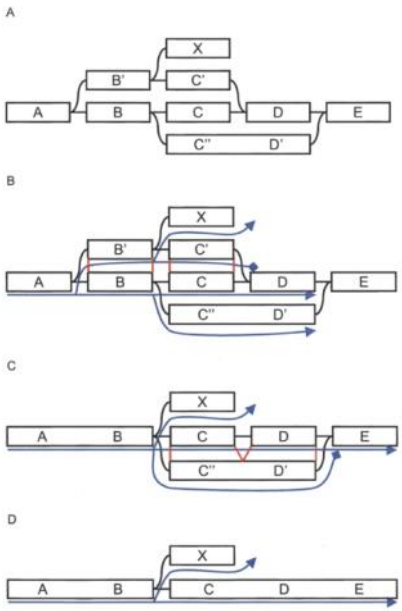



Ilustración 32 Ejemplo de corrección de errores mediante el algoritmo Tour Bus

A) Grafo Original

B) La búsqueda empieza en A y se propaga hacia la derecha. La progresión de la trayectoria superior (a través de B' y C') se detiene debido a que D ha sido visitado previamente. Las secuencias de nucleótidos correspondientes a la alternativa de caminos B' y C' y BC se extraen del grafo alineados y se comparan

C) Los dos caminos son muy similares, por lo que el camino más largo B' y C' se combina con el más corto B y C. La fusión está dirigida por la alineación de las secuencias de consenso, se indica en líneas rojas en B. Hay que tener en cuenta que el nodo X, que estaba conectado al nodo B', está ahora conectado al nodo B. La búsqueda progresa, y la ruta de acceso inferior ( a través de C' y D') llega segundo en E. Una vez más, las rutas correspondientes C' y D' y el CD se comparan.

D) CD y C' D' se consideran similares por lo que se combinan.

- **Cuarta Fase:** Esta es la última fase del algoritmo. En esta fase el ensamblador se encarga de eliminar las repeticiones que haya en el grafo de Bruijn y se encarga de ensamblar los Scaffolds finales del ensamblaje. Dichos scaffolds se generan mediante los nodos únicos del grafo y sus vecinos. En primer lugar se genera un scaffold entre dos nodos cualquiera y a partir de ahí se va aumentando los Scaffolds añadiendo los vecinos de los nodos únicos que vayamos agregando.

El caso del ensamblador Velvet es un caso peculiar ya que está dividido en dos programas independientes que hacen tareas complementarias.

Estos programas son Velveth y Velvetg. A continuación vamos a describir con detalle cada uno de ellos.

- **Velveth:** Este subprograma de Velvet es el encargado de ayudar a formar la estructura de datos que se va a utilizar para ensamblar todo el conjunto del genoma, función que realizará el programa Velvetg.
- **Velvetg:** 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.

### 2.7.6 Ensamblador Ray

El ensamblador Ray, es un ensamblador que utiliza el Grafo de Bruijn para construir su estructura de datos.

La principal característica que podemos de estacar de este ensamblador es que es un ensamblador que utiliza memoria distribuida para realizar sus operaciones. En los dos casos anteriores, el ensamblador Minia y el ensamblador Velvet, utilizaban memoria compartida. Por lo tanto el ensamblador Ray utiliza MPI como interfaz de paso de mensajes entre todos los nodos que estén funcionando de manera simultánea. Gracias a esta característica podemos aprovechar aún más el potencial del supercomputador Magerit, y no tendremos que limitarnos a las 16 CPUs que contiene como máximo cada uno de los nodos de tecnología Intel del supercomputador.

El funcionamiento de este ensamblador es análogo al utilizado en el ensamblador Velvet. En primer lugar, a partir de una serie de librerías con lecturas procedemos a construir el Grafo de Bruijn.

Una vez construido el Grafo de Bruijn pasamos a una fase de simplificación del mismo para intentar optimizar el tiempo de ejecución y el consumo total de recursos. La simplificación trata de comprobar que cada una de las lecturas, con un valor de k determinado, solo aparece una vez en el Grafo, así evitamos repeticiones innecesarias, consumo de memoria y tiempo de ejecución al intentar ensamblar dichas lecturas.

A diferencia de otros ensambladores que utilizan el Grafo de Bruijn para construir su estructura de datos, Ray no implementa un mecanismo de corrección de errores. Por esta razón el usuario tiene que tener cuidado con los valores de k que introduce en el sistema. Ya que valores muy grandes o inadecuados generarán múltiples errores que no serán corregidos con lo que se verá afectado el resultado final del proceso de ensamblaje.

Durante la siguiente etapa, el algoritmo se encarga de definir sub secuencias que son llamadas semillas. Estas semillas son caminos a lo largo del grafo que se consideran, casi de manera inequívoca, como sub secuencias del resultado final del proceso de ensamblaje del genoma. Gracias a la determinación de estas semillas y su posterior extensión a lo largo del Grafo podemos generar los primeros contigs del proceso. Los contigs se van formando mientras haya posibilidades e

extensión, si se llega a un punto en el que estas posibilidades no consideran claras el algoritmo se detiene.

El proceso anterior se vuelve a producir, pero esta vez sobre los contigs, en vez de sobre las lecturas, gracias a esto podremos construir los Scaffolds asociados a un conjunto de contigs.

### 2.7.7 QUAST-(Quality Assessment Tool for Genome Assemblies)

QUAST (Quality Assessment Tool for Genome Assemblies), es una aplicación que nos permite evaluar la calidad de los ensambladores obteniendo distintos tipos de métricas.

Las métricas analizadas por QUAST son las siguientes:

- **#Contigs:** Número total de contigs en el ensamblado.
- **Contig más largo:** Longitud del contig más largo del ensamblado.
- **Longitud total:** Número total de bases en el ensamblado.
- **Longitud de Referencia:** Número total de bases en la referencia.
- **N50:** El parámetro N50 es una medida de la longitud media de un conjunto de secuencias y se define como el valor X, tal que, al menos la mitad del ensamblado está contenido en contigs de tamaño menor o igual X. ( Ver Capítulo 2.4.2).
- **NG50:** El parámetro NG50 es una medida análoga a la anterior pero en vez de considerar la mitad del conjunto ensamblado, consideramos le mitad del genoma que estamos ensamblando. ( Ver Capítulo 2.4.2).
- **N75 y NG75:** Análogos a los dos anteriores pero con un porcentaje del 75% en vez del 50%.
- **Número de contigs sin alinear:** Es el número de contigs que no están alineados.
- **Fracción del genoma:** Es el número total de bases alineadas en la referencia, dividido por el tamaño del genoma.
- **Ratio de duplicación:** Es el número de bases alineadas en el conjunto, dividido por el número total de bases alineadas en la referencia. Si el conjunto contiene muchos contigs que cubren las mismas regiones de la referencia, su ratio de duplicación puede ser mayor que 1.
- **Número de genes:** Número total de genes en el conjunto ensamblado, sobre la base de una lista proporcionada por el usuario de las posiciones de los genes en el genoma de referencia.
- **Alineamiento más largo:** Es la longitud de la alineación más grande en el conjunto.

- **Parámetros de Entrada.**
  - **-o <fichero de salida>:** Con esta opción especificamos el directorio de salida para almacenar los resultados analizados.
  - **-R <nombre del fichero>:** Genoma de referencia. Nos permite analizar más parámetros de los resultados pudiendo comparar los resultados con el genoma de referencia.
  - **-G o - -genes <nombre del fichero>:** Archivo de genes anotados.
  - **-O o – operons <nombre del fichero>:** Archivo de operaciones
  - **--min-contig<int>:** Número máximo de hilos que puede utilizar QUAST. Por defecto coge el número de CPUS que podemos disponer.

Existen muchas más opciones de entrada, pero podríamos definir las anteriores como las más características.

- **Salida:**
    - **report.txt:** Resumen de evaluación en formato de texto simple.
    - **report.tsv:** Versión del resumen separada por tabuladores. Ideal para hojas de cálculo.
    - **report.tex:** Versión del resumen en formato LaTeX
    - **plots.pdf:** Fichero con tablas del resumen.
    - **report.html:** Versión html del resumen.
    - **contigs_reports:**
        - **misassemblies_report:** Reporte de contigs mal ensamblados.
        - **unaligned_report:** Reporte de contigs sin alinear

# 3. EVALUACIÓN DE RIESGOS

Cómo hemos podido ver a lo largo de toda la descripción de la introducción y del estado del arte del proyecto, el campo en el que nos hemos centrado en el proyecto es un campo de gran amplitud y en el cual podemos considerar y utilizar numerosas tecnologías, ya sean antiguas o más nuevas para resolver el problema que abordamos en el presente proyecto: Ensamblar genes.

La evaluación de riesgos se ha realizado observando detenidamente todos los tipos de tecnologías, y de opciones, que tenemos a nuestro alcance para realizar el proyecto. En nuestro caso en concreto disponemos de un supercomputador de gran capacidad (Magerit 2) y las tecnologías más avanzadas que podemos encontrar en el mercado. Gracias a estas características podríamos decantarnos por utilizar los ensambladores más novedosos.

No debemos olvidar que el ensamblaje de genes es un proceso costoso y largo por lo que tenemos que asegurarnos bien de elegir la mejor solución posible para intentar reducir el consumo de tiempo y de recursos, y por lo tanto de dinero.

Pese a los numerosos tipos de ensambladores que hay en el mercado, hemos intentado simplificar esa lista para comprobar los ensambladores, que a priori, nos van a ofrecer mejores resultados en los análisis que vamos a realizar.

Para evitar problemas de calidad del ensamblaje hemos decidido utilizar los ensambladores que utilizan la tecnología de los Grafos de Bruijn para generar su estructura de datos (Ver Sección 2.3.2).

La razón, quizás más importante, para elegir este tipo de ensambladores ha sido limitar, en lo más posible, la aparición de riesgos y problemas innecesarios. Este tipo de ensambladores son los ensambladores que más se están utilizando y probando hoy en día, por lo que podremos encontrar mucha más documentación y experimentos de apoyo para conseguir completar con éxito los experimentos que se plantean en el presente proyecto.

Otra de las razones para elegir este tipo de ensambladores ha sido la necesidad de probar la veracidad de los excelentes resultados, que a priori, presenta el ensamblador Minia (Ver Sección 2.7.4). Para realizar esta comprobación tenemos que utilizar ensambladores de similares características, aunque difieran en la gestión de memoria interna (Ver Sección 2.4.4).

Dentro de esta gran selección también tendríamos que seleccionar un ensamblador por tipo de memoria interna. Gracias a que para nuestro problema disponemos de unos de los supercomputadores más potentes de España la selección de ensamblador de tipo de memoria no es tan crítica, ya que la tecnología no nos limita esta elección.

Pese a que la tecnología no nos limita esta última elección, también es necesario destacar que tenemos que centrarnos, y como se ha comentado anteriormente, en encontrar el ensamblador que consuma menos tiempo, recursos y por tanto dinero.

# 4. DESARROLLO

## 4.1 Introducción

El principal objetivo del proyecto es comprobar el funcionamiento de dos ensambladores de distinto tipo, analizando sus diferentes resultados y comentándolos.

El proceso consistirá en ejecutar los dos ensambladores seleccionados con 4 tipos de secuenciación de diversos organismos vivos. Dichos organismos son dos bacterias (Sthapylococcus Aureus, *Rhodobacter Sphaeroides*) y un mamífero (*Homo Sapiens Sapiens*)

Las pruebas con dichos algoritmos consistirán en la ejecución sistemática de cada uno de los ensambladores variando diversos parámetros de ejecución como son los valores de los k-mers de cada ensamblador y el número de CPUs utilizadas.

Dentro de los posibles valores de k probaremos los valores que se encuentren en el intervalo de 25 a 45 con una variación de 10 en 10, y el valor k=27 que según se puede comprobar en la documentación del ensamblador Minia (Ensamblador Minia) es el mejor valor para ensamblar el genoma humano.

El número de CPUs utilizadas variarán entre 1 CPUs y 16 CPUs. Utilizando concretamente 1,2, 4, 8 y 16 CPUs para las pruebas en el caso de memoria compartida (salvo en el caso del ensamblador minia que lo vamos a probar solo con 1 CPU ya que es el único ensamblador que utiliza un "single thread"). En el caso de memoria distribuida el número de CPUs variará entre 1CPUs y 128 CPUs, utilizando concretamente 1, 2, 4, 8, 16, 32,64 y 128 CPUs

Con estas variaciones se busca comprobar la configuración más óptima para encontrar los resultados más óptimos en el menor tiempo posible y con el menor consumo de recursos.

Una vez ejecutados todas las combinaciones posibles de parámetros extraeremos el consumo de memoria, tiempo de ejecución y los valores N50, el número de contigs, número de scaffolds.

Con todos esos parámetros podremos compararlos entre sí para especificar que ensamblador es el más óptimo para conseguir un ensamblaje de genomas de buena calidad.

### 4.1.1 Datasets de Genomas Utilizados

Par la realización de este proyecto hemos tenido que utilizar diversos recursos proporcionados por terceros.

Los recursos, que principalmente hemos tenido que utilizar, son un conjunto de lecturas, tanto cortas, como largas de genomas de diversos seres vivos.

Dichos datasets los hemos recopilado de dos fuentes distintas de información.

La primera fuente de información ha sido la fundación GAGE (Genome Assembly Gold-Standard Evaluations). Que en su página oficial (Gage) Nos proporciona diversos datasets, con lecturas

cortas y largas de seres vivos como las bacterias *Staphylococcus Aureus* y *Rhodobacter Sphaeroides*, y el cromosoma 14 del genoma Humano. En la página de GAGE podemos encontrar las siguientes características para los genomas utilizados.

- ***Staphylococcus Aureus*:**

| Tipo de DataSet: | Pair-end Fragment | Short Jump |
|---|---|---|
| Tamaño del Genoma: | 2903081 | 2903081 |
| Promedio de Lecturas: | 101bp | 37bp |
| Lecturas Insertadas: | 180bp | 3500bp |
| Número de Lecturas | 1.294.104 | 3.494.070 |

Tabla 5 Características Librerías *Staphylococcus Aureus*

- ***Rhodobacter Sphaeroides*:**

| Tipo de DataSet: | Pair-end Fragment | Short Jump |
|---|---|---|
| Tamaño del Genoma: | 4.603.060 | 4.603.060 |
| Promedio de Lecturas: | 101bp | 101bp |
| Lecturas Insertadas: | 180bp | 3500bp |
| Número de Lecturas | 2.050.868 | 2.050.868 |

Tabla 6 Características Librerías *Rhodobacter Sphaeroides*

- ***Homo Sapiens Sapiens* Cromosoma 14**

| Tipo de DataSet: | Pair-end Fragment | Short Jump | Long Jump |
|---|---|---|---|
| Tamaño del Genoma: | 88.289.540 | 88.289.540 | 88.289.540 |
| Promedio de Lecturas: | 101bp | 101bp | 76-101bp |
| Lecturas Insertadas: | 155bp | 2283-2803bp | 35,295-35,318bp |
| Número de Lecturas | 36.504.800 | 22.669.408 | 2.405.064 |

Tabla 7Características Librerías *Homo Sapiens Sapiens* Cromosoma 14

## 4.2 Ensamblador Minia

### 4.2.1 Descarga e Instalación

En primer lugar tenemos que descargar la última versión del código fuente de Minia de su página oficial. (Ensamblador Minia)

En dicha página oficial encontraremos el enlace de descarga, en forma de hipervínculo. Cómo hemos visto en secciones anteriores nos dispondremos a ejecutar el comando wget para descargar la información. Para ello tendremos que encontrar la url del archivo a descargar. Con esa dirección ejecutamos el comando wget:

- wget -b http://minia.genouest.org/files/minia-1.4961.tar.gz.

Tras unos minutos de espera tendremos el archivo descargado en nuestro directorio en formato .tar.gz. Dentro de este archivo podremos encontrar el código fuente del ensamblador, que tendremos que compilar una vez descomprimido para adaptarlo a la arquitectura del supercomputador.

Para descomprimir archivos .tar.gz utilizaremos el siguiente comando:

- tar –xvf archivo.tar.gz

Una vez que hemos descomprimido el código fuente tenemos que entrar en la carpeta donde lo hemos instalado y ejecutar el archivo makefile para proceder a compilar el código. Para entrar en la carpeta utilizaremos el siguiente comando:

- cd nombreCarpeta

Para ejecutar el archivo makefile utilizaremos el comando “make”.

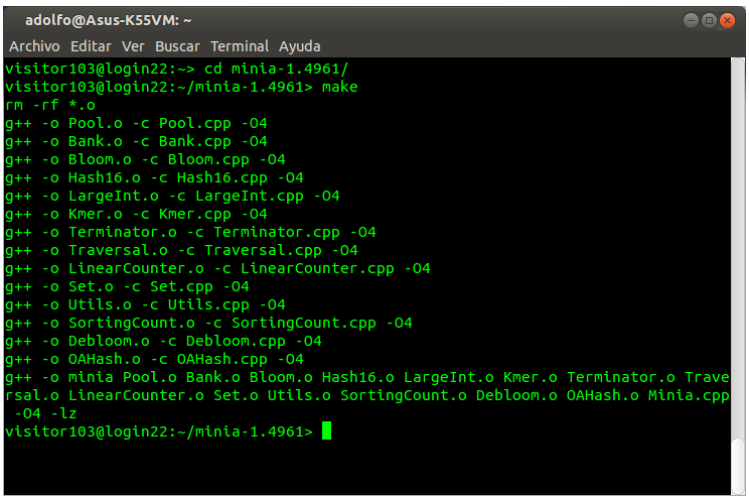



Ilustración 33 Compilación Ensamblador Minia

La configuración anterior es la configuración por defecto de Minia. Si quisiéramos compilar minia para valores grandes de K tendríamos que volver a ejecutar el comando make. Para ello, en primer lugar, tendríamos que borrar todos los ficheros binarios generados anteriormente y a continuación ejecutar el comando make con el parámetro k:

- make clean && make k=100

Una vez realizada la compilación del código fuente, ya sea para valores normales de k o para grandes valores de k, ya estaríamos listos para ejecutar el ensamblador en el supercomputador. Antes de mandar una tarea al supercomputador deberíamos cerciorarnos de que el funcionamiento es el correcto. Para ello, en la carpeta donde hemos descomprimido minia, disponemos de un script .sh que nos indicará si todo ha funcionado correctamente y estamos listos para ejecutar el ensamblador.

Para ejecutar un script sh utilizaremos el siguiente comando:

- ./nombreScript

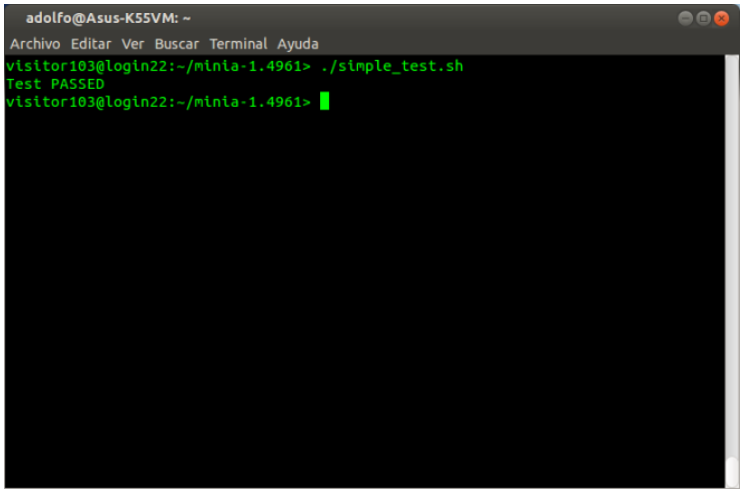



Ilustración 34 Test Funcionamiento Minia

Como podemos ver en la imagen anterior, todo ha funcionado correctamente por lo que podemos confirmar que el ensamblador ha sido instalado correctamente y estamos listos para su ejecución contra el supercomputador.

Puede darse el caso de que para ejecutar el Script anterior tengamos que dar permisos de ejecución al fichero correspondiente. Para realizar esta tarea utilizaremos el comando chmod.

### 4.2.2 Preparación de las Lecturas

Antes de realizar la ejecución de Minia con todas las lecturas que hemos descargado para comprobar el funcionamiento de los ensambladores.

Tenemos que descomprimir la información para que sea accesible por los ensambladores. En el caso del ensamblador minia no tenemos que realizar ninguna tarea de conversión de datos ni nada parecido para que sea totalmente funcional. La única tarea que tenemos que hacer es crear un fichero de texto plano con las rutas de todos los ficheros de las lecturas. Cómo hemos visto en la Sección 2.7.4 podemos agregar varios ficheros de lecturas por medio de la concatenación de las rutas de todos ellos en un fichero de texto que pasaremos como parámetro al ensamblador. Esta tarea nos permite simplificar mucho el comando de ejecución de este ensamblador.

### 4.2.3 Ejecución

Como hemos visto en la Sección 2.7.3. Para enviar trabajos a Magerit 2 tenemos que configurar un fichero de tipo script con todos los parámetros que consideremos oportunos para realizar la ejecución de las tareas.

En la siguiente imagen podemos observar la configuración por defecto del Script que corresponde a las lecturas de la bacteria *Rhodobacter Sphaeroides* con 1 CPU y un valor de k = 25.

Para el resto de lecturas y configuraciones sería un script análogo pero cambiando los parámetros necesarios por los adecuados.

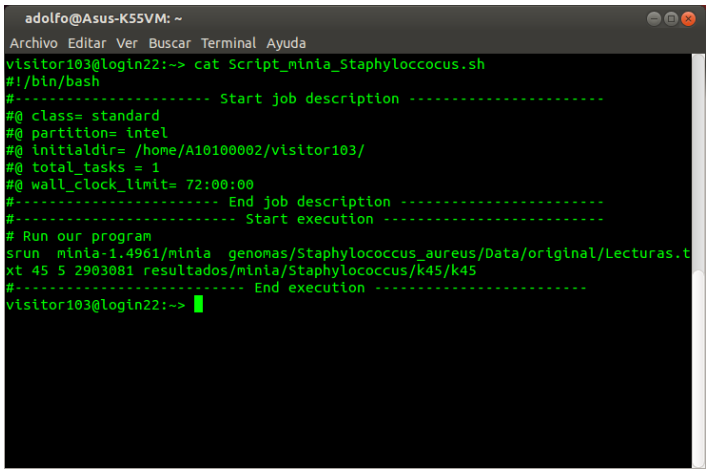



Ilustración 35 Ejemplo de Script de Ejecución de Tareas Minia

Como podemos observar en la imagen, la configuración del Script es bastante sencilla. En el indicamos que el número de CPUs que necesitamos, en este caso 1, y el tiempo límite de ejecución. Este tiempo se debería ajustar para intentar adaptarlo al tiempo real de ejecución que tendrá nuestro programa.

Una vez que hemos configurado todos estos parámetros tenemos que definir el comando de ejecución de Minia con los parámetros que hemos visto en la Sección 2.7.4.

- **Primer parámetro:** Ruta donde se encuentra el instalador de Minia.
- **Segundo parámetro:** Valor que asignamos a K. En este caso K =25.
- **Tercer parámetro:** Parámetro de mínima abundancia. Este parámetro no interfiere en el proceso por lo que no es muy importante.
- **Cuarto parámetro:** Tamaño esperado del genoma resultante.
- **Quinto parámetro:** Ruta de salida para los ficheros generados.

Finalmente, una vez que hemos generado el Script para cada configuración solo tenemos que enviárselo al gestor de coles de Magerit con el comando sbatch.

### 4.2.4 Extraer Resultados

Como punto negativo del ensamblador Minia, tenemos que indicar que en este apartado tiene bastantes carencias. En primer lugar no dispone de ficheros de log, o de resultados por lo que tenemos que recurrir a leer los ficheros de lectura de ejecución de trabajo del gestor de cola

SLURM. Cómo indicamos en la Sección 2.7.3. Toda ejecución de tareas en Magerit en un fichero del siguiente formato: slurm-"identificador".out. Siendo el identificador el número de trabajo.

En dicho fichero podemos encontrar la información del tiempo de ejecución del ensamblador y del consumo de memoria RAM del mismo. Dicha información se encuentra al final del documento. Por lo que leemos el fichero utilizando el siguiente comando:

- tail –n "Número de Líneas" "Nombre del Fichero"

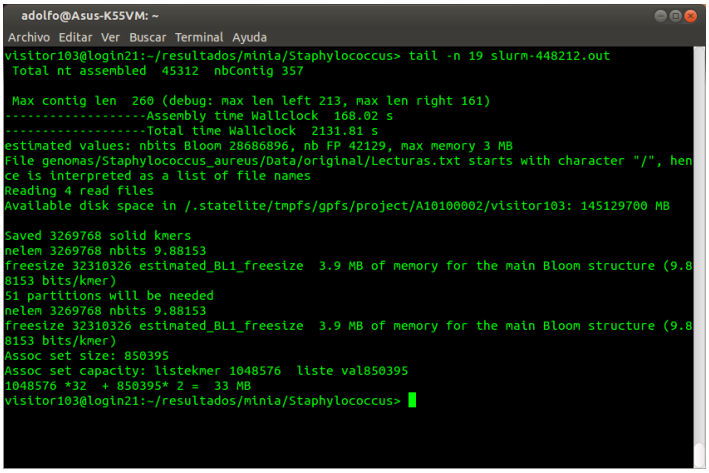



**Tabla 8 Recopilación de Datos Ensamblador Minia**

En el fichero tenemos varios parámetros de importancia:

- **Assembly Time Wallclock:** Tiempo que ha tardado el ensamblador minia en realizar la tarea de ensamblar el genoma especificado (solamente).
- **Total Time Wallclock:** Tiempo que ha tardado el ensamblador en realizar todas las tareas. Creación del Grafo de Bruijn, creación del filtro de Bloom etc.
- **Total:** Indica el consumo de memoria RAM en bits por parte de la estructura de datos.

Para extraer las estadísticas concretas que ha producido el proceso de ensamblaje tenemos que utilizar la herramienta descrita en la Sección 2.7.7

Para instalar la herramienta utilizamos los siguientes comandos:

- wget https://downloads.sourceforge.net/project/quast/quast-2.1.tar.gz
- tar -xzf quast-2.1.tar.gz
- cd quast-2.1

También tenemos la opción de instalar la librearía, si no está instalada, Matplotlib. Para generar gráficas que se guardarían en ficheros LaTeX o pdf. En nuestro caso no es necesario ya que no queremos valores tan detallados.

Para ejecutar la herramienta utilizamos el siguiente comando:

- python quast-2.1/quast.py --min-contig 10 -R genomas/Staphylococcus_aureus/Data/original/genome.fasta -o resultados/minia/Staphylococcus/k15/ resultados/minia/Staphylococcus/k15/k15.contigs.fa

Donde tenemos los siguientes parámetros:

- **--min-contig:** Longitud mínima que aceptamos como contig.
- **-R:** Genoma de referencia. En nuestro caso es indiferente proporcionarlo o no ya que solo vamos a calcular los valores N50, N75, número de contigs… y para calcular estos valores no necesitamos el genoma de referencia.
- **-o:** Ruta donde queremos guardar todas las estadísticas proporcionadas por la herramienta
- Ruta donde se encuentra el fichero con los contigs de salida del ensamblador Minia.

## 4.3 Ensamblador Velvet

### 4.3.1 Descarga e Instalación

El proceso de instalación del ensamblador Velvet es análogo al proceso de instalación del ensamblador Minia.

En primer lugar tenemos que descargarnos el código fuente del ensamblador de su página oficial (Ensamblador Velvet). Con el siguiente comando:

- wget –b http://www.ebi.ac.uk/~zerbino/velvet/velvet_1.2.09.tgz

Una vez descargado el código fuente, tenemos que descomprimir el paquete e introducirnos en la carpeta que se acaba de crear. Para ello, y como siempre, utilizamos los siguientes comandos:

- tar –xvf velvet_1.2.09.tgz
- cd velvet_1.2.09.

Ahora que estamos en la carpeta del ensamblador podemos comenzar a realizar el proceso de instalación. Para realizar la instalación tenemos que ejecutar el siguiente comando:

- make “OPENMP=1” “CATEGORIES=4” “MAXKMERLENGTH=65”

Donde tenemos tres parámetros importantes, la descripción de estos parámetros es la siguiente:

- **OPENMP:** Variable que si le asignamos valor 1 activamos la ejecución multi-hilo. Cómo ya hemos comentado durante el desarrollo de esta memoria, Velvet es un ensamblador de memoria compartida, por lo que podremos utilizar tantas CPUs como CPUs tenga la máquina donde se ejecutará. En nuestro caso el mayor número de CPUs que podemos tener es 16.
- **CATEGORIES:** Variable que permite a Velvet distinguir diferentes bibliotecas de inserción. En nuestro caso en principio como máximo utilizaremos 4 categorías. Pero este parámetro lo podremos cambiar siempre que queramos recompilando el código fuente.
- **MAXKMERLENGTH:** Máximo valor de K que podremos asignar para realizar el ensamblado de genes. En nuestro caso nuestros valores de K siempre van de 15 a 45, pero le

asignamos un valor un poco superior. Cómo en el caso anterior, se puede reconfigurar recompilando.

El ensamblador Velvet tiene muchos más parámetros de compilación y de ejecución. En este proyecto no es necesario explicarlos, pero se pueden consultar visitando en el manual del ensamblador (Manual Ensamblador Velvet).

### 4.3.2 Ejecución

La ejecución del ensamblador Velvet sigue los mismos principios que la ejecución del ensamblador Minia. Tenemos que construirnos nuestro fichero .sh para mandar la tarea al gestor de colas SLURM.

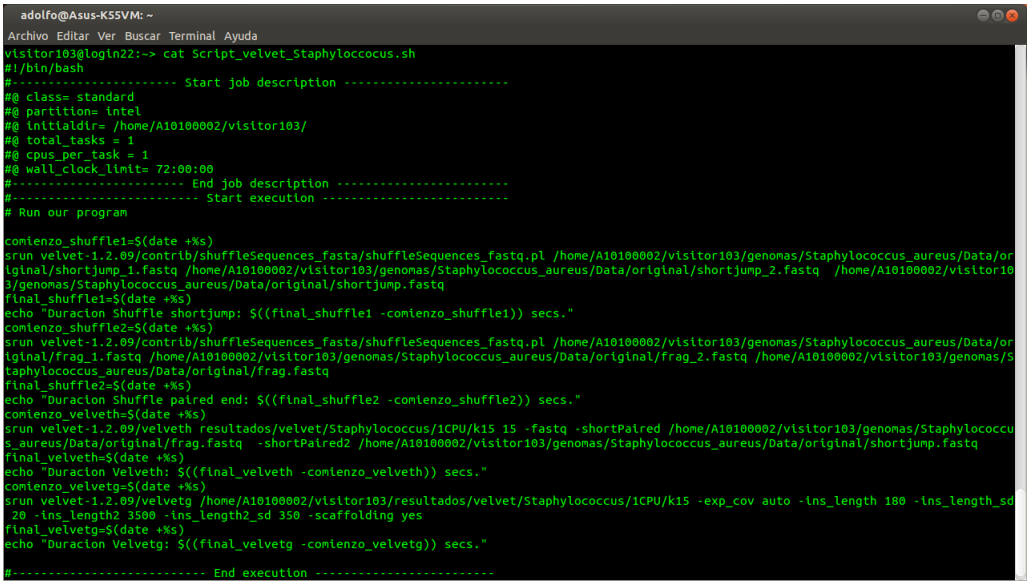



Ilustración 36 Ejemplo Script de Ejecución Ensamblador Velvet

Cabe destacar que para el ensamblador Velvet no vale con ejecutar un solo comando que englobe todas las tareas. En realidad tenemos que ejecutar 3 comandos distintos:

- **shuffleSequences_fastq.pl:** Este programa (que podemos encontrar en la carpeta de Velvet) sirve para "barajar" los pares de lecturas que tenemos en cada librería ya que Velvet necesita tener un fichero de unión para librearías pareadas. Este comando hay que ejecutarlo para cada par de librerías (en el caso del ejemplo 2).

- **velveth:** Programa de Velvet que nos proporciona el Grafo de Bruijn que posteriormente utilizará velvetg para realizar el proceso de ensamblaje. En este programa tenemos que indicar la ruta de salida, el valor de K, el formato de los archivos y el tipo de lecturas (en este caso short-paired).

- **velvetg:** Programa que ejecuta el proceso de ensamblaje. Tenemos que pasarle como parámetro la ruta de salida (la misma que en velveth) y otros parámetros auxiliares. (Manual Ensamblador Velvet)

### 4.3.3 Extraer Resultados

La extracción de resultados del ensamblador Velvet es una tarea compleja ya que tenemos que realizar bastantes tareas para conseguir recuperar el tiempo de ejecución, el consumo de Memoria RAM y las estadísticas de las lecturas.

- **Tiempo:** El ensamblador Velvet no proporciona ninguna interfaz para poder medir el tiempo de los procesos que lo acompañan. Para ello tenemos que introducir en el Script de ejecución medidores de tiempo antes y después de cada comando de ejecución para conseguir calcular el tiempo de ejecución a mano.

- **Consumo de Memoria RAM.** El ensamblador Velvet tampoco proporciona una interfaz para conseguir medir el consumo de memoria RAM. Para ello tendremos que utilizar una expresión teórica (Consumo de Memoria Velvet):
  - Consumo de Memoria = -109635 + 18977*Tamaño de las lecturas + 86326*Tamaño del Genoma + 233353*Número de Lecturas(en millones)- 51092*K

  El resultado viene indicado en KB. Puede darse el caso, que para algunos valores de K, el resultado sea negativo, si esperamos que se produzca un proceso de ensamblaje correcto aplicamos valor absoluto a la fórmula anterior.

- **Estadísticas de los contigs conseguidos:** El ensamblador Velvet proporciona una interfaz de resultados bastante limitada. En el fichero de salida de ejecución del ensamblador podemos observar un resumen de una línea donde nos indica los valores N50, número de contigs, longitud máxima de un contig y longitud total.
  Para calcular estos valores de otra manera y comprobar su veracidad vamos a utilizar la herramienta QUAST como pudimos ver en el caso del ensamblador Minia.
  - python quast-2.1/quast.py --min-contig 10 -R genomas/Staphylococcus_aureus/Data/original/genome.fasta -o resultados/velvet/Staphylococcus/k15/ resultados/velvet/Staphylococcus/k15/k15.contigs.fa

## 4.4 Ensamblador Ray

### 4.4.1 Descarga e Instalación

El proceso de descarga e instalación del ensamblador Ray es prácticamente idéntico al que hemos seguido en los dos ensambladores anteriores. En primer lugar nos tenemos que descargar el código fuente de la página oficial del ensamblador (Ensamblador Ray). Para descargarlo en el supercomputador Magerit utilizaremos el siguiente comando:

- wget –b http://sourceforge.net/projects/denovoassembler/files/Ray-v2.2.0.tar.bz2/download

Después de descargar el código fuente, tenemos que descomprimir el paquete descargado e introducirnos en la carpeta que se ha creado. Para ello ejecutamos los siguientes comandos:

- tar –xvf Ray-v2.2.0.tar.bz2
- cd Ray-v2.2.0

Una vez dentro tenemos que prepararnos para compilar el código fuente. Para ello utilizaremos el comando make de la siguiente manera:

- make PREFIX=build MAXKMERLENGTH=65

Las opciones que acompañan al make tienen la siguiente funcionalidad:

- **PREFIX:** Este parámetro indica al fichero MakeFile que está en proceso de construcción y de compilación del código fuente.
- **MAXKMERLENGTH**: Parámetro que nos permite indicar el valor máximo de k. En nuestros experimentos ese valores es k=45 pero le indicamos 20 unidades más por si queremos realizar alguna prueba específica. Si queremos aumentar este parámetro hay que recompilar el código fuente.

### 4.4.2 Ejecución

Para ejecutar Ray tenemos que realizar un script .sh para enviárselo al gestor de colas SLURM.

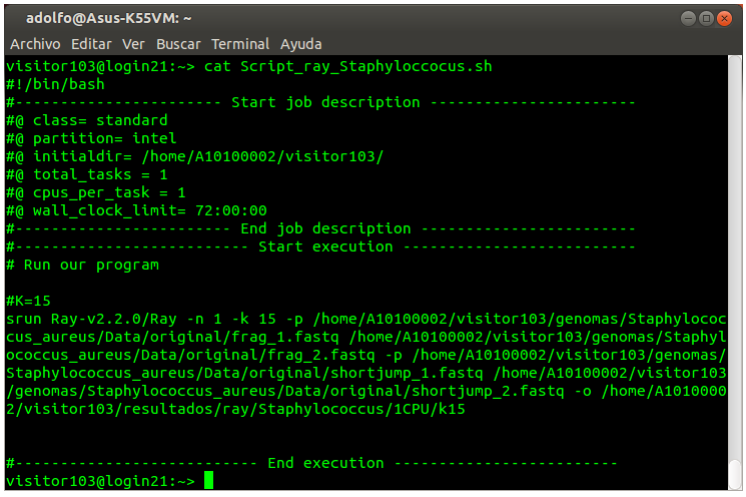



Tabla 9 Script de Ejecución Ensamblador Ray

En este script podemos distinguir los siguientes parámetros:

- **-n:** Número de CPUs que queremos utilizar para ejecutar el ensamblador Ray. Tiene que coincidir con el valor total_tasks de la descripción del script.
- **-k:** Valor que asignamos a K para realizar el proceso de ensamblaje.
- **-p:** Con este parámetro indicamos que vamos a insertar un par de lecturas (da igual del tipo que sean).
- **-o:** Ruta donde guardamos los resultados.

### 4.4.3 Extraer Resultados

Para extraer los resultados generados por el ensamblador Ray tenemos que dirigirnos a la ruta que hemos indicado para guardar todos los datos de salida. En esa ruta podremos encontrar lo siguiente:

- **Contigs.fasta:** Contiene las secuencias contiguas en formato FASTA.
- **ContigLengths.txt:** Contiene las longitudes de las secuencias contiguas.
- **Scaffolds.fasta:** Contiene el scaffolds de secuencias en formato FASTA.
- **ScaffoldsLengths.txt:** Contiene las longitudes de los scaffolds.
- **ScaffoldsComponents.txt:** Contiene los componentes de cada scaffold.
- **ScaffoldsLinks.txt:** Contiene los enlaces de los scaffolds
- **OutputNumbers.txt:** Resumen de todas las estadísticas, valores N50…

En la ruta de salida de generan muchos más ficheros, pero hemos indicado los más importantes para el desarrollo del proyecto. El resto se pueden consultar en el manual de Ray. (Manual Ensamblador Ray)

# 5. RESULTADOS

Para analizar los resultados alcanzados con todas las ejecuciones con distintos parámetros de este ensamblador, vamos a distribuir la información por cada uno de los organismos vivos con los que hemos trabajado en estos experimentos.

Dentro de cada sección vamos a representar la información por medio de tablas y gráficas ilustrativas. Los valores que vamos a analizar en esta sección son los siguientes: Tiempo de ejecución, Consumo de Memoria RAM, Valores N50.

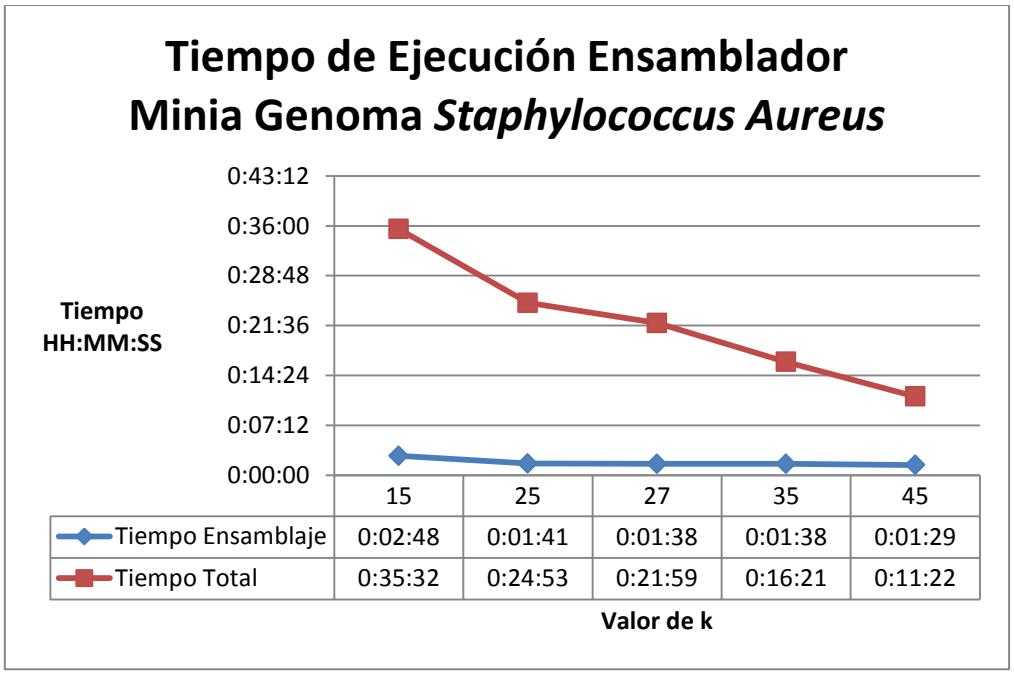
Dichos valores serán representados para cada tipo de configuración que hemos utilizados en las pruebas. Número de CPUs y valores de K.

## 5.1 Resultados Ensamblador Minia

En este apartado se van a representar las tablas y gráficas más relevantes de las pruebas realizadas con el ensamblador Minia. En algunas de ellas la escala de la gráfica se ha cambiado por una escala logarítmica de base 5 para mejorar la percepción de los datos.

### 5.1.1 Sthapylococcus Aureus

- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Tiempo Ensamblaje | 0:02:48 | 0:01:41 | 0:01:38 | 0:01:38 | 0:01:29 |
| Tiempo Total | 0:35:32 | 0:24:53 | 0:21:59 | 0:16:21 | 0:11:22 |

Tabla 10 Gráfico del Tiempo de Ejecución Ensamblador Minia para el genoma Sthapylococcus Aureus

Cómo podemos observar en la gráfica anterior de resumen. Observamos una disminución de tiempo considerable mientras aumentamos el valor de k. Se puede observar que el tiempo de ensamblaje

propiamente dicho se mantiene más o menos constante, pero lo que disminuye es el tiempo de tareas adicionales (montar el grafo de Bruijn etc.)

- **Consumo de Memoria RAM:**

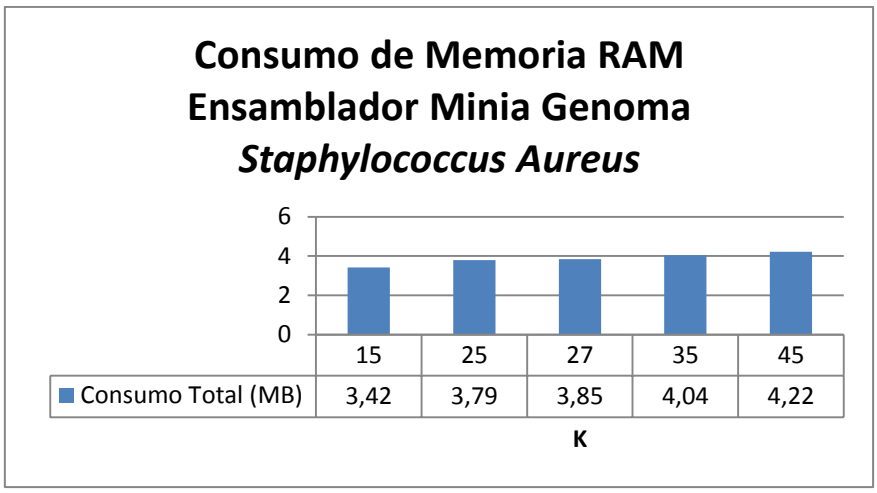



Tabla 11 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Staphylococcus Aureus*

En esta gráfica resumen podemos observar que el consumo de memoria RAM aumenta ligeramente cuando aumentamos el valor de k.

- **Estadísticas:**

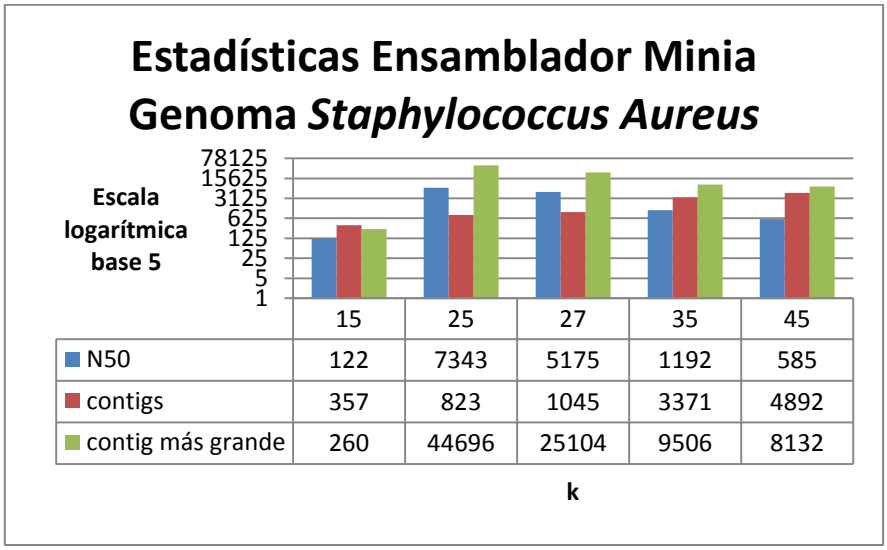



Tabla 12 Estadísticas de Ensamblaje para Ensamblador Minia Genoma *Staphylococcus Aureus*

Cómo podemos ver en la combinación de las gráficas anteriores. Los mejores resultados los obtenemos para un valor de k=25.

### 5.1.2 *Rhodobacter Sphaeroides*

- **Tiempo de Ejecución:**

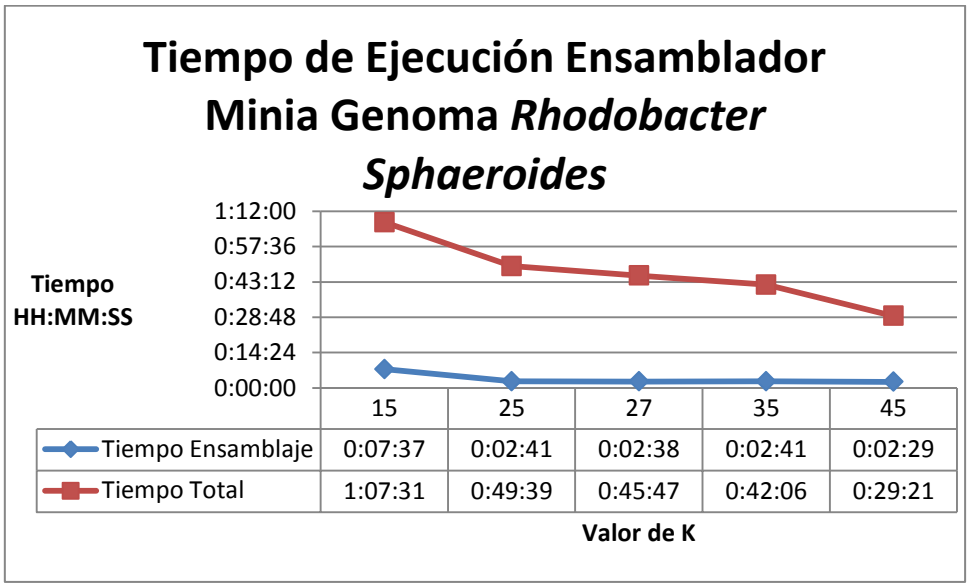



Tabla 13 Gráfico del tiempo de Ejecución Ensamblador Minia Genoma *Rhodobacter Sphaeroides*

En este ejemplo en particular pasa un caso análogo al del ejemplo del genoma *Staphylococcus Aureus*. Cada vez que aumentaos el valor de k disminuye el tiempo de ejecución, en mayor medida en tareas adicionales. En el tiempo de ensamblaje más o menos se mantiene constante, salvo para el valor de k=15 que es un valor mucho mayor.

- **Consumo de Memoria RAM:**

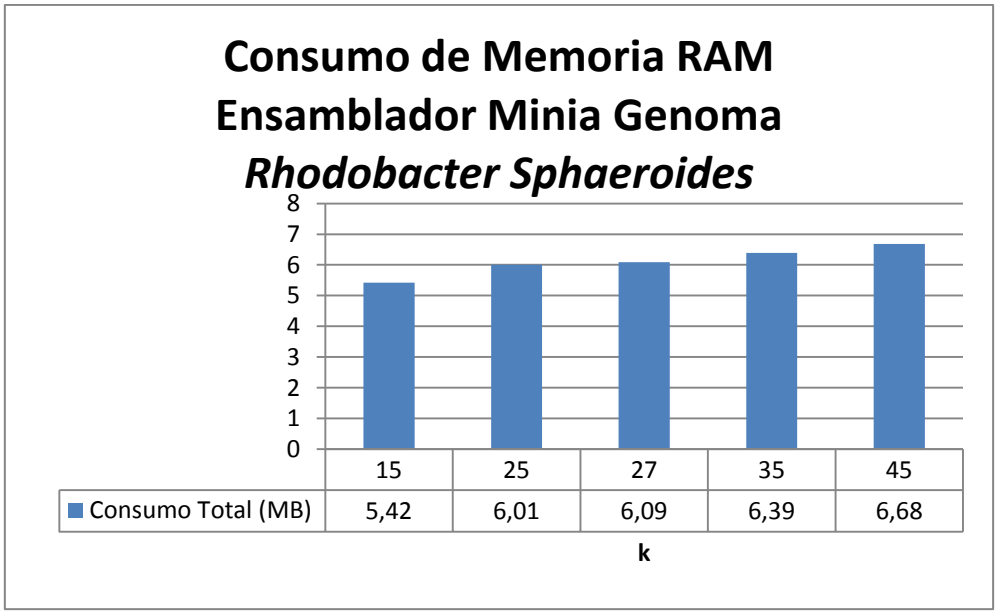



Tabla 14 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Rhodobacter Sphaeroides*

En este caso el consumo de memoria RAM aumenta también si aumentamos el valor de k, pero ese aumento es bastante progresivo, y entre los 5 valores de k seleccionadas solo se observa un aumento de menos de 1 Mb de memoria RAM.

- **Estadísticas:**

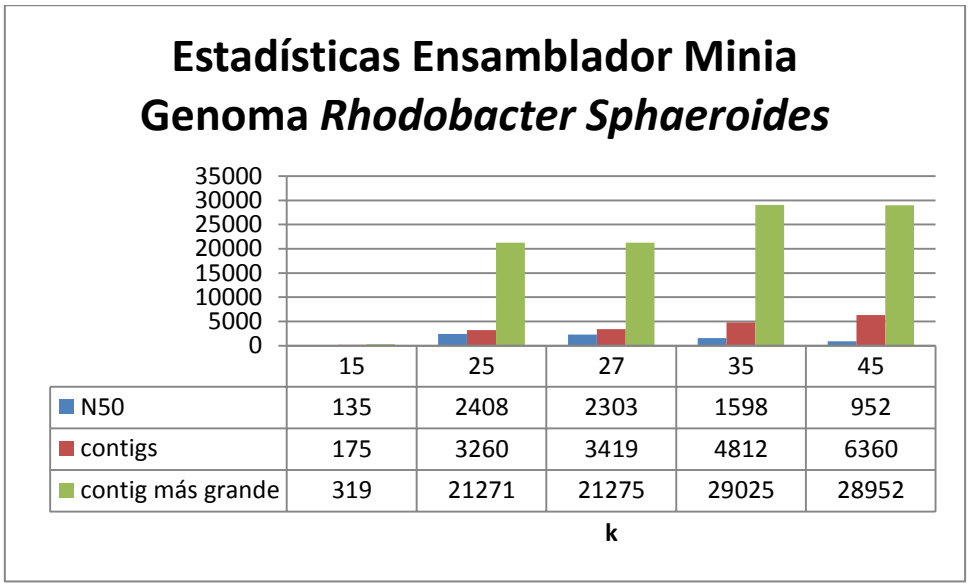



Tabla 15 Estadísticas Ensamblador Minia Genoma *Rhodobacter Sphaeroides*

En este caso observamos mejores valores para un valor de k= 25. Aunque bastante cercano al valor de k =27. Para el resto de valores vemos una disminución bastante relevante de los valores N50.

### 5.1.3 *Homo Sapiens Sapiens* Cromosoma 14

- **Tiempo de Ejecución:**

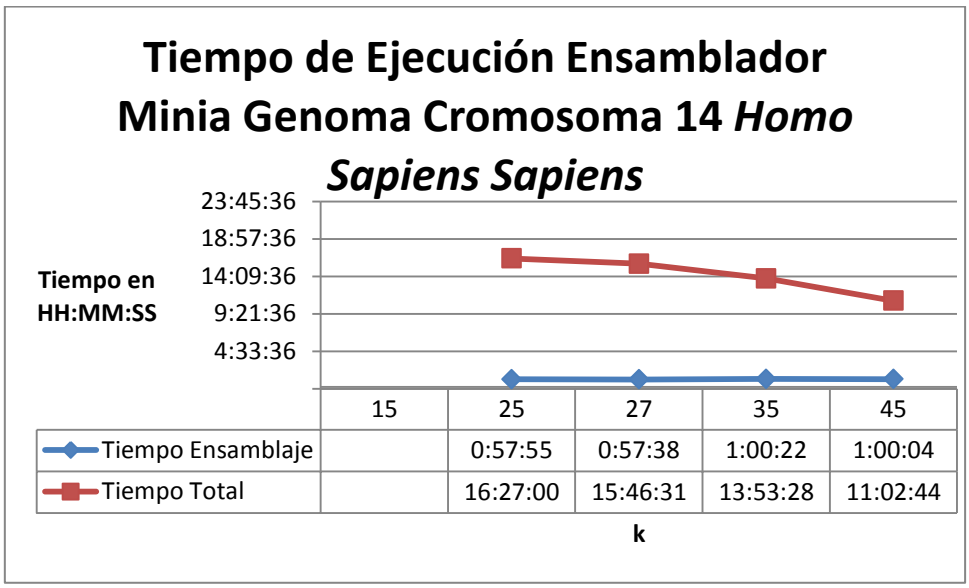



Tabla 16 Tiempo de Ejecución Ensamblador Minia Genoma *Homo Sapiens Sapiens* Cromosoma 14

Cómo hemos podido ver en los genomas anteriores el resultado es análogo. Cuando aumenta el valor de k disminuye el tiempo de ejecución, siendo más palpable en las tareas adicionales del proceso. En este caso incluso el tiempo de ensamblaje sufre un pequeño ascenso entre el valor de k 27 y valor de k 35.

- **Consumo de Memoria RAM:**

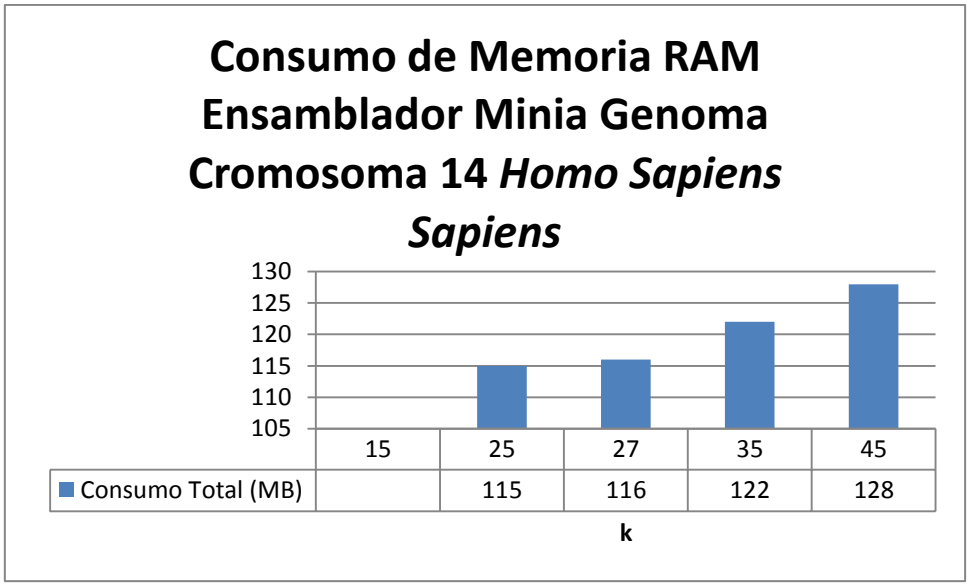



Tabla 17 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Minia para el genoma *Homo Sapiens Sapiens* Cromosoma 14

Cómo hemos visto en los ejemplos anteriores. Cada vez que aumentamos el valor de K aumenta el consumo de Memoria RAM.

- **Estadísticas:**

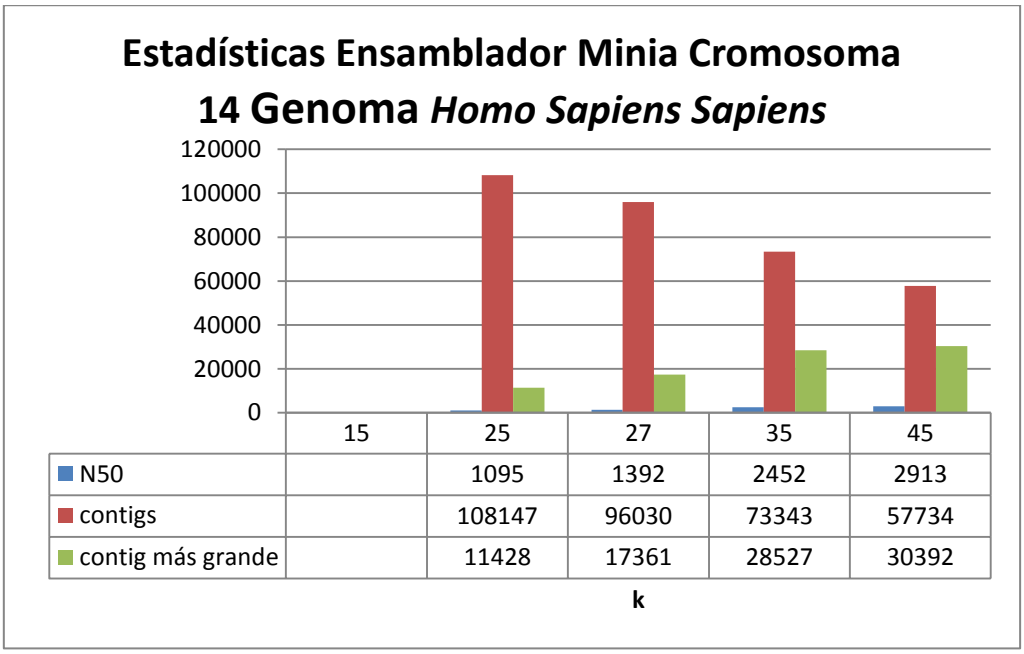


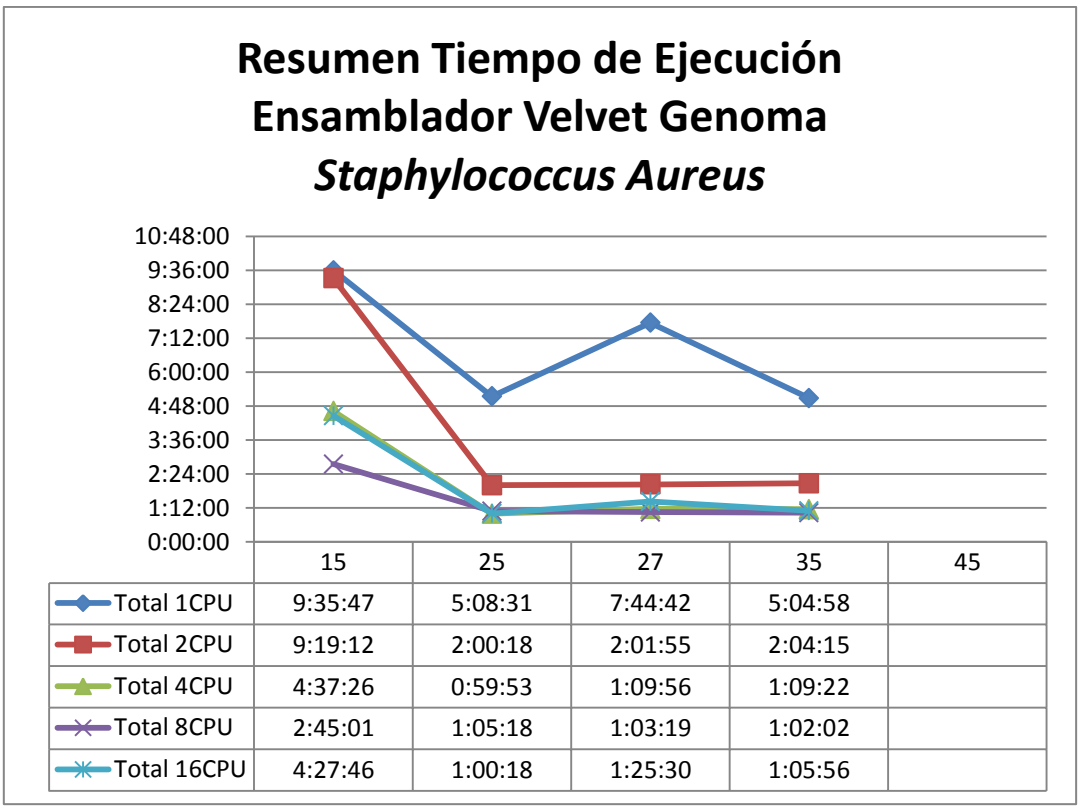

**Tabla 18 Estadísticas Ejecución Ensamblador Minia para el genoma *Homo Sapiens Sapiens* Cromosoma 14**

Cómo podemos observar en la gráfica anterior el mejor resultado de N50 es para el valor de k=45.

## 5.2 Resultados Velvet

### 5.2.1 Sthapylococcus Aureus

- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Total 1CPU | 9:35:47 | 5:08:31 | 7:44:42 | 5:04:58 | |
| Total 2CPU | 9:19:12 | 2:00:18 | 2:01:55 | 2:04:15 | |
| Total 4CPU | 4:37:26 | 0:59:53 | 1:09:56 | 1:09:22 | |
| Total 8CPU | 2:45:01 | 1:05:18 | 1:03:19 | 1:02:02 | |
| Total 16CPU | 4:27:46 | 1:00:18 | 1:25:30 | 1:05:56 | |

**Tabla 19 Tiempo de Ejecución Ensamblador Velvet Genoma *Staphylococcus Aureus***

Este ejemplo es un poco distinto al ensamblador anterior, en este caso tenemos varias ejecuciones con 5 números de CPUs distintos.

Cómo podemos observar cuando aumentamos el número de CPUs podemos observar una mejora notable en el rendimiento, pero hay un punto en que esa mejora se estabiliza. A partir de 4 CPUs la mejora es casi nula. El incremento más visible es en el paso de 1CPU a 2CPU. Excepto para el valor de k=15 que se mantiene estable y solo mejora a partir de 4CPU.

También podemos observar que el ensamblador Velvet para este genoma y valor de k = 45 no ofrece ningún resultado.

- **Consumo (teórico) de Memoria RAM:**

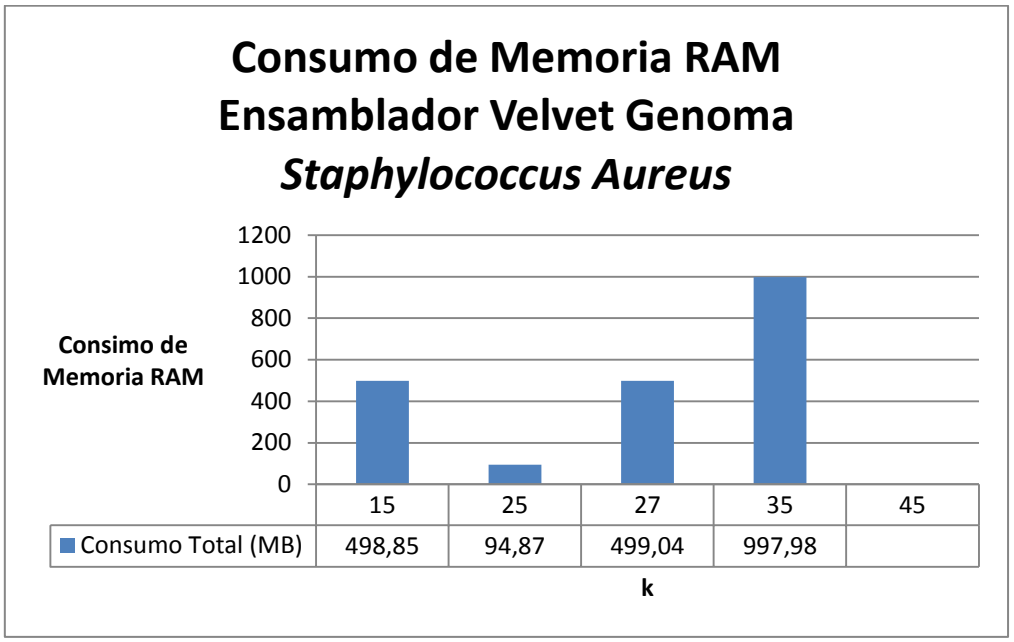



Tabla 20 Gráfico de consumo teórico de Memoria RAM Ejecución Ensamblador Velvet para el genoma *Staphylococcus Aureus*

En este ejemplo podemos observar una distribución un poco diferente de los consumos de memoria RAM, respecto al caso anterior. En este caso podemos observar que con k=25 el consumo de memoria RAM se reduce considerablemente. Para el resto de valores se observa un mayor consumo.

- **Estadísticas:**

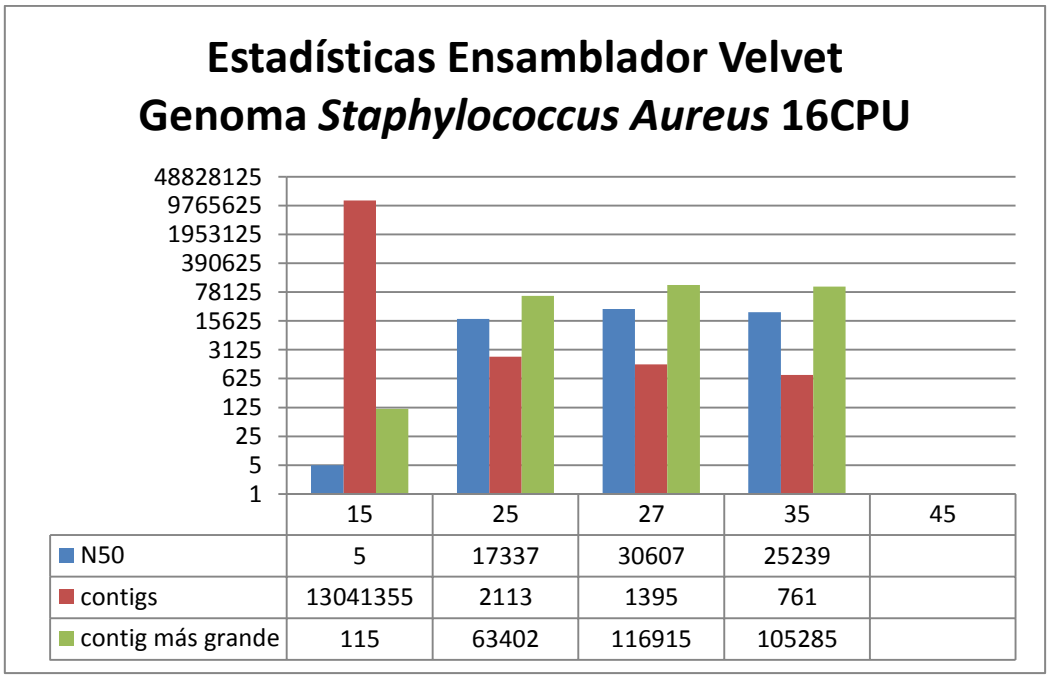
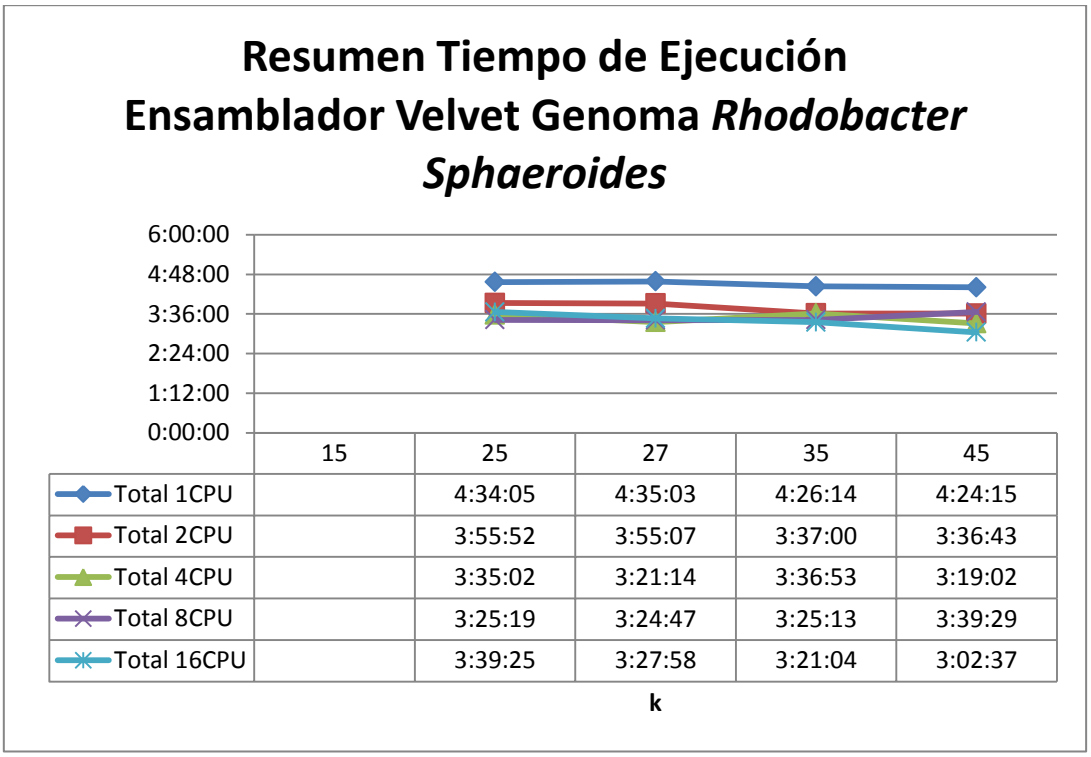



Tabla 21 Estadísticas Ensamblador Velvet 166 CPU Genoma *Staphylococcus Aureus*

En este caso podemos observar que el mejor valor de k=27. Se han cogido los valores para 16 CPUs debido a que ha sido la ejecución más rápida. Para el resto de CPUs los valores son muy similares

### *5.2.2 Rhodobacter Sphaeroides*

- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Total 1CPU | | 4:34:05 | 4:35:03 | 4:26:14 | 4:24:15 |
| Total 2CPU | | 3:55:52 | 3:55:07 | 3:37:00 | 3:36:43 |
| Total 4CPU | | 3:35:02 | 3:21:14 | 3:36:53 | 3:19:02 |
| Total 8CPU | | 3:25:19 | 3:24:47 | 3:25:13 | 3:39:29 |
| Total 16CPU | | 3:39:25 | 3:27:58 | 3:21:04 | 3:02:37 |

Tabla 22 Tiempo de Ejecución Ensamblador Velvet Genoma *Rhodobacter Sphaeroides*

En este caso, y cómo hemos visto en el genoma anterior. La mayor mejora se puede observar entre 1CPU y 2CPUs a partir de ahí la mejora es casi inexistente.

Esto es debido a que el ensamblador no tiene suficientes tareas para repartir entre todas las CPUs asignadas.

- **Consumo de Memoria RAM:**

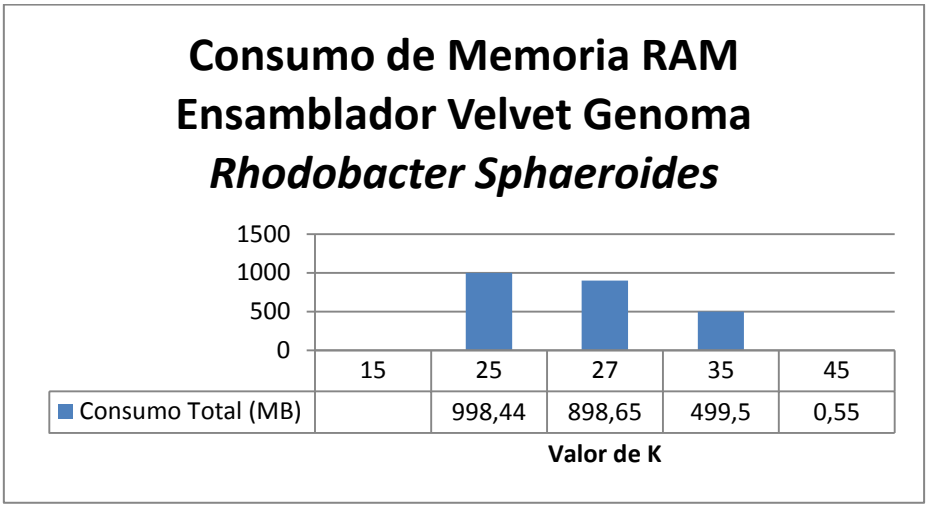



Tabla 23 Gráfico de consumo de Memoria RAM Ejecución Ensamblador Velvet para el genoma *Rhodobacter Sphaeroides*

En este ejemplo, podemos observar que cuanto más aumenta el valor de k, menor es el consumo de memoria RAM.

- **Estadísticas:**

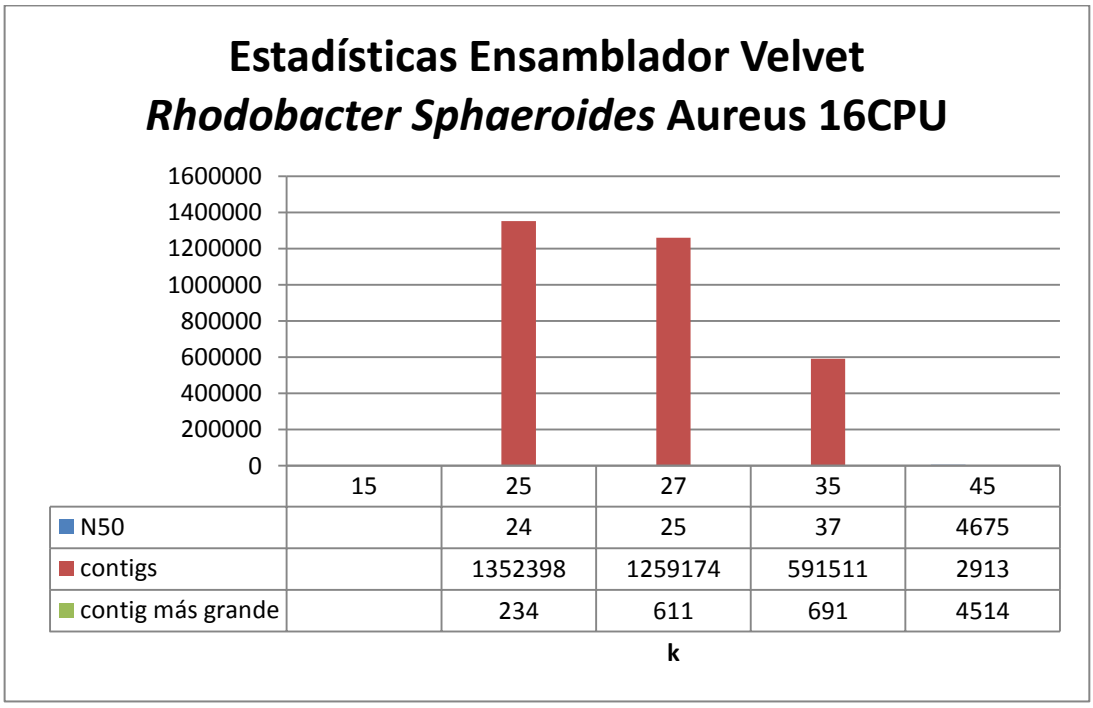



Tabla 24 Estadísticas Ensamblador Velvet 16CPU Cromosoma *Rhodobacter Sphaeroides*

En este caso el mejor valor de k, es k=35. Pero como podemos observar con valor de N50 bastante bajo.

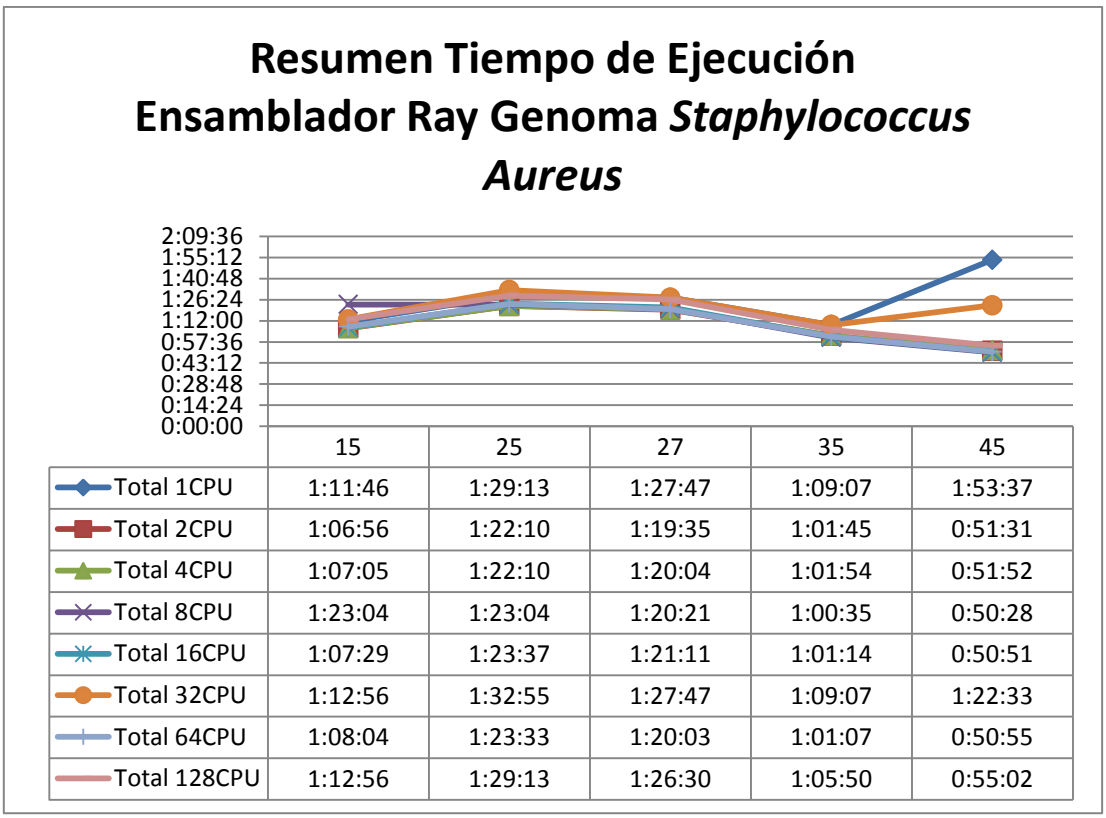
### 5.2.3 Home Sapiens Sapiens Cromosoma 14

Las pruebas correspondientes al cromosoma 14 del genoma *Homo Sapiens Sapiens* no se han podido realizar debido a que superaba el máximo de memoria permitido por las máquinas de memoria compartida.

## 5.3 Resultados Ray

### 5.3.1 Sthapylococcus Aureus

- **Tiempo de Ejecución:**

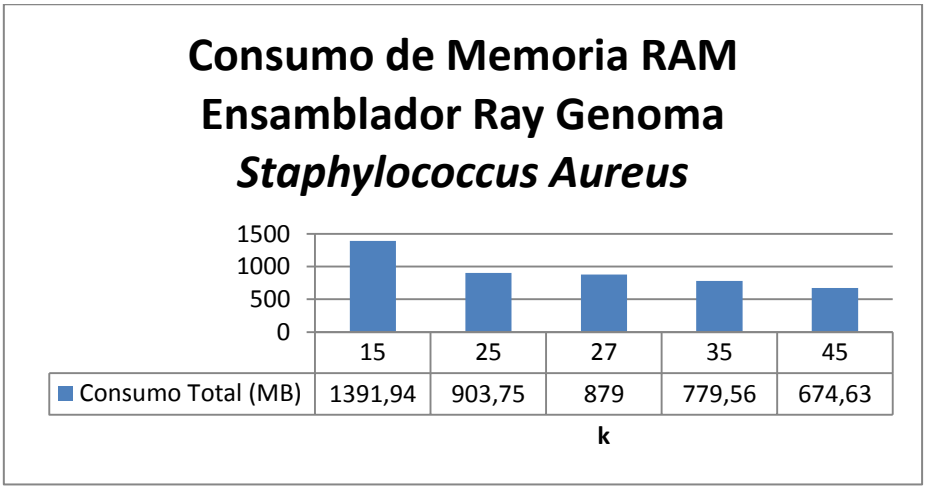



| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Total 1CPU | 1:11:46 | 1:29:13 | 1:27:47 | 1:09:07 | 1:53:37 |
| Total 2CPU | 1:06:56 | 1:22:10 | 1:19:35 | 1:01:45 | 0:51:31 |
| Total 4CPU | 1:07:05 | 1:22:10 | 1:20:04 | 1:01:54 | 0:51:52 |
| Total 8CPU | 1:23:04 | 1:23:04 | 1:20:21 | 1:00:35 | 0:50:28 |
| Total 16CPU | 1:07:29 | 1:23:37 | 1:21:11 | 1:01:14 | 0:50:51 |
| Total 32CPU | 1:12:56 | 1:32:55 | 1:27:47 | 1:09:07 | 1:22:33 |
| Total 64CPU | 1:08:04 | 1:23:33 | 1:20:03 | 1:01:07 | 0:50:55 |
| Total 128CPU | 1:12:56 | 1:29:13 | 1:26:30 | 1:05:50 | 0:55:02 |

Tabla 25 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Staphylococcus Aureus*

Cómo podemos observar, para el ejemplo del genoma *Staphylococcus Aureus* el tiempo de ejecución es bastante similar para todos los números de CPU. Para una ejecución de una sola CPU el tiempo es mayor que para el resto, pero luego la ejecución paralela no presenta muchas ventajas en términos temporales. Cómo en casos anteriores, esto es debido a que el ensamblador no tiene suficientes tareas como para que todas las CPUs puedan aportar mejora a la ejecución.

- **Consumo de Memoria RAM:**

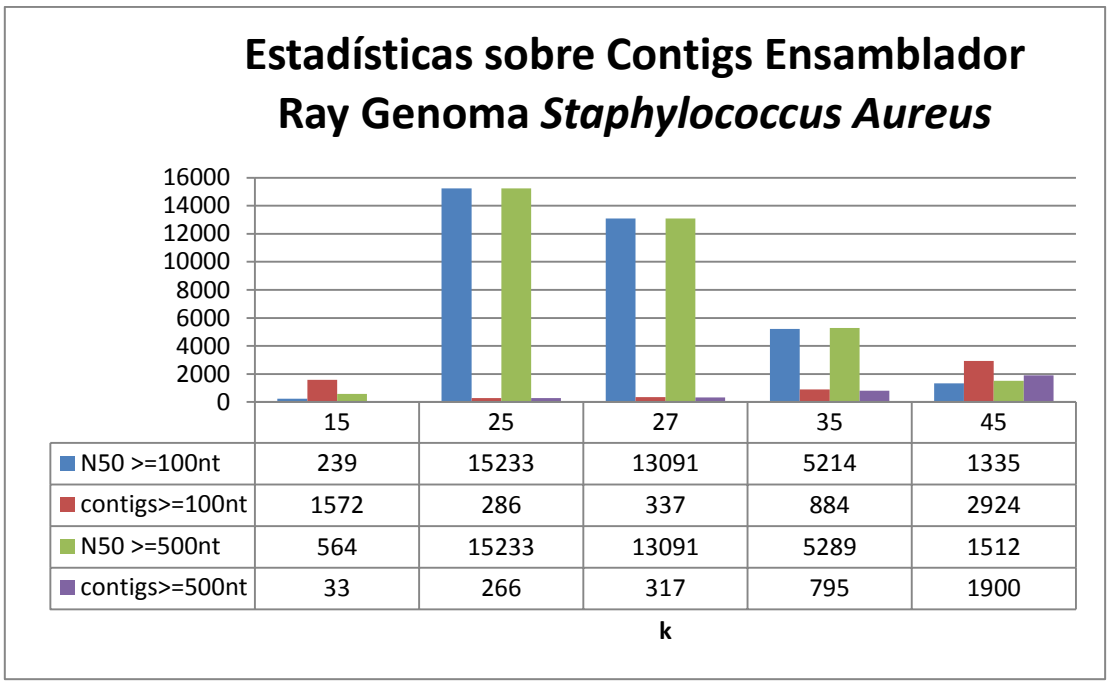



| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Consumo Total (MB) | 1391,94 | 903,75 | 879 | 779,56 | 674,63 |

Tabla 26 Consumo de Memoria RAM Ensamblador Ray Genoma *Staphylococcus Aureus*

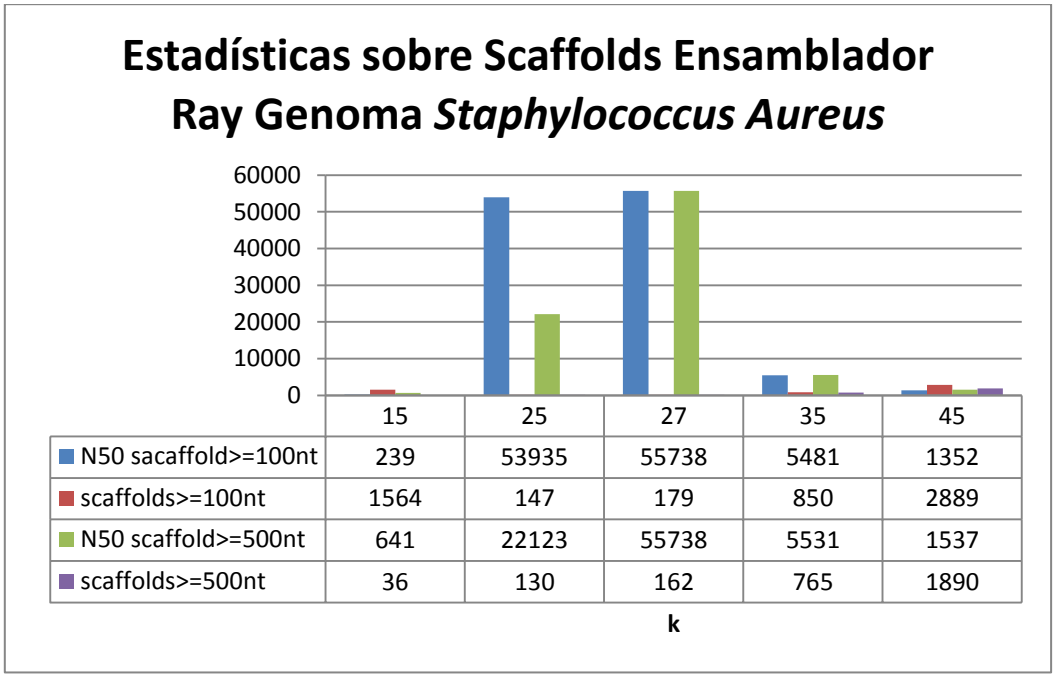
En el ejemplo actual se presenta una mejora de consumo de memoria RAM cuanto más aumenta el valor de k.

- **Estadísticas:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| N50 >=100nt | 239 | 15233 | 13091 | 5214 | 1335 |
| contigs>=100nt | 1572 | 286 | 337 | 884 | 2924 |
| N50 >=500nt | 564 | 15233 | 13091 | 5289 | 1512 |
| contigs>=500nt | 33 | 266 | 317 | 795 | 1900 |

Tabla 27 Estadísticas sobre Contigs Ensamblador Ray Genoma *Staphylococcus Aureus*



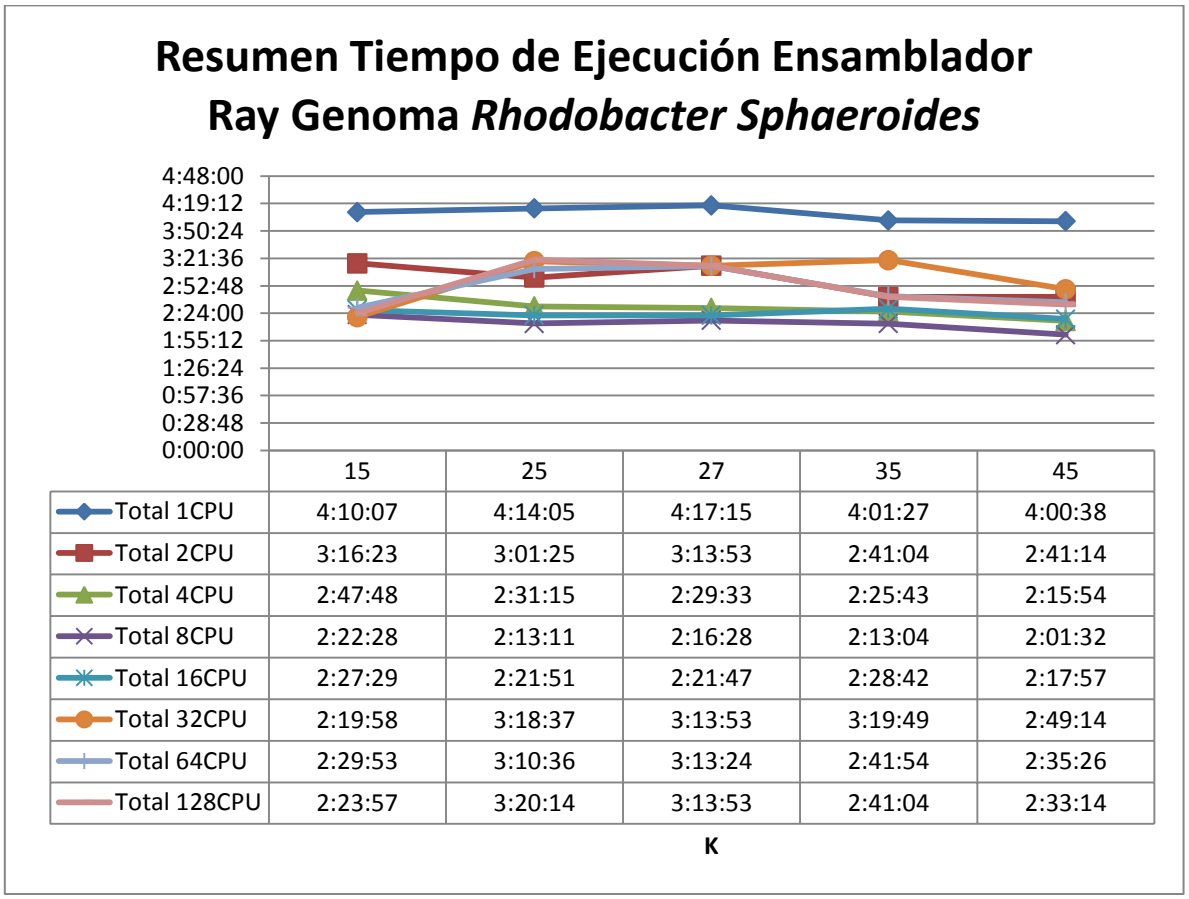

| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| N50 sacaffold>=100nt | 239 | 53935 | 55738 | 5481 | 1352 |
| scaffolds>=100nt | 1564 | 147 | 179 | 850 | 2889 |
| N50 scaffold>=500nt | 641 | 22123 | 55738 | 5531 | 1537 |
| scaffolds>=500nt | 36 | 130 | 162 | 765 | 1890 |

Tabla 28 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Staphylococcus Aureus*

En este ensamblador podemos observar una mejora respecto al resto de ensambladores. Este es capaz de extraer datos de Scaffolds, el siguiente nivel de la jerarquía del proceso de ensamblado (Ver Sección 2.2.2) Lo que presenta una mejora respecto al resto de ensambladores.

### 5.3.2 *Rhodobacter Sphaeroides*

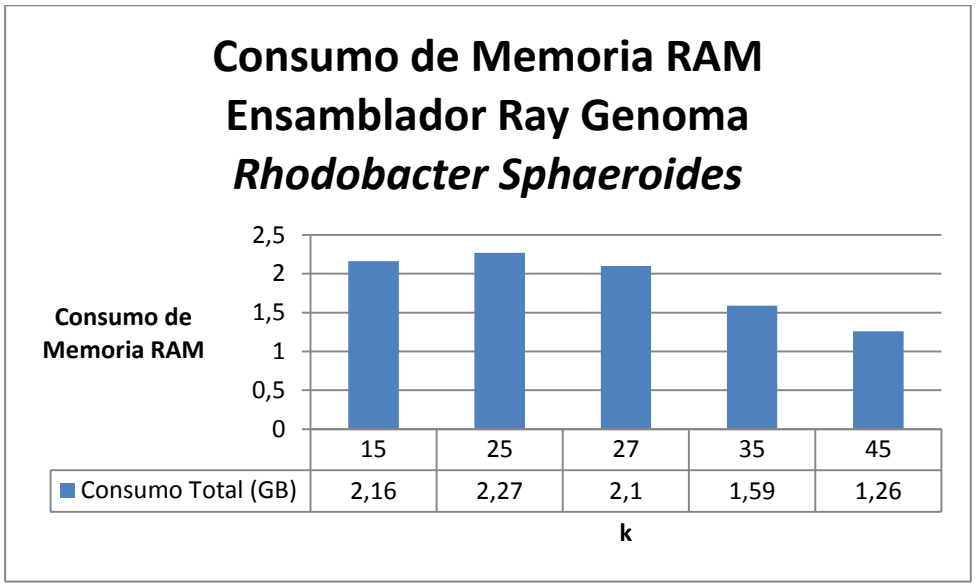
- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Total 1CPU | 4:10:07 | 4:14:05 | 4:17:15 | 4:01:27 | 4:00:38 |
| Total 2CPU | 3:16:23 | 3:01:25 | 3:13:53 | 2:41:04 | 2:41:14 |
| Total 4CPU | 2:47:48 | 2:31:15 | 2:29:33 | 2:25:43 | 2:15:54 |
| Total 8CPU | 2:22:28 | 2:13:11 | 2:16:28 | 2:13:04 | 2:01:32 |
| Total 16CPU | 2:27:29 | 2:21:51 | 2:21:47 | 2:28:42 | 2:17:57 |
| Total 32CPU | 2:19:58 | 3:18:37 | 3:13:53 | 3:19:49 | 2:49:14 |
| Total 64CPU | 2:29:53 | 3:10:36 | 3:13:24 | 2:41:54 | 2:35:26 |
| Total 128CPU | 2:23:57 | 3:20:14 | 3:13:53 | 2:41:04 | 2:33:14 |

Tabla 29 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Rhodobacter Sphaeroides*

- **Consumo de Memoria RAM:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Consumo Total (GB) | 2,16 | 2,27 | 2,1 | 1,59 | 1,26 |

Tabla 30 Consumo de Memoria RAM Ensamblador Ray Genoma *Rhodobacter Sphaeroides*

- **Estadísticas:**

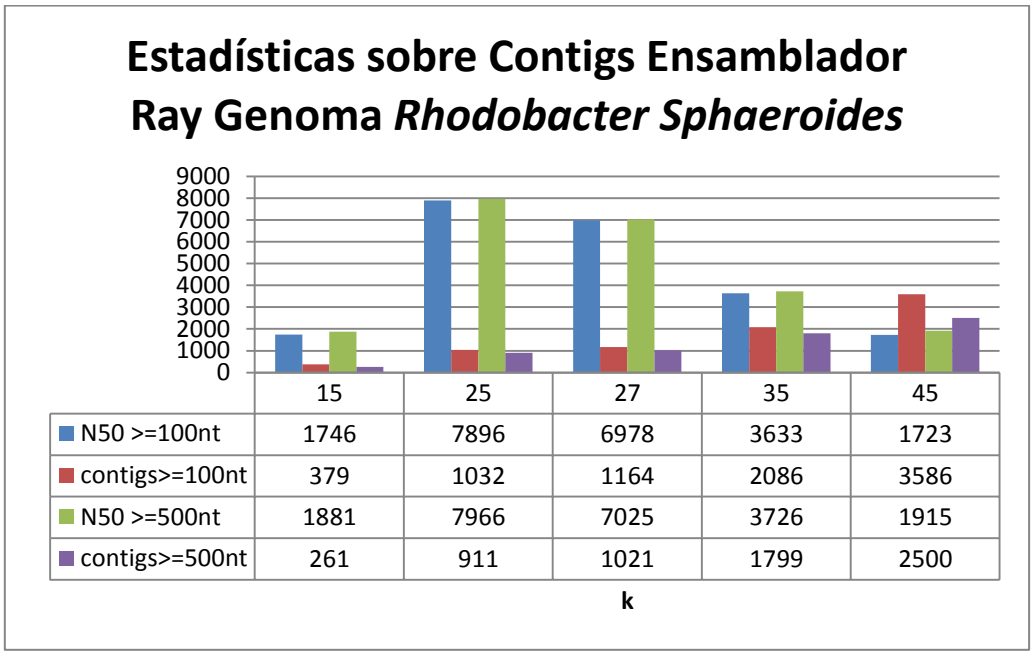



Tabla 31 Estadísticas sobre Contigs Ensamblador Ray Genoma *Rhodobacter Sphaeroides*

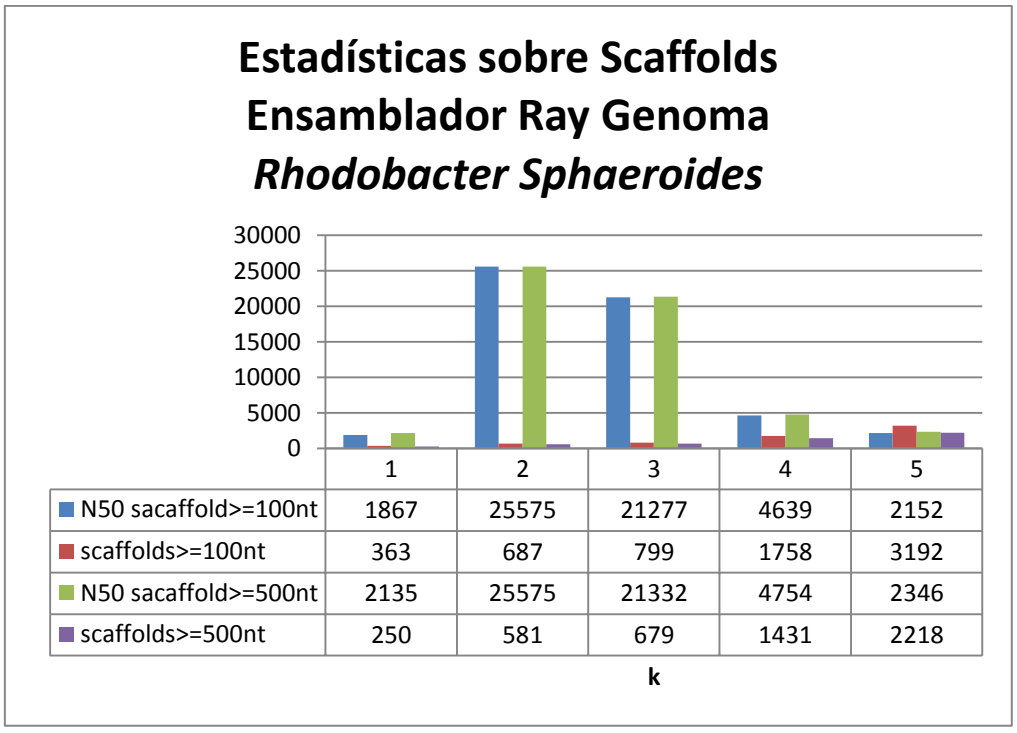
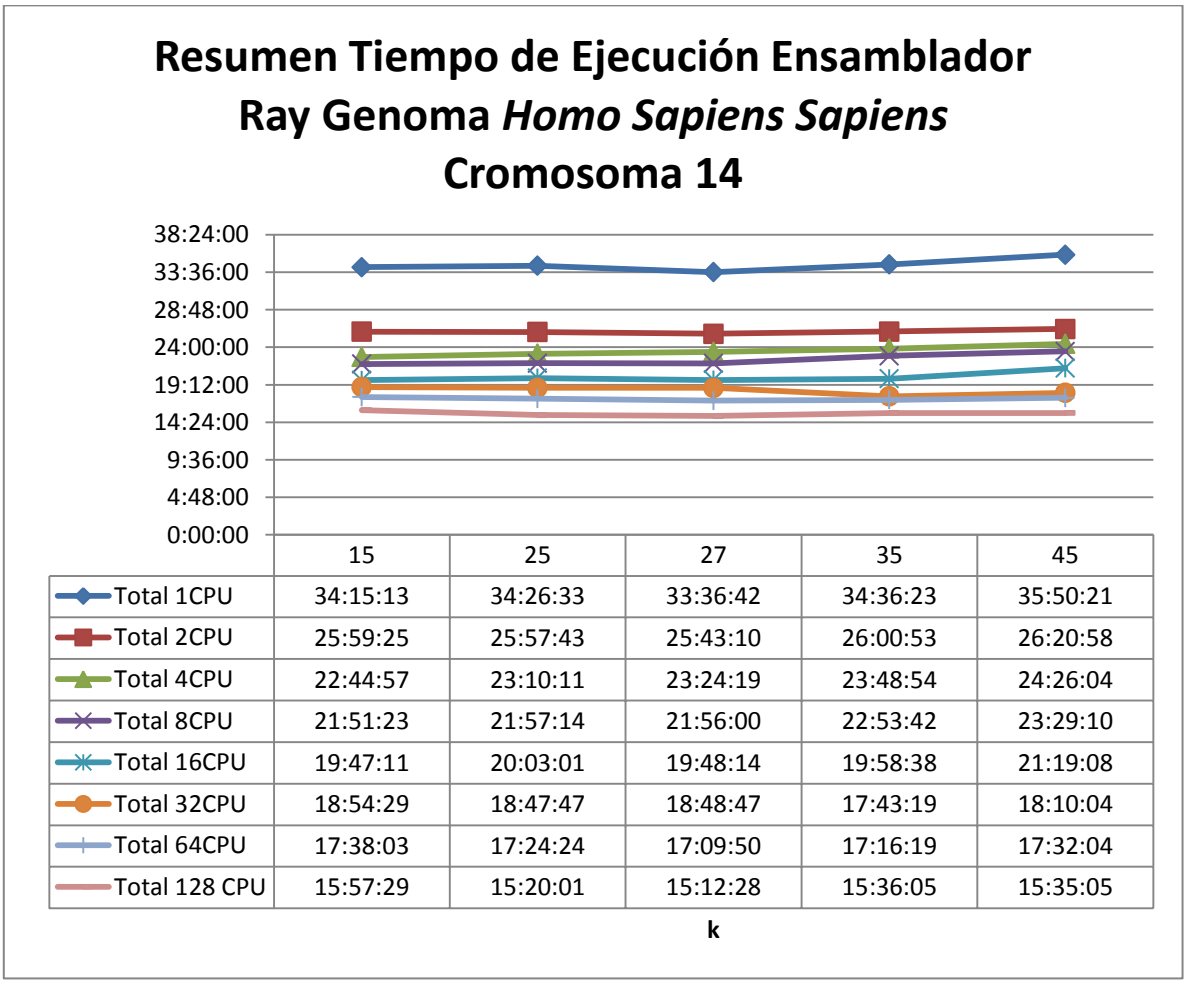



Tabla 32 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Rhodobacter Sphaeroides*

### 5.3.3 Home Sapiens Sapiens Cromosoma 14

- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Total 1CPU | 34:15:13 | 34:26:33 | 33:36:42 | 34:36:23 | 35:50:21 |
| Total 2CPU | 25:59:25 | 25:57:43 | 25:43:10 | 26:00:53 | 26:20:58 |
| Total 4CPU | 22:44:57 | 23:10:11 | 23:24:19 | 23:48:54 | 24:26:04 |
| Total 8CPU | 21:51:23 | 21:57:14 | 21:56:00 | 22:53:42 | 23:29:10 |
| Total 16CPU | 19:47:11 | 20:03:01 | 19:48:14 | 19:58:38 | 21:19:08 |
| Total 32CPU | 18:54:29 | 18:47:47 | 18:48:47 | 17:43:19 | 18:10:04 |
| Total 64CPU | 17:38:03 | 17:24:24 | 17:09:50 | 17:16:19 | 17:32:04 |
| Total 128 CPU | 15:57:29 | 15:20:01 | 15:12:28 | 15:36:05 | 15:35:05 |

Tabla 33 Resumen Tiempo de Ejecución Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14

El ejemplo del ensamblador Ray con el genoma *Homo Sapiens Sapiens* Cromosoma 14 es un ejemplo bastante bueno para ver las virtudes de los ensambladores que utilizan sistemas de memoria distribuida. En este ejemplo si se presenta una mejoría aumentando el número de CPUs utilizadas. De 1CPU a 128CPUs hay una mejora bastante significativa. Esto es debido a que el genoma *Homo Sapiens Sapiens* Cromosoma 14 presenta una complejidad suficiente para conseguir que todas las CPUs aporten mejoría al proceso.

- **Consumo de Memoria RAM:**

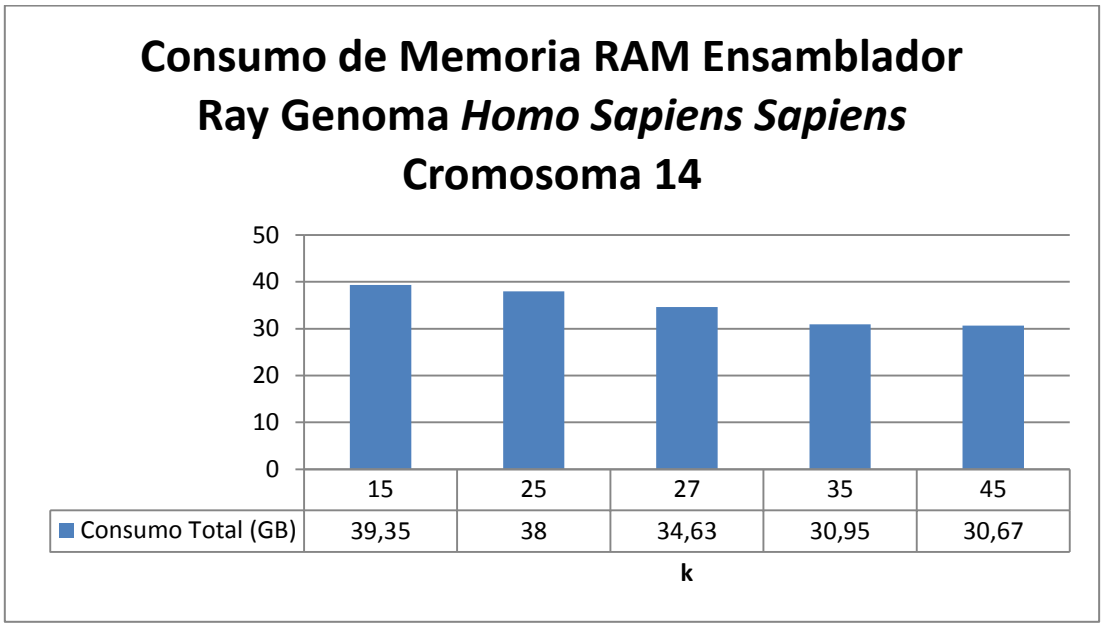



Tabla 34 Consumo de Memoria RAM Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14

Cómo en casos anteriores, cuando aumenta el valor de k, observamos una mejoría en el consumo de memoria RAM. En este caso el consumo de memoria RAM es muy elevado y es posible abordarlo al utilizar memoria distribuida ya que los nodos de Magerit 2 solo disponen de 30 Gb de memoria RAM.

- **Estadísticas:**

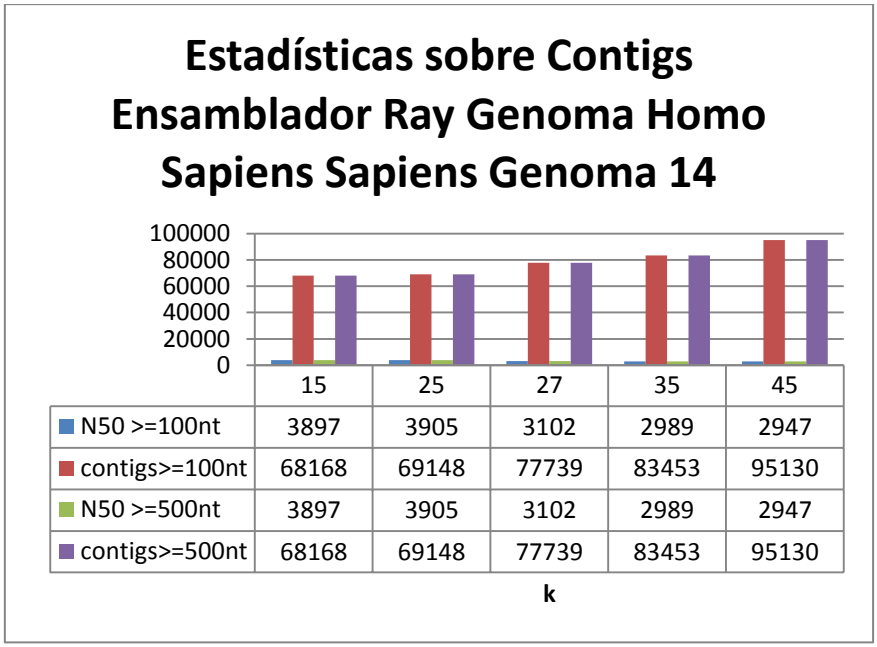



Tabla 35 Estadísticas sobre Contigs Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14

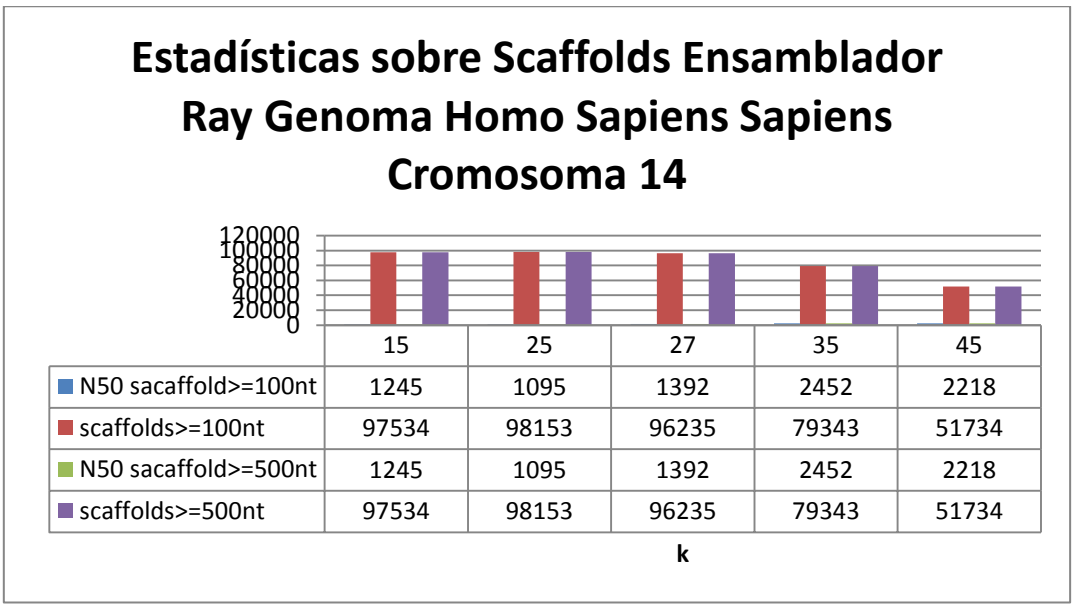



**Tabla 36 Estadísticas sobre Scaffolds Ensamblador Ray Genoma *Homo Sapiens Sapiens* Cromosoma 14**

En este ejemplo podemos observar que el mejor valor de K, es K =45 y que todos los Contigs y Scaffolds son >= 500nt, es decir que son de gran longitud.

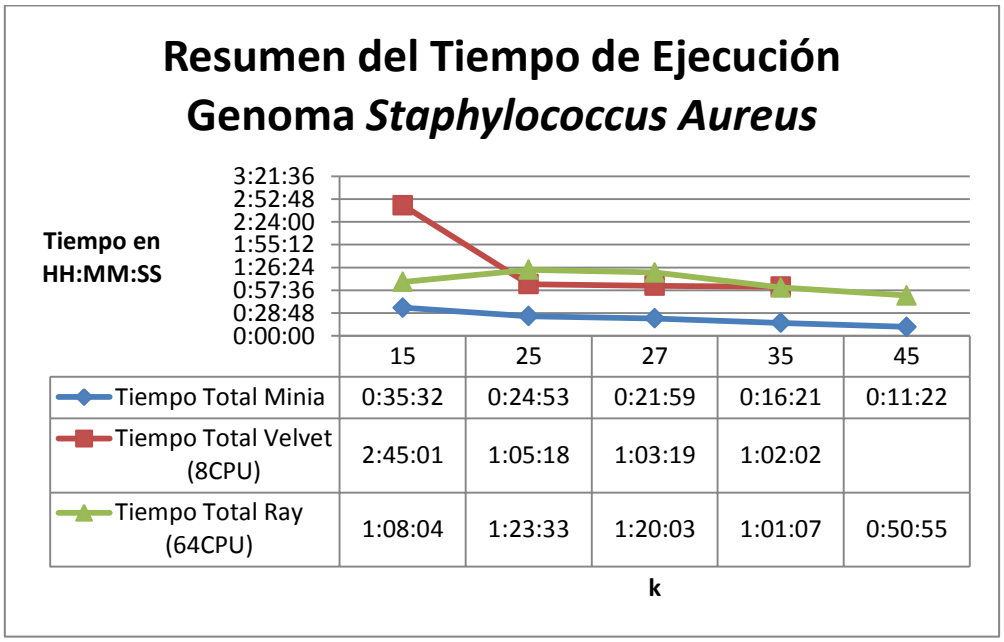
## 5.4 Resumen de Resultados

En este apartado vamos a resumir los mejores resultados de cada ensamblador para cada genoma determinado. Así podremos comparar de manera sencilla los tres ensambladores observando unas pocas gráficas.

El tiempo de ejecución, debido a las numerosas combinaciones posibles debido al número de CPUs utilizadas, se compara con la mejor configuración para cada ensamblador.

### 5.4.1 Sthapylococcus Aureus

- **Tiempo de Ejecución:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Tiempo Total Minia | 0:35:32 | 0:24:53 | 0:21:59 | 0:16:21 | 0:11:22 |
| Tiempo Total Velvet (8CPU) | 2:45:01 | 1:05:18 | 1:03:19 | 1:02:02 | |
| Tiempo Total Ray (64CPU) | 1:08:04 | 1:23:33 | 1:20:03 | 1:01:07 | 0:50:55 |

Tabla 37 Resumen del Tiempo de Ejecución Genoma *Staphylococcus Aureus*

La comparación de tiempos de ejecución se lleva a cabo con la configuración de 1CPU para el ensamblador Minia (la única posible) 8CPUs para el ensamblador Velvet y 64 CPUs para el ensamblador Ray.

En la comparación podemos observar que el ensamblador Ray se mantiene siempre como el ensamblador más rápido para todos los Valores de K.

Entre el ensamblador Velvet y el Ensamblador Ray, no hay diferencias muy significativas salvo para el valor de K =15, para el resto el comportamiento es bastante similar.

Eso nos indica que el genoma no es suficientemente complejo como para que un ensamblador de Memoria distribuida (Ray) marque la diferencia.

- **Consumo de Memoria RAM:**

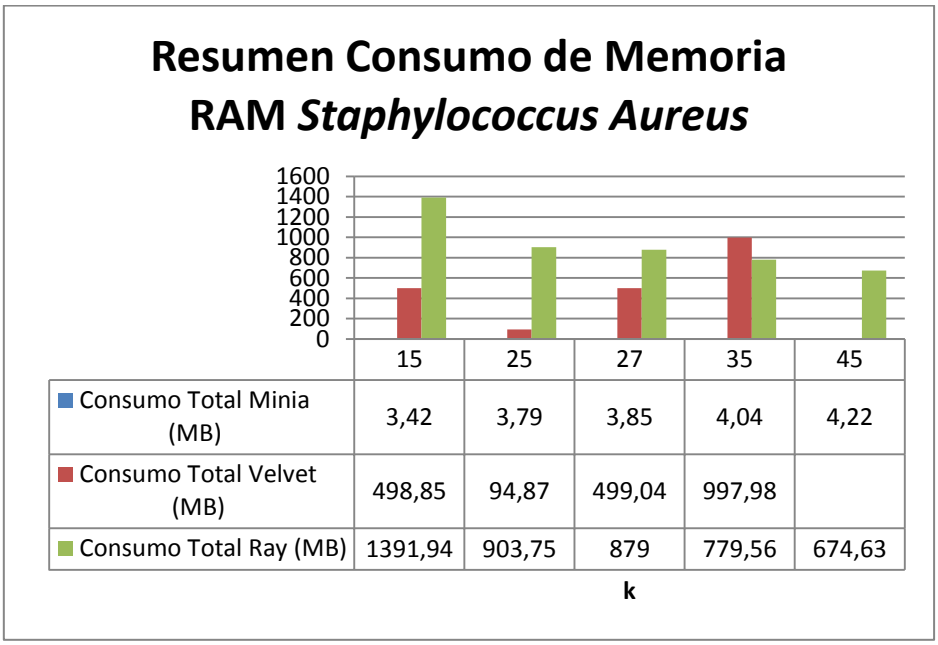



Tabla 38 Resumen Consumo de Memoria RAM *Staphylococcus Aureus*

En este ejemplo podemos observar que el consumo de memoria RAM es muy dispar entre los ensambladores. El ensamblador Minia tiene un sorprendente consumo óptimo de memoria RAM. Luego entre Velvet y Ray hay menos diferencia, pero se observa que la utilización de un sistema distribuido conlleva un mayor consumo de memoria RAM.

- **Estadísticas:**

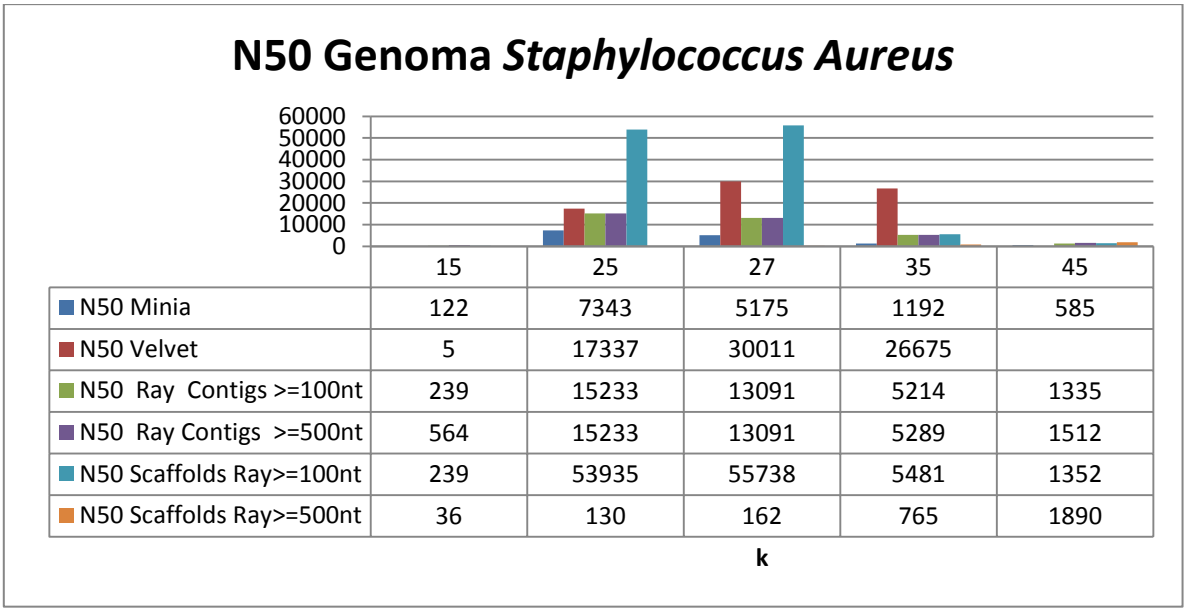
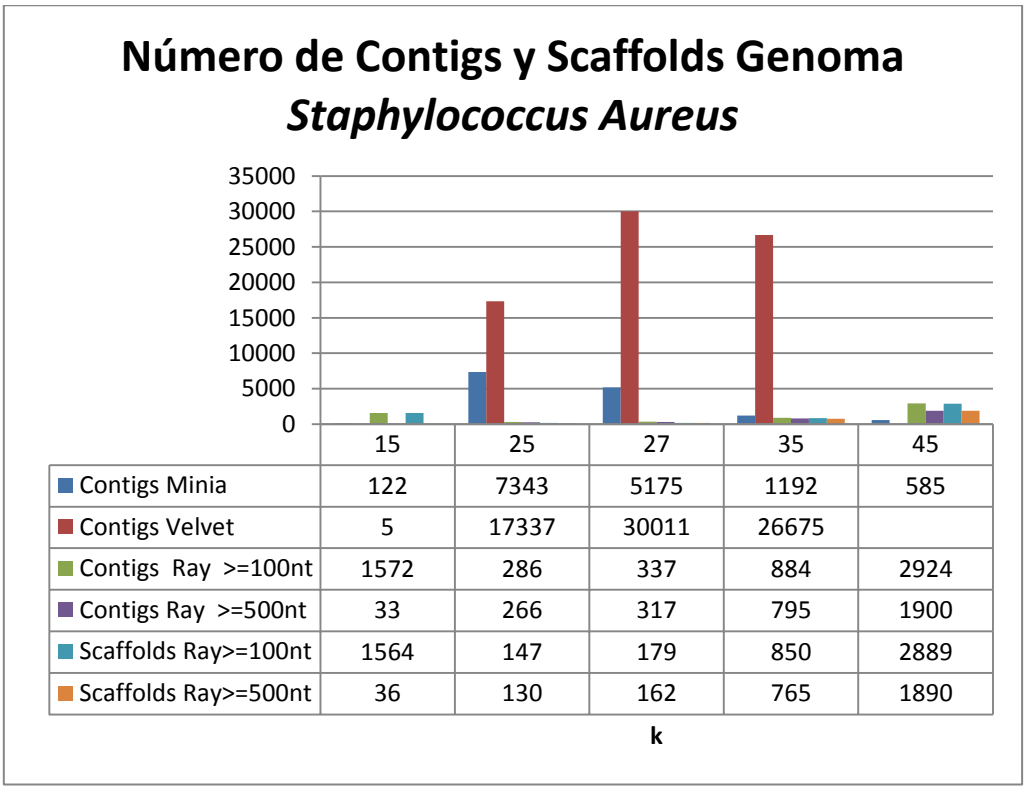



Tabla 39 N50 Genoma *Staphylococcus Aureus*

En este ejemplo podemos observar que el ensamblador Minia presenta un comportamiento bastante discreto en valores N50, los ensambladores Velvet y Ray presentan mejores resultados. Siendo Velvet el que marcaría la diferencia. Por lo que para un genoma "sencillo" cómo el *Staphylococcus Aureus* un ensamblador de memoria compartida es suficiente y presenta buenos resultados.

Aunque el ensamblador Ray es el único que es capaz de presentar un nivel más en la jerarquía (Scaffolds) del proceso de ensamblado.

Si es necesario obtener resultados bastante concretos y de gran calidad convendría utilizar el ensamblador Ray, teniendo en cuenta que debemos poder utilizar un sistema de memoria distribuida.



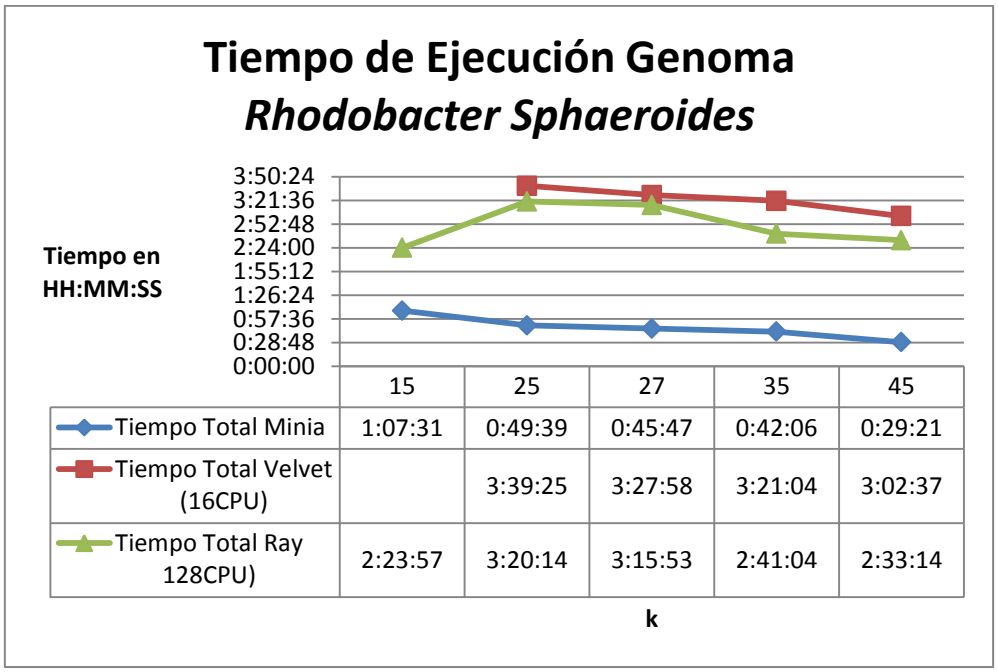

| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Contigs Minia | 122 | 7343 | 5175 | 1192 | 585 |
| Contigs Velvet | 5 | 17337 | 30011 | 26675 | |
| Contigs Ray >=100nt | 1572 | 286 | 337 | 884 | 2924 |
| Contigs Ray >=500nt | 33 | 266 | 317 | 795 | 1900 |
| Scaffolds Ray>=100nt | 1564 | 147 | 179 | 850 | 2889 |
| Scaffolds Ray>=500nt | 36 | 130 | 162 | 765 | 1890 |

**Tabla 40 Número de Contigs y Scaffolds Genoma *Staphylococcus Aureus***

Para el número de Contigs y Scaffolds pasa algo análogo al caso de los valores N50, en este caso hay que tener en cuenta que cuanto mayor es el valor de N50, menor debería ser el número de contigs. Ya que menos contigs contienen más longitud del genoma que se intenta conseguir.

### 5.4.2 *Rhodobacter Sphaeroides*

- **Tiempo de Ejecución:**

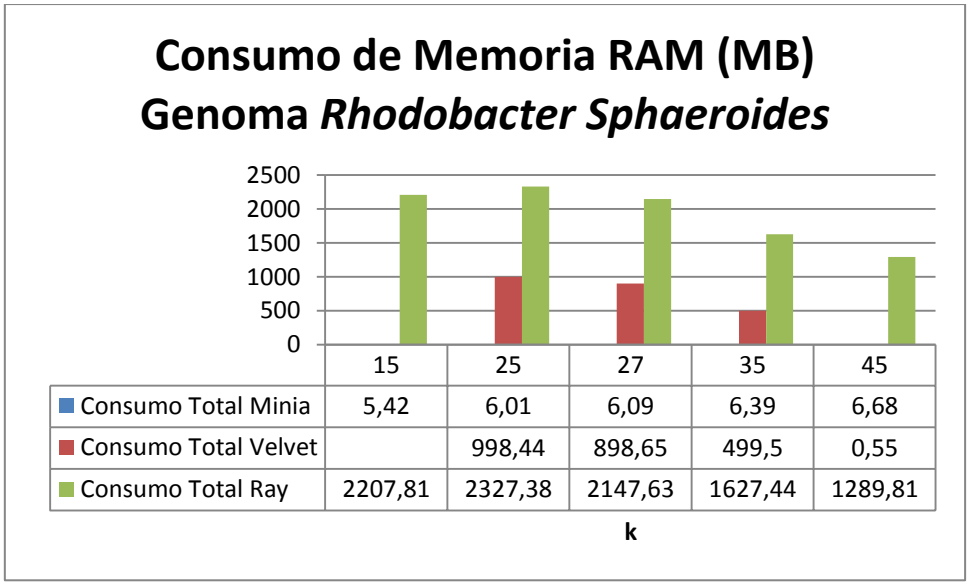



| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Tiempo Total Minia | 1:07:31 | 0:49:39 | 0:45:47 | 0:42:06 | 0:29:21 |
| Tiempo Total Velvet (16CPU) | | 3:39:25 | 3:27:58 | 3:21:04 | 3:02:37 |
| Tiempo Total Ray 128CPU) | 2:23:57 | 3:20:14 | 3:15:53 | 2:41:04 | 2:33:14 |

Tabla 41 Tiempo de Ejecución Genoma *Rhodobacter Sphaeroides*

En este ejemplo, pasa un proceso análogo al del genoma anterior. El ensamblador Minia presenta unos resultados sorprendentes de rendimiento. Luego están bastante parejos los ensambladores Ray y Velvet.

- **Consumo de Memoria RAM:**



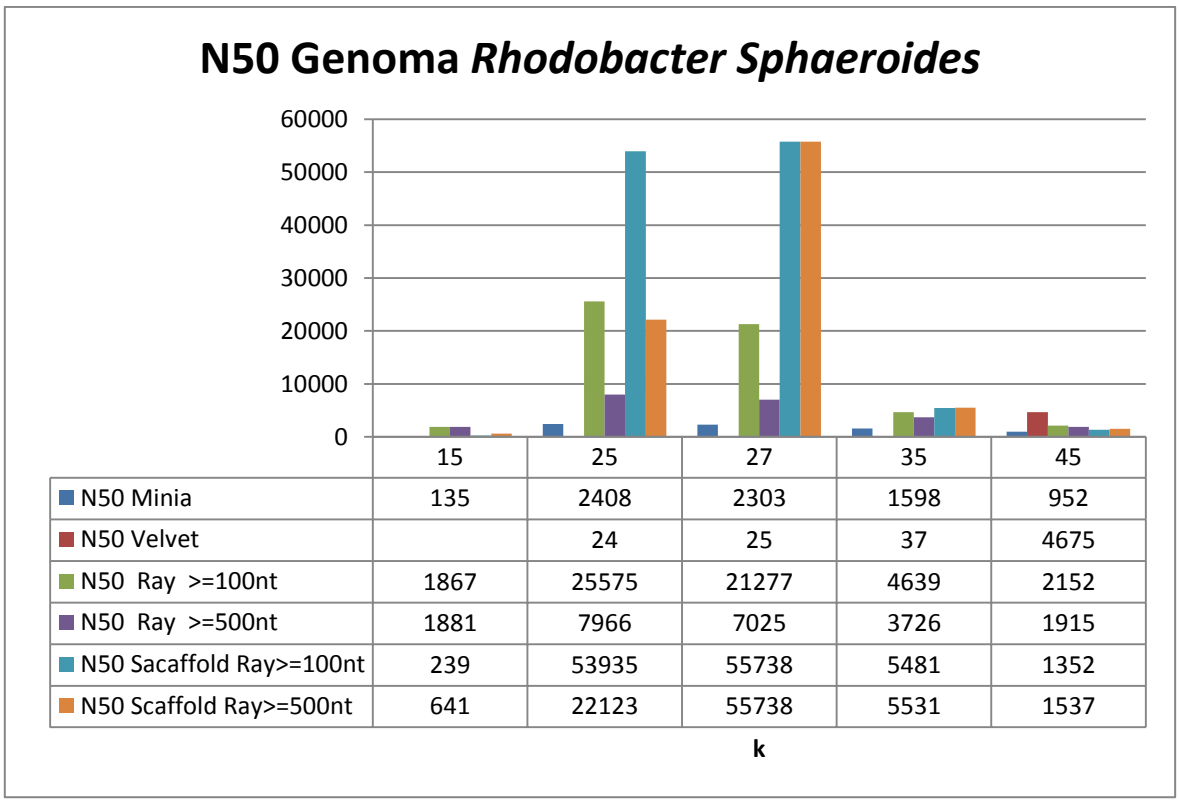

| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Consumo Total Minia | 5,42 | 6,01 | 6,09 | 6,39 | 6,68 |
| Consumo Total Velvet | | 998,44 | 898,65 | 499,5 | 0,55 |
| Consumo Total Ray | 2207,81 | 2327,38 | 2147,63 | 1627,44 | 1289,81 |

Tabla 42 Consumo de Memoria RAM Genoma *Rhodobacter Sphaeroides*

Una vez más, como en el ejemplo anterior, podemos observar grandes discrepancias en el consumo de memoria RAM. Siendo el ensamblador Minia el que mejor rendimiento presenta y el

ensamblador Ray el que mayor consumo de Memoria RAM presenta debido a su característica de utilizar un sistema de memoria distribuida.

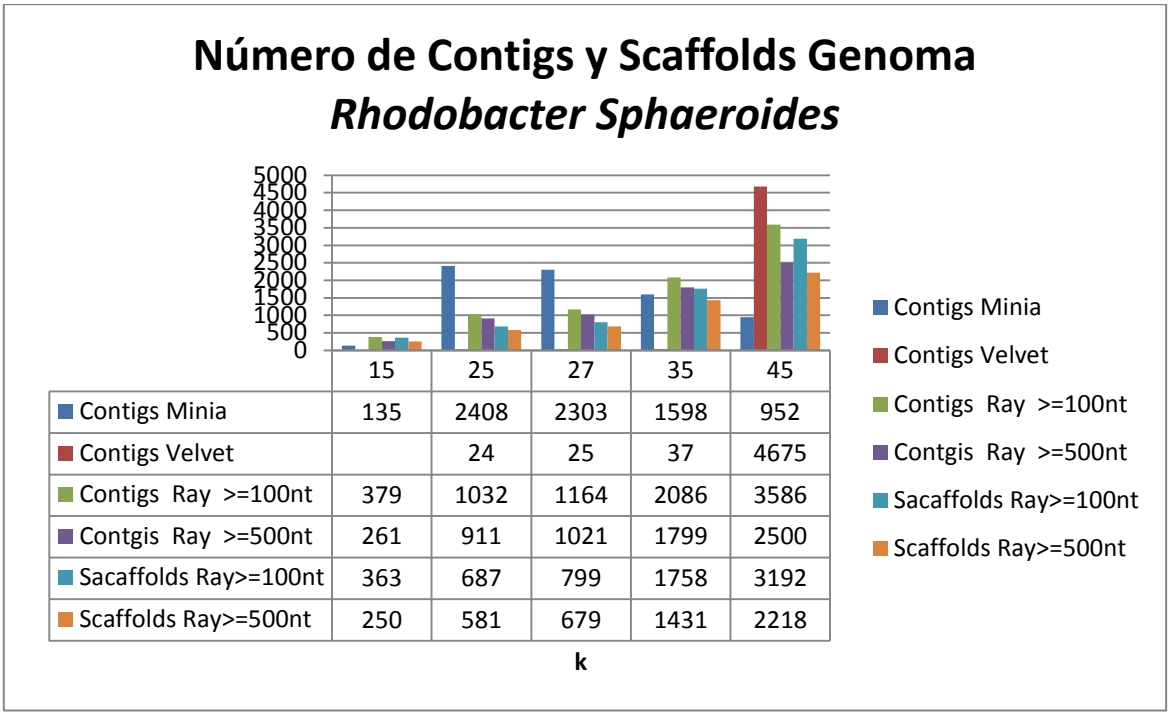
- **Estadísticas:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| N50 Minia | 135 | 2408 | 2303 | 1598 | 952 |
| N50 Velvet | | 24 | 25 | 37 | 4675 |
| N50 Ray >=100nt | 1867 | 25575 | 21277 | 4639 | 2152 |
| N50 Ray >=500nt | 1881 | 7966 | 7025 | 3726 | 1915 |
| N50 Sacaffold Ray>=100nt | 239 | 53935 | 55738 | 5481 | 1352 |
| N50 Scaffold Ray>=500nt | 641 | 22123 | 55738 | 5531 | 1537 |

k

Tabla 43 N50 Genoma *Rhodobacter Sphaeroides*




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Contigs Minia | 135 | 2408 | 2303 | 1598 | 952 |
| Contigs Velvet | | 24 | 25 | 37 | 4675 |
| Contigs Ray >=100nt | 379 | 1032 | 1164 | 2086 | 3586 |
| Contgis Ray >=500nt | 261 | 911 | 1021 | 1799 | 2500 |
| Sacaffolds Ray>=100nt | 363 | 687 | 799 | 1758 | 3192 |
| Scaffolds Ray>=500nt | 250 | 581 | 679 | 1431 | 2218 |

k

Tabla 44 Número de Contigs y Scaffolds Genoma *Rhodobacter Sphaeroides*

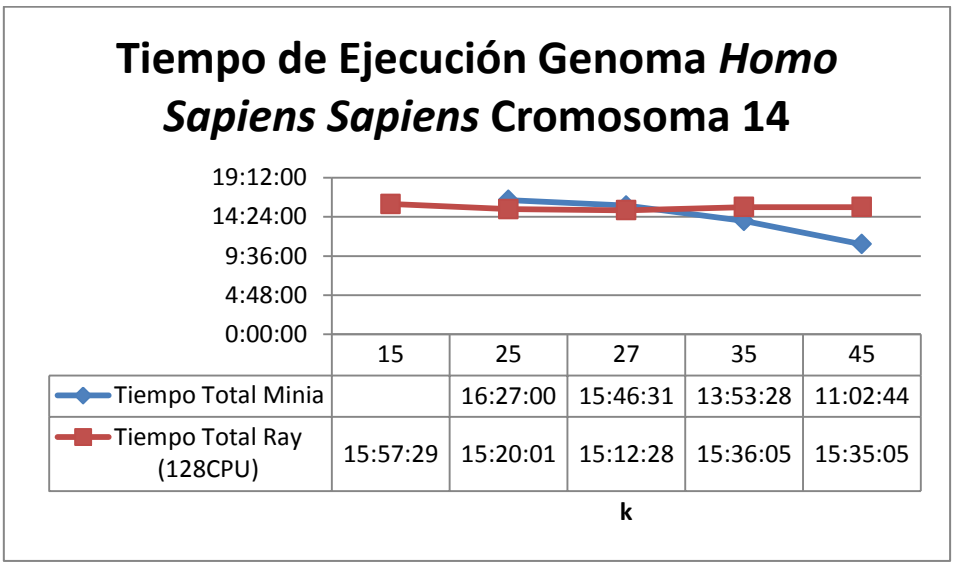
En este ejemplo podemos observar que el ensamblador Ray presenta mejores resultados que los ensambladores Velvet y Minia. Siendo el ensamblador Velvet el que peores resultados nos da con diferencia, exceptuando para el valor k = 45, lo que nos puede indicar que para este genoma el ensamblador Velvet funciona bien con valores de k relativamente grandes.

Para el número de Contigs y Scaffolds el resultado es bastante similar al del genoma anterior, destacando que el ensamblador Ray es el único que presenta Scaffolds.

### 5.4.3 *Homo Sapiens Sapiens* Cromosoma 14

Para este genoma solo podemos comparar los ensambladores Minia y Ray ya que el ensamblador Velvet no es capaz de realizar las operaciones necesarias con 30 GB de RAM.

- **Tiempo de Ejecución:**



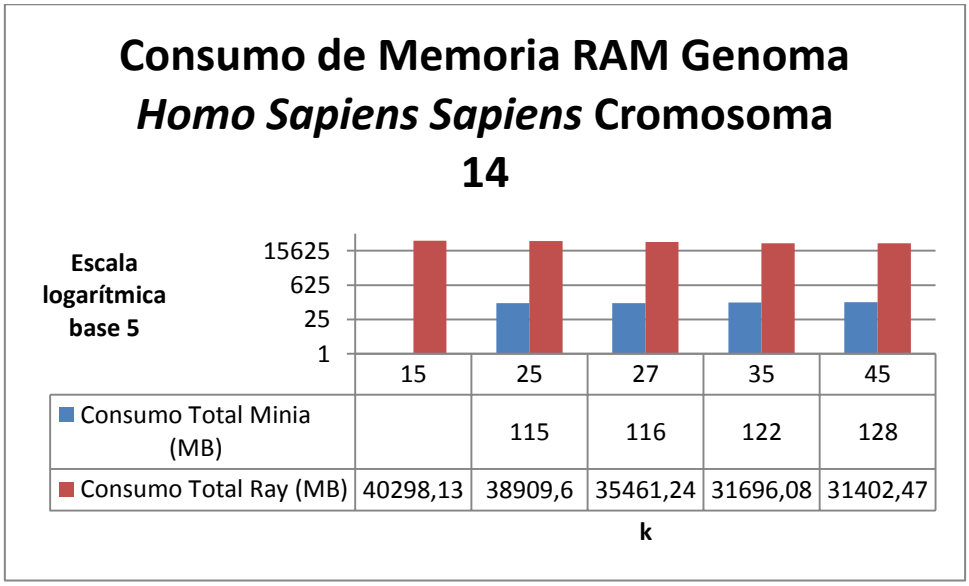

| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Tiempo Total Minia | | 16:27:00 | 15:46:31 | 13:53:28 | 11:02:44 |
| Tiempo Total Ray (128CPU) | 15:57:29 | 15:20:01 | 15:12:28 | 15:36:05 | 15:35:05 |

Tabla 45 Tiempo de Ejecución Genoma *Homo Sapiens Sapiens* Cromosoma 14

En este caso, y por primera vez, podemos observar que los rendimientos entre el ensamblador Minia, y la mejor configuración del ensamblador Ray, son bastante parejos, salvo para k =35 y k=45.

Este es un ejemplo bastante bueno de como para genomas que ya tienen un nivel de complejidad un sistema de memoria distribuida ya presenta bastantes ventajas.

- **Consumo de Memoria RAM:**

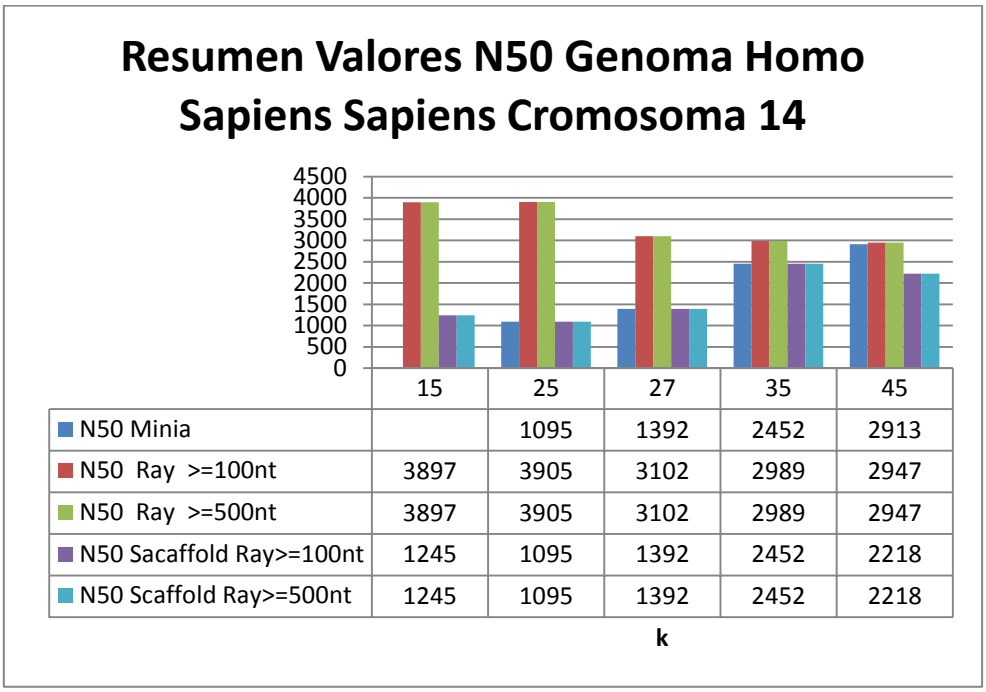



| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Consumo Total Minia (MB) | | 115 | 116 | 122 | 128 |
| Consumo Total Ray (MB) | 40298,13 | 38909,6 | 35461,24 | 31696,08 | 31402,47 |

Tabla 46 Consumo de Memoria RAM Genoma *Homo Sapiens Sapiens* Cromosoma 14

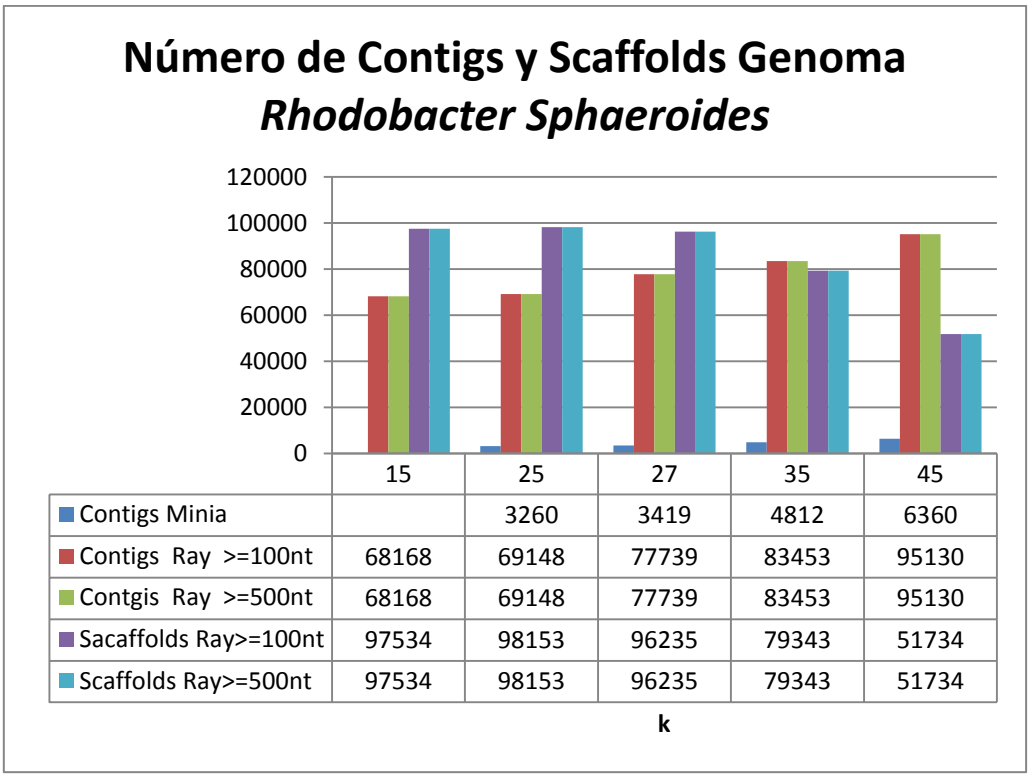
Aunque en este caso el tiempo de ejecución es bastante parecido, en el consumo de memoria RAM no pasa lo mismo. En este caso observamos que para el ensamblador Minia es suficiente con unos cientos de MB, mientras que para el ensamblador Ray es necesario utilizar miles de MB, o una decenas de GB.

- **Estadísticas:**




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| N50 Minia | | 1095 | 1392 | 2452 | 2913 |
| N50 Ray >=100nt | 3897 | 3905 | 3102 | 2989 | 2947 |
| N50 Ray >=500nt | 3897 | 3905 | 3102 | 2989 | 2947 |
| N50 Sacaffold Ray>=100nt | 1245 | 1095 | 1392 | 2452 | 2218 |
| N50 Scaffold Ray>=500nt | 1245 | 1095 | 1392 | 2452 | 2218 |

Tabla 47 Resumen Valores N50 Genoma *Homo Sapiens Sapiens* Cromosoma 14




| | 15 | 25 | 27 | 35 | 45 |
|---|---|---|---|---|---|
| Contigs Minia | | 3260 | 3419 | 4812 | 6360 |
| Contigs Ray >=100nt | 68168 | 69148 | 77739 | 83453 | 95130 |
| Contgis Ray >=500nt | 68168 | 69148 | 77739 | 83453 | 95130 |
| Sacaffolds Ray>=100nt | 97534 | 98153 | 96235 | 79343 | 51734 |
| Scaffolds Ray>=500nt | 97534 | 98153 | 96235 | 79343 | 51734 |

Tabla 48 Número de Contigs y Scaffolds Genoma *Rhodobacter Sphaeroides*

En este caso, podemos observar que el ensamblador que presenta mejores resultados es el ensamblador Ray, con la particularidad de que los contigs y scaffolds tienen una longitud bastante grande.

# 6. CONCLUSIONES

Gracias al trabajo realizado podemos sacar varias conclusiones. En primer lugar se puede tener claro que el desarrollo tecnológico ha permitido una mejoría en la producción, almacenamiento y procesamiento de datos biológicos para su investigación.

Como hemos visto en el transcurso del proyecto, la obtención del genoma completo de un ser vivo es un proceso de gran complejidad. Dependiendo de los análisis previos a la hora de realizar la experimentación podemos encontrar resultados muy optimizados, adecuados y fáciles de manejar. En cambio si el análisis no se realiza de la manera adecuada podemos obtener datos poco fiables, de manera lenta.

Dentro de este proyecto hemos analizado tres ensambladores distintos. Dos de memoria compartida, y uno de memoria distribuida, que además se podría utilizar en servicios de Cloud Computing con ciertas modificaciones (Cloud Computing Ensamblador Ray)

Al realizar las comprobaciones de funcionamiento de los tres ensambladores para encontrar el más completo y adecuado para conseguir el genoma completo de seres vivos gracias a librerías con lecturas cortas. Hemos podido descubrir que el más completo en cuanto a la extracción de datos de salida es el ensamblador Ray. Cómo hemos visto, para el ensamblado Minia y para el ensamblador Velvet hemos tenido que utilizar herramientas complementarias (cómo QUAST, Ver Sección 2.7.7) para conseguir estadísticas de los resultados extraídos al realizar el proceso de ensamblaje. Y también, como hemos visto en el caso del ensamblador Velvet, hemos tenido que calcular de forma teórica el consumo de memoria RAM (Ver Sección 4.3.3) y hemos tenido que utilizar conocimientos básicos de programación para calcular el tiempo que ha tardado en ejecutar cada una de las tareas asociadas al problema.

Estos detalles, aunque no afectan al resultado final del proceso, pueden que nos haga perder tiempo y recursos con operaciones y desarrollos alternativos. En el caso del ensamblador Ray hemos podido encontrar todos los datos necesarios en los propios ficheros de salida del ensamblador, por lo que hemos ahorrado tiempo, recursos, y por lo tanto dinero al utilizarlo.

Cómo es lógico no nos podemos llevar por estas características para concluir la elección de un ensamblador para desarrollar nuestro problema, ya sea en nuestra empresa, en un instituto de investigación. Para ello tenemos que realizar un análisis de la calidad de los resultados de cada ensamblador. Cómo hemos durante la consecución de los resultados de los experimentos, hay un ensamblador que se planteaba como muy bueno y novedoso (ensamblador Minia) que no ha cumplido las expectativas. Es cierto que hemos conseguido resultados espectaculares de rendimiento respecto a consumos de memoria RAM y tiempo de ejecución. Pero la calidad del producto ensamblado se ha visto afectada, si bien hay que indicar que en algunos casos si da la talla, en cómputos generales está un paso por detrás del resto de ensambladores.

Puede darse el caso que para algunos tipos de proyectos o experimentos que no necesitemos demasiada calidad para conseguir los objetivos planteados. Pero en la mayoría de los casos, y como es lógico, sí que necesitaremos una calidad mínima para realizar nuestros proyectos con seguridad.

A parte de comparar los ensambladores por las características anteriormente citadas, también tenemos que comprar su rendimiento dependiendo a su clasificación por tipo de memoria. Cómo ya se ha indicado en este proyecto tenemos dos ensambladores de memoria compartida (Minia y Velvet) y uno de memoria distribuida (Ray).

Para la realización del proceso de ensamblado de genomas de seres vivos relativamente pequeños no nos hace falta una gran máquina ni sistemas de memoria distribuida. Con un sistema de memoria compartida, como Velvet o Minia, es suficiente. Pero cómo hemos podido ver en el proyecto, cuando aumentamos la complejidad y el tamaño de los genomas, los sistemas de memoria compartida se van quedando cortos para realizar todas las operaciones necesarias en un tiempo relativamente corto. Aunque en este proyecto no hemos tenido la capacidad de ensamblar genomas realmente grandes, (el más grande ha sido el cromosoma 14 del genoma humano) hemos podido apreciar la tendencia en los resultados. En genomas relativamente simples es suficiente con sistemas de memoria compartida, pero para genomas más complejos, (mamíferos, aves, etc.) es necesario y recomendable utilizar sistemas de memoria distribuida.

De todas formas el tamaño de los genomas es limitado y hará un momento en el que la tecnología pueda realizar el proceso de ensamblaje de forma más sencilla. El siguiente paso es el área de la metagenómica (Metagenómica). Es te campo se encarga de ensamblar genomas de seres vivos de un ecosistema completo sin saber a qué organismo pertenece. Por ejemplo las raíces de una planta. En ella aparte de encontrar en genoma de la propia planta podemos descubrir genomas de otros seres vivos que allí puedan convivir.

Frente a los aspectos puramente técnicos el proyecto, con la realización de este proyecto, también, hemos podido comprobar que el mundo de la informática puede estar ligado, y relacionado, con áreas muy distintas y lejanas. Gracias a esto podemos conseguir una polivalencia en el mercado laboral muy importante. Aunque el campo de la biología y bioinformática parecen campos experimentales, podemos observar que todos estos conocimientos y problemas pueden tener un carácter puramente empresarial, ya que muchas compañías necesitan realizar estas pruebas para conseguir, como por ejemplo, desarrollar un nuevo fármaco, una nueva vacuna o cualquier producto que pueda mejorar la calidad de vida de las personas.

Es previsible que las mejoras tecnológicas conviertan al proceso de ensamblaje de genes en un proceso más común que pueda ser utilizado en medicina como un medio de diagnóstico más.

# 7. LÍNEAS FUTURAS

Los avances tecnológicos de los últimos años han permitido una evolución muy importante en el tratamiento computacional y almacenaje de datos biológicos.

Como hemos visto a lo largo de este proyecto procesar, almacenar y utilizar estos datos de forma eficiente es una tarea muy compleja y laboriosa por lo que da lugar a que las tareas de secuenciación, anotación y ensamblaje de genes necesitan la utilización de los mejores equipos hardware y software para completarse con una calidad adecuada en el menor tiempo posible y utilizando la menor cantidad de recursos posibles.

Pese a esta evolución tecnológica en el campo de la bioinformática, hoy en día todavía es muy complicado desarrollar las tareas citadas anteriormente de manera eficaz. Como hemos podido ver muchas veces es necesario una cantidad ingente de memoria y de recursos para procesar y almacenar toda la información de procesos como la secuenciación del genoma humano. Pese a ello, desde el desarrollo de las primeras tecnologías en el campo de la informática hasta hoy ha habido una gran progresión mejorando la utilización de recursos y la velocidad de almacenaje y procesamiento de todos los datos biológicos. Por lo que observando la tendencia actual, en los próximos años, se observará mejoras continuas en los requisitos tecnológicos de los proyectos bio-informáticos. Un ejemplo claro, y que hemos podido ver en este proyecto, es el ensamblador Minia, un nuevo ensamblador que utiliza una nueva visión de los grajos de Bruijn para conseguir ensamblar genes utilizando muchos menos recursos que ensambladores análogos más antiguos.

Es previsible que esta tendencia de descubrimiento de nuevos algoritmos y tecnologías que mejoren de manera bastante destacable el campo completo de la bioinformática. También es bastante previsible debido a que es un campo relativamente joven, apenas unas décadas de vida, y en el campo de la tecnología, con el paso del tiempo se pueden observar uno avances que se podrían considerar imposibles.

En el futuro el campo de la bioinformática seguirá ganando importancia y ofreciendo avances, ventajas, soluciones y despejando las innumerables dudas que tiene el ser humano acerca de todos los aspectos biológicos que lo componen y que lo rodena.

Esta necesidad del ser humano de intentar explicar todo lo que le rodea y mejorar sus condiciones de vida harán necesario la mejora continua de todo el campo de la bioinformática, ya sea para explicar aspectos desconocidos de la vida humana, o descubrir curas para enfermedades que actualmente azotan a la raza humana.

En mi opinión por todos los aspectos que he ido explicando en este capítulo de líneas futuras, hacen que en el futuro el campo de la bioinformática siga ganando potencial. También es importante indicar que es posible que por la situación económica mundial estos avances no se puedan producir a un ritmo lógico y normal. Los numerosos recortes en I+D+I y en centros de investigación están provocando un estancamiento en los avances de este campo tan importante para áreas como son la biología-biotecnología y la informática.

De todas formas es previsible que la situación económica mejore algún día por lo que se puedan volver a destinar recursos a la investigación.

Un campo muy importante para la humanidad y que nunca ha debido de ser “abandonado” debido a las numerosas ventajas que ofrece al ser humano, para explicar aspectos de la vida que no tenían explicación y también para mejorar cada vez más las condiciones de vida.

Las reflexiones que he reflejado en este apartado hasta el momento, son reflexiones sobre líneas futuras en el aspecto investigativo de los componentes que conforman el proceso de ensamblaje de genes. Pero, también querría indicar Líneas Futuras en el campo técnico, que a mi parecer, es el campo más importante y que mejor se adapta y se acerca a lo que es en si la ingeniería informática.

Debido a que el proceso de ensamblaje de genes es un proceso complejo que necesita de cantidades ingentes de memoria, por lo menos a día de hoy, para conseguir unos resultados aceptables. En un futuro, y como he dicho antes, seguramente se mejoren estas características para producir ensambladores que consuman menos recursos. Pero hasta que esa mejora sea muy sustancial, yo creo que lo lógico es pensar que los investigadores intentarán realizar sus experimentos en las máquinas más potentes que tengan a su alcance.

En este aspecto puede atraer la idea del Cloud Computing, es una idea novedosa y que contrae muchas ventaja, no habría que tener un hardware específico para realizar las operaciones, simplemente tendríamos que alquilarlo. El problema de esta tecnología es que para experimentos de esta envergadura es necesaria una gran infraestructura de comunicaciones que se encargue de realizar los pasos de mensajes entre nodos con la mayor eficiencia posible para conseguir resultados óptimos.

Como resumen principal de este capítulo, creo, que es acertado indicar que este campo de investigación va a seguir aumentando su potencial e intentará disminuir sus costes para adaptarse a la situación económica actual. Aunque este aspecto, y como acabo de indicar, no tiene por qué influir en la calidad de las investigaciones ya que, en la actualidad, están surgiendo que permitirían una unión armoniosa de estas dos necesidades. Seguir investigando para mejorar nuestra calidad de vida e intentar un ahorro de costes para sortear la situación de crisis económica actual.

# 8. BIBLIOGRAFÍA


*Algoritmo Ensamblador Velvet*. (Febrero de 2013). Obtenido de http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2336801/

*Algoritmo de Dijkstra*. (s.f.). Recuperado el Febrero de 2013, de http://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html

*Cesvima*. (s.f.). Recuperado el Diciembre de 2012, de http://www.cesvima.upm.es/

Chikhi, R. (Enero de 2013). Workshop On Genomics. Pennsylvania State University.

*Cloud Computing Ensamblador Ray*. (s.f.). Recuperado el Mayo de 2013, de http://dskernel.blogspot.com.es/2012/05/computing-assembly-in-cloud.html

*Consumo de Memoria Velvet*. (s.f.). Recuperado el Abril de 2013, de http://biowulf.nih.gov/apps/velvet.html

*Definición Orgánulo*. (s.f.). Recuperado el Octubre de 2012, de http://www.biologia.edu.ar/cel_euca/celula4.htm

*De-Novo Genome Assembly*. (s.f.). Recuperado el Diciembre de 2012, de http://www.nature.com/nmeth/journal/v9/n4/full/nmeth.1935.html

*Documentación Magerit 2*. (s.f.). Recuperado el Diciembre de 2012, de http://docs.cesvima.upm.es/magerit-user-guide/es/

*Ensamblador Minia*. (s.f.). Recuperado el Enero de 2013, de http://minia.genouest.org/

*Ensamblador Ray*. (s.f.). Recuperado el Abril de 2013, de http://denovoassembler.sourceforge.net/

*Ensamblador Velvet*. (s.f.). Recuperado el Febrero de 2013, de http://www.ebi.ac.uk/~zerbino/velvet/

*Formato FASTA*. (s.f.). Recuperado el Diciembre de 2012, de http://www.bioperl.org/wiki/FASTA_sequence_format

*Formato FASTQ*. (s.f.). Recuperado el Diciembre de 2012, de http://maq.sourceforge.net/fastq.shtml

*Formatos de Secunecias*. (s.f.). Recuperado el Octubre de 2012, de http://bioinformatica.me/tag/formato-embl/

*Gage*. (s.f.). Recuperado el Abril de 2013, de http://gage.cbcb.umd.edu/data/index.html

Gurevich, A., Saveliev, V., Vyahhi, N., & Tesle, G. (2013). *QUAST: quality assessment tool for genome assemblies.*

*Manual Ensamblador Ray*. (s.f.). Recuperado el Abril de 2013, de http://denovoassembler.sourceforge.net/manual.html

*Manual Ensamblador Velvet*. (s.f.). Recuperado el Febrero de 2013, de http://www.ebi.ac.uk/~zerbino/velvet/Manual.pdf

*Metagenómica*. (s.f.). Recuperado el Abril de 2013, de http://www.oei.es/divulgacioncientifica/reportajes_416.htm

*Next Generation Sequencing Library*. (s.f.). Recuperado el Enero de 2013, de http://ngslib.i-med.ac.at/

*Software Putty*. (s.f.). Recuperado el Noviembre de 2012, de http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

*Software WinSCP*. (s.f.). Recuperado el Octubre de 2012, de http://winscp.net/eng/docs/lang:es

*Utilidad Slurm*. (s.f.). Recuperado el Diciembre de 2012, de http://www.ibm.com/developerworks/library/l-slurm-utility/

# ANEXOS

## Conexión remota a Magerit 2.

Como bien sabemos. Magerit 2 es uno de los supercomputadores más potentes de España y se encuentra situado en el CESVIMA dentro del campus de Montegancedo de la Universidad Politécnica de Madrid. A veces puede ser necesario realizar la conexión al supercomputador a distancia para comprobar la ejecución de las tareas o, simplemente, para mandar nuevas tareas sin necesidad de trasladarse a las instalaciones de manera física.

Para conectarnos de forma remota tenemos varias aplicaciones que nos facilitan mucho la tarea. A continuación vamos a especificar las aplicaciones utilizadas tanto en Windows como en Linux.

En este proyecto ha predominado la utilización de Linux para realizar la conexión con el supercomputador. Pero a veces ha tenido que realizarse desde Windows al no tener ordenadores con Linux instalado.

### Conexión remota desde Windows

Windows es el sistema operativo que podemos encontrar en el casi 100% de los ordenadores de todo el mundo, y quizás, el más fácil de utilizar para el usuario de a pie. En este sistema operativo disponemos de dos aplicaciones complementarias que nos permiten, tanto conectarnos remotamente al supercomputador y comenzar a realizar nuestro trabajo, como enviar cualquier tipo de archivo que necesitemos trasladar de nuestro ordenador local al supercomputador.

- **Putty** (Software Putty)

  Software básico para realizar conexiones telnet remotas contra otro servidor u ordenador. Su funcionamiento es bastante sencillo e introduciendo los datos básicos de conexión más el login y el password no permite conectarnos sin problemas a cualquier servidor. La interfaz gráfica es bastante amigable y una vez conectados nos simula el terminal de Linux, o del propio servidor para realizar nuestro trabajo.

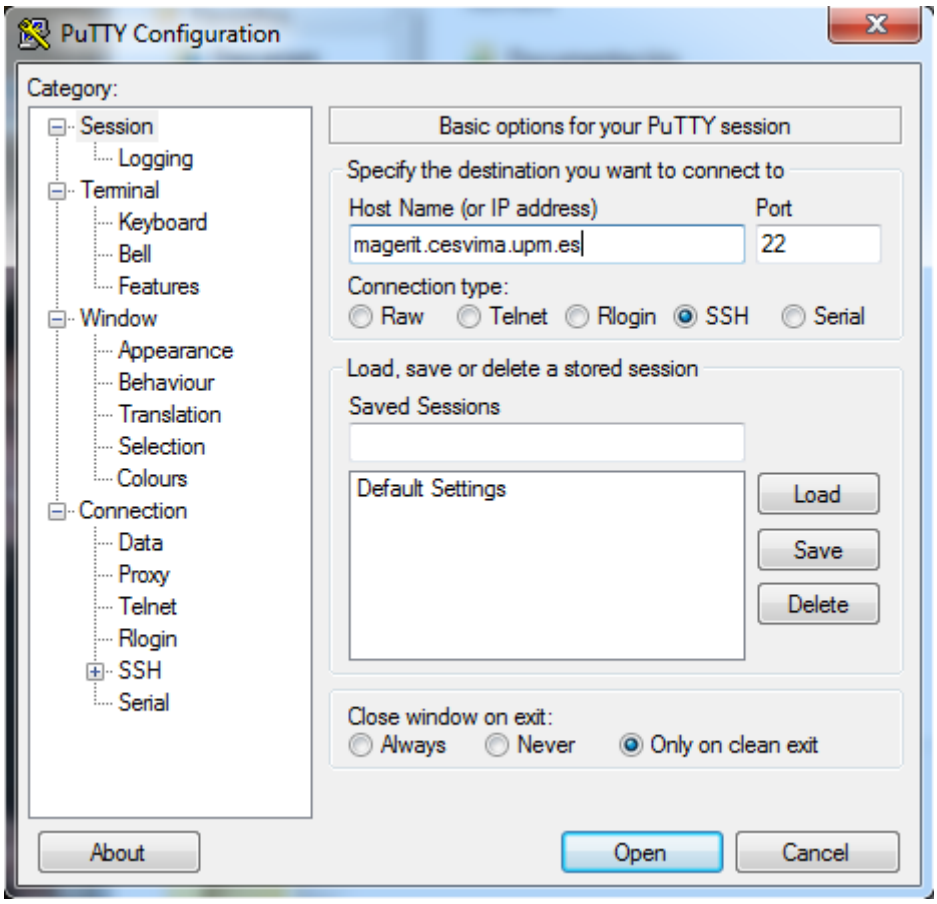



Ilustración 37 Configuración Putty

Una vez introducida la ruta de Magerit. Hacemos click en el botón “open” para poder introducir nuestro user y nuestro password

Finalmente, después de realizar todo el proceso anterior ya estaremos dentro de nuestra cuenta de usuario de Magerit. En ella podremos ver un directorio en el cual podremos realizar todas las tareas necesarias para ejecutar nuestros trabajos con normalidad.

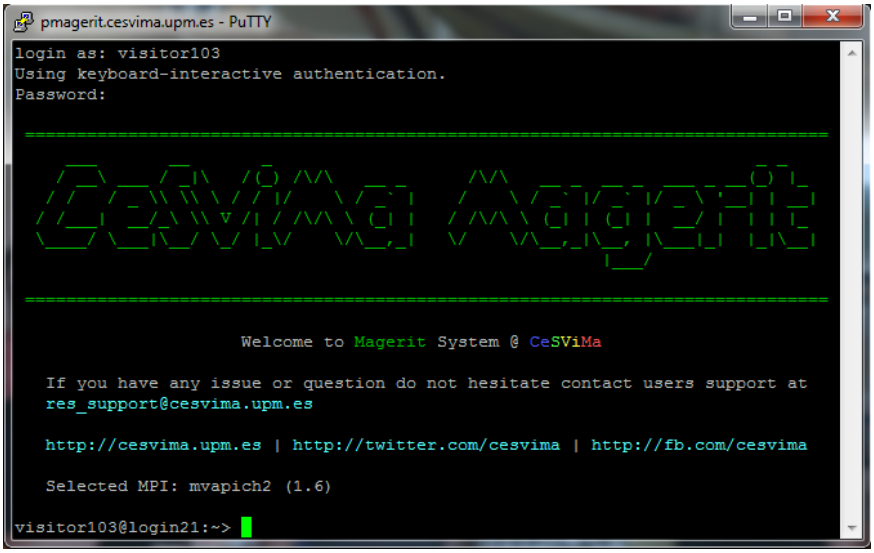



Ilustración 38 Pantalla de bienvenida Magerit en Windows.

- **WinSCP** (Software WinSCP)

Software complementario al software Putty encargado de transferir archivos entre el ordenador local y el ordenador que estamos controlando de forma remota. Todo el envío de información se realiza mediante una interfaz gráfica que simula las carpetas de ambos ordenadores y da la sensación de que estamos traspasando la información de una carpeta a otra sin más.

La manera de iniciar sesión es análoga a la utilizada en el software Putty. Al ser un software corriendo bajo Windows, dispondremos de una interfaz gráfica para realizar estas tareas.

A diferencia del caso anterior, no será suficiente con introducir nuestro user y password. También tendremos que modificar la opción denominada "Archivo de Protocolo". En dicha opción tendremos que poner como protocolo utilizado el protocolo SCP (Secury Copy). El protocolo SCP se basa en el protocolo SSH para realizar en envío de la información.

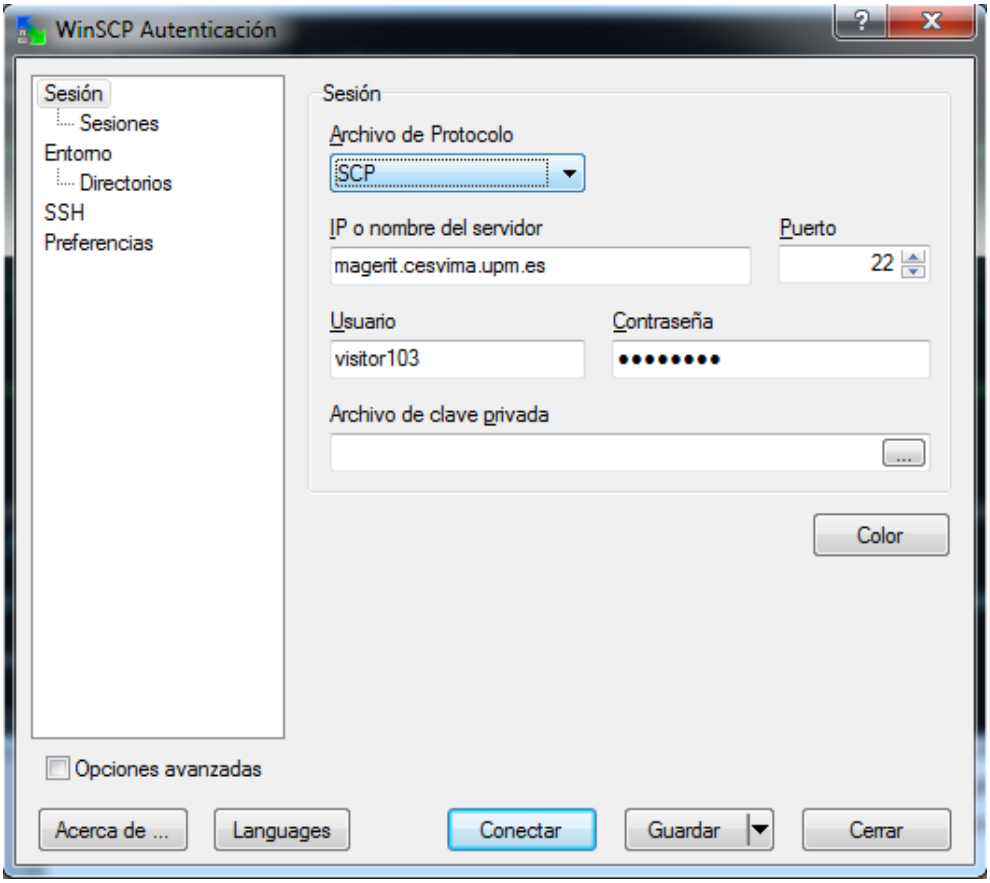



Ilustración 39 Configuración WinSCP

Una vez que hemos introducido todos los datos necesarios para realizar la conexión hacemos click sobre el botón "Conectar" Una vez realizada esta acción nos desaparecerá la ventana actual y nos aparecerá una nueva ventana que nos irá mostrando todos los datos relevantes del proceso de conexión. Si todo va bien dicha ventana desaparecerá al cabo de unos instantes.

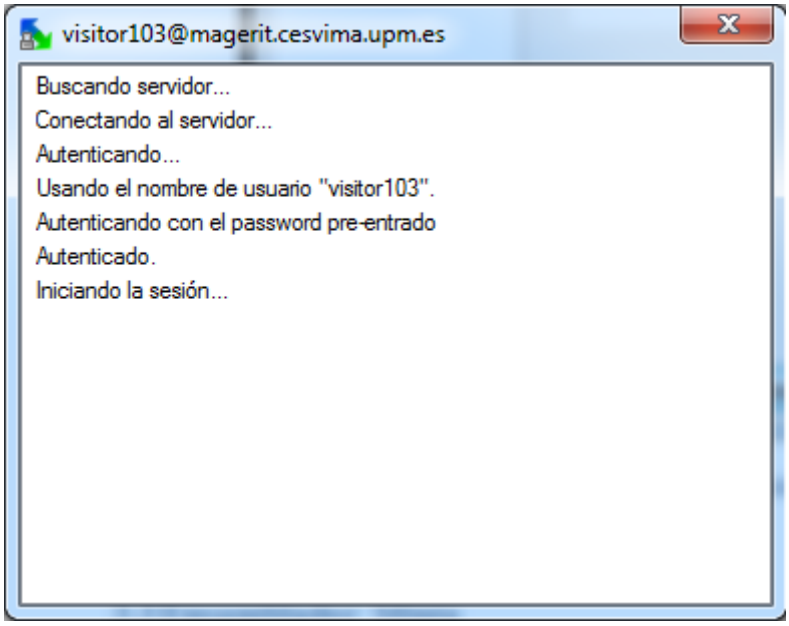



Ilustración 40 Proceso de Login WinSCP

Finalmente, nos aparecerá la ventana principal de la aplicación. En dicha ventana tendremos visibles todos los directorios presentes tanto, localmente como remotamente. En la parte izquierda de la aplicación se encontrarían todos los directorios almacenados en el ordenador en el cual estamos realizando la conexión y a la izquierda todos los directorios presentes en Magerit.

Simplemente tendremos que valernos de la interfaz gráfica para realizar el envío de la información deseada.

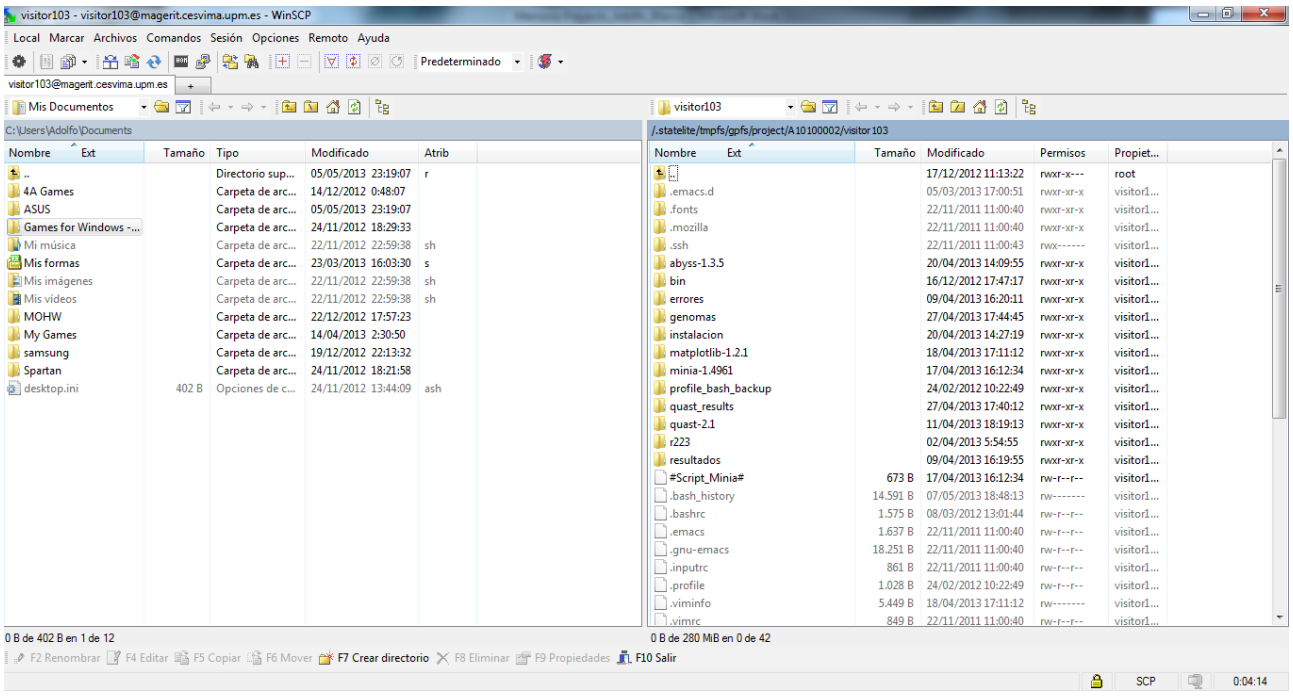



Ilustración 41 Pantalla principal WinSCP

### Conexión remota desde Linux

En la sección anterior hemos visto cómo realizar la conexión y el envío de información en ordenadores con sistema operativo Windows. En esta sección vamos a ver el proceso análogo para sistemas operativos Linux. En este caso, concretamente, utilizamos una distribución Ubuntu en su versión 12.10 de 64 bits.

Este sistema operativo será el utilizado a partir de ahora para explicar todos los procesos relevantes que se sucedan a lo largo del proyecto. Se ha utilizado este sistema operativo debido a la mejor usabilidad de conexiones remotas por parte de distribuciones Linux, que por sistemas operativos Windows.

Para realizar todas las tareas necesarias durante el desarrollo del proyecto vamos a utilizar la Terminal, o Shell que viene instalada por defecto en todas las distribuciones de Linux, en todas sus versiones.

La terminal, no es más que un intérprete de comandos que nos permite realizar todas las opciones deseadas en nuestra distribución Linux, es la interfaz de usuario tradicional.

La particularidad de esta interfaz de usuario es que podremos utilizar todos los protocolos utilizados para la conexión (SSH), y el envío de datos al supercomputador (SCP), sin necesidad de instalar software alternativo en nuestro ordenador.

- **Terminal**

Para ejecutar una terminal en una distribución Ubuntu tendremos que buscar su nombre en la lista de programas de nuestro ordenador o hacer click en el icono de Terminal  .

Al hacer click en una de las opciones anteriores nos aparecerá una pantalla negra, similar a la que pudimos observar en el software Putty al realizar la conexión con el supercomputador de forma satisfactoria.

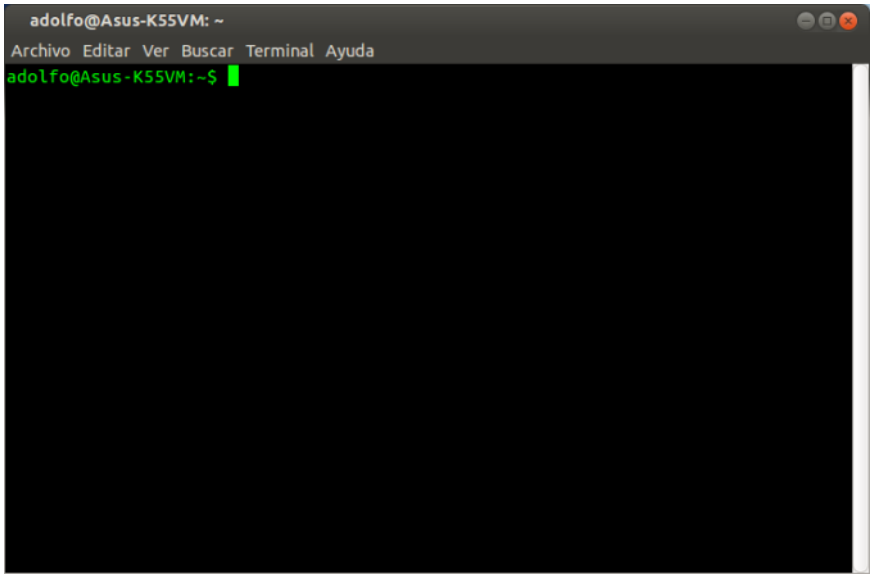



Ilustración 42 Terminal Linux

En la termina procederemos a introducir todos los comandos, o mandatos que deseemos para realizar nuestra tarea. Si tenemos alguna duda del funcionamiento de algún comando en Linux podremos consultar, siempre que queramos, el manual de usuario del comando en la Terminal con el siguiente comando:

- man Comando

En primer lugar para realizar una conexión remota con el supercomputador Magerit tendremos que utilizar el comando SSH. Concretamente lo utilizaremos de la siguiente manera:

- ssh user@host

Siendo user nuestro usuario de conexión normal en Magerit y host la dirección de conexión.


Ilustración 43 Comando SSH Linux

Una vez que hemos introducido el comando correspondiente presionamos el botón intro. Al realizar esta acción nos pedirá nuestra contraseña para confirmar la autenticación y, una vez introducida, podremos observar la pantalla de bienvenida de Magerit.

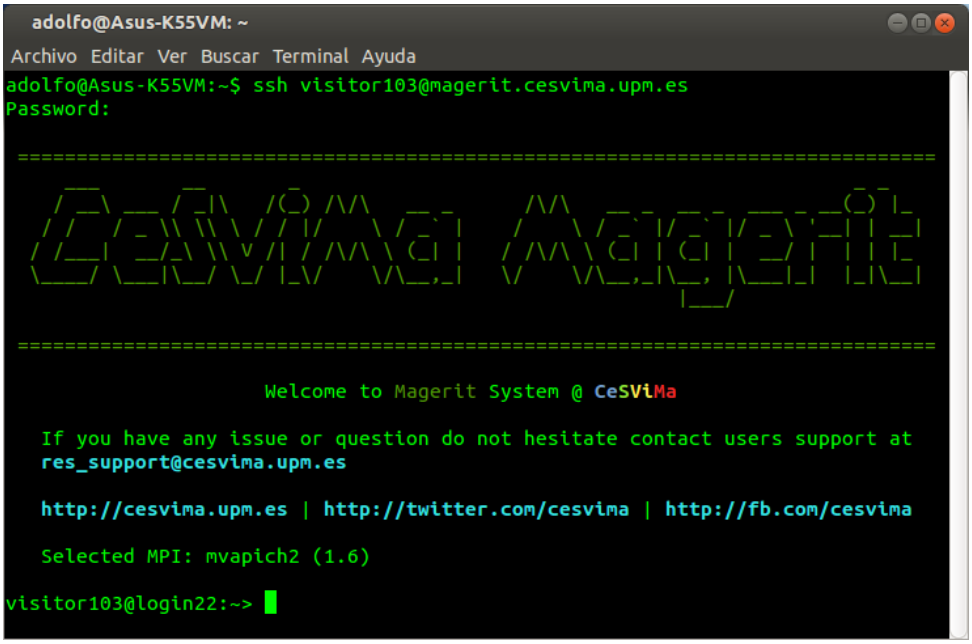



Ilustración 44 Pantalla de Bienvenida Magerit en Linux

En el caso de Linux, para transferir archivos al supercomputador, podremos utilizar la terminal directamente. Para ello tendremos que introducir el comando correspondiente del protocolo SCP. Cabe destacar que tenemos que ejecutar dicho comando en la sesión local y no en la sesión del supercomputador.

- **Enviar datos desde nuestra máquina al supercomputador:**
  - scp ArchivoOrigen usuario@host:directorio/ArchivoDestino
- **Enviar datos del supercomputador a nuestra máquina:**
  - scp usuario@host:directorio/ArchivoOrigen ArchivoDestino

Si por alguna razón el sistema anterior no resulta cómodo para transferir archivos, las distribuciones Linux también ofrecen Software alternativo, ya sea instalado de serie, o instalándolo nosotros aparte. Dicho software puede ser Filezilla (software de terceros) o la utilidad que nos ofrece la distribución Linux para realizar el proceso de copia de datos remota a través de una interfaz gráfica. Dicha aplicación se denomina Connect-Server y es la que explicaremos en este apartado. Para ejecutar dicha aplicación tendremos que buscarla en el directorio de aplicaciones.

- **Connect Server**

La aplicación Connect Server es muy parecida a la aplicación WinSCP que pudimos observar en la sección anterior. Para realizar la conexión tenemos que introducir la dirección del Host Magerit, nuestro user y nuestro password. También se recomienda introducir la ruta de nuestra carpeta HOME de Magerit en la opción "Carpeta:" debido a que la aplicación Connect Server nos conecta directamente en el directorio raíz de Magerit. Si nos sabemos cuál es nuestra carpeta HOME podremos comprobarlo en nuestra terminal, una vez iniciada sesión en Magerit, con el comando pwd.


Ilustración 45 Aplicación Connect Server Ubuntu

Una vez que hemos introducido todos los datos de manera correcta nos aparecerá una nueva ventana, que simula ser una ventana local, donde podremos observar todo el contenido de nuestra carpeta HOME. También podemos navegar por el directorio de carpetas de Magerit como si fuera una carpeta completamente local. A partir de ahí podemos copiar y exportar datos como si de una carpeta local se tratara.

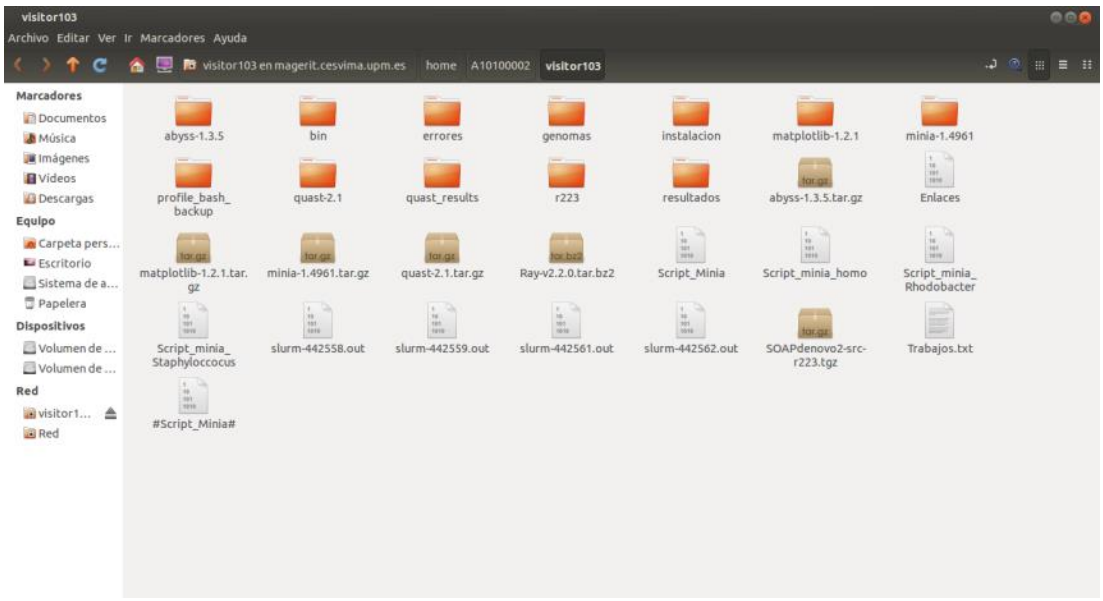



Ilustración 46 Carpeta Home de Magerit en Linux

### Descarga de Datos en Magerit

Cómo puede parecer lógico para realizar todas las comprobaciones que queremos realizar en este proyecto tendremos que transferir los archivos de los genomas y el código fuente de los ensambladores al supercomputador para utilizarlos según nuestras necesidades.

Por lo que hemos estado viendo en estas secciones ya dispondríamos de una interfaz para realizar esta tarea. Podríamos realizar la descarga de dicha información en nuestra máquina local y posteriormente transferirla a Magerit por alguno de los métodos que hemos podido ver anteriormente.

Durante la explicación del proyecto, en la introducción, hemos descrito que los ensambladores se utilizan para procesamiento masivo de datos. Dicho procesamiento masivo necesita de grandes cantidades de información de lecturas de genomas para realizar los experimentos. La magnitud del tamaño de dicha información puede ser de varios Gb, y debido a las limitaciones que tenemos en nuestro país con las velocidades de conexión a internet no parece muy buena idea descargar todo el contenido en local y luego transferirlo al supercomputador ya que duplicaríamos (como mínimo, ya que la velocidad de subida suele ser bastante menor a la velocidad de bajada.) el tiempo de descarga de toda la información necesaria.

Para solucionar este problema podríamos descargar directamente la información en el supercomputador.

Para realizar este proceso utilizaremos el siguiente comando (dentro de la sesión en Magerit):

- wget –b URL de Descarga

Siendo la opción –b opcional ya que, lo que realiza esta opción es la descarga de la información en segundo plano guardando la información de la descarga en el archivo wget-log en vez de mostrarla en todo momento por pantalla.