TÍTULO

ESTUDIO Y EVALUACIÓN DE ALGORITMOS Y PROGRAMAS DE ENSAMBLAJE DE SECUENCIAS

AUTOR

Miguel Hormigo Ruiz

Esta edición electrónica ha sido realizada en 2011

Director Manuel Gonzalo Claros Llanos

Curso II Máster Oficial en Bioinformática

ISBN 978-84-7993-943-4

Universidad Internacional de Andalucía, 2011

Preparado:
Código: MMHR V1/10
Fecha: 8/04/2011
Versión: 4

Estudio y Evaluación de Algoritmos y Programas de Ensamblaje de Secuencias

II Máster Oficial en Bioinformática (Universidad Internacional de Andalucía)

Autor: Miguel Hormigo Ruiz
Director del Proyecto: Manuel Gonzalo Claros Llanos

ESTA PÁGINA SE HA DEJADO EN BLANCO INTENCIONADAMENTE.

HOJA DE ESTADO DEL DOCUMENTO

Versión	Fecha	Págs.	Procesador	Cambios
1	17/01/2011	40	Word 2000 español	
2	24/01/2011	49	Word 2000 español	
4	6/03/2001	57	Word 2000 español	
4	8/04/2011	90	Word 2000 español	

ÍNDICE

LISTA FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS

BAC	(*Bacterial Artificial Chromosome*)	Cromosoma artificial bacteriano
WGS	(*Whole-genome Sequencing*)	Secuenciación completa del genoma
EST	(*Expression Sequence Tag*)	Marcador de secuencia expresada
NGS	(*Next-generation sequencing*)	Secuenciación de nueva generación
ADNc		(ADN Complementario)
SNP	(Single-nucleotide Polymorphism)	Polimorfismo de nucleótido simple
QV	(quality value)	Indicador de calidad
nt		(nucleótido)
BWT	(Burrows-Wheeler Transform)	Transformación de Burrows-Wheeler
pb		pares de bases

GLOSARIO DE TÉRMINOS

- **Lectura** (*read*): pequeño segmento de bases consecutivas de ADN identificado mediante un instrumento de secuenciación.
- **Extremos apareados** (*paired-ends*, *mate-pairs* o *paired-end reads*): pares de lecturas de los dos extremos de un solo fragmento clonado por lo que la distancia entre ellas es conocida de manera aproximada.
- ***Contig***: palabra apocopada del término inglés *contiguous* porque definen una secuencia continua de ADN reconstruido a partir de un conjunto de lecturas.
- ***Supercontig*** o ***metacontig*** (*Scaffold*): grupos de *contigs* que se pueden ordenar y orientar haciendo uso de extremos apareados.
- ***Unitig****:* es un *contig* formado por solapamiento no ambiguo de secuencias únicas.
- **Cobertura** (*coverage*, *read depth, coverage depth*): sobremuestreo del genoma para la generación de lecturas.

1. RESUMEN

1.1. INTRODUCCIÓN

Los mecanismos de secuenciación actuales general gran cantidad de información en pequeños fragmentos desordenados que hay que unir correctamente para obtener la secuencia original.

Las técnicas utilizadas para resolver el problema se basan principalmente en dos líneas de trabajo complementarias entre sí:

- La búsqueda de un modelo matemático mediante el que controlar una solución óptima: en este sentido, las aproximaciones se basan en utilizar combinatoria y teoría de grafos, tales como algoritmos de superposición-trazado-consenso (basado en grafos de intersección) o algoritmos de **EULER** (basados en el concepto de «grafo de De Bruijn» [20]).
- El uso de hipótesis que reduzcan la complejidad del problema, tales como que dos fragmentos son adyacentes si se solapan de alguna manera, es decir, si la parte final de una secuencia coincide con la inicial de la siguiente. Estas hipótesis, a su vez, no tienen por qué ser totalmente válidas ya que, por ejemplo, pueden existir secuencias repetitivas y también debido a que la superposición parcial de dos fragmentos puede deberse simplemente al azar y no a que dichos fragmentos sean realmente adyacentes en la secuencia original. Además, la secuencia original no aparece cubierta en su totalidad por el conjunto de fragmentos y, por otra parte, se producen errores experimentales en la lectura de cada fragmento. Estos errores hacen que dar con la secuencia original correcta sea una tarea muy compleja.

Estos principios y la evolución constante de la investigación del ensamblaje de secuencias junto con el avance de la industria y las técnicas de NGS hacen necesario realizar un estudio de la situación actual de los algoritmos básicos de ensamblaje de secuencias, los programas base que los utilizan, la situación actual de dichos programas y la evolución de las técnicas de ensamblaje de secuencias en los próximos años.

1.1. OBJETIVOS

Los objetivos del proyecto son los siguientes:

- Realizar un estudio sobre los métodos de ensamblaje, teniendo en cuenta su componente teórica (algoritmos y heurísticas utilizadas) y su desarrollo en programas de ensamblaje (origen, investigación, desarrollo, referencias iniciales y actuales, y futuro del método).
- Evaluar desde un punto de vista teórico los distintos programas de ensamblaje de secuencias, observando tanto los comerciales como los de libre distribución, clasificando su desarrollo, construcción, objetivos, uso, aplicación, referencias, funcionamiento (fiabilidad, rendimiento, consumos, …) y previendo futuro del programa.
- Instalar y evaluar el funcionamiento de los programas más habituales de ensamblaje de secuencias, teniendo en cuenta su acceso a los mismos.

1.2. MÉTODOS Y FASES DE TRABAJO

El método de trabajo será principalmente la recopilación de información sobre los algoritmos y programas de ensamblaje de secuencias y la instalación de los programas que se puedan descargar, instalar y ejecutar en plataformas Linux y Microsoft o bien en entornos Web.

Las fases serán las siguientes:

1. **Introducción al ensamblaje de secuencias**: origen, algoritmos, técnicas, precursores, estado del arte y evolución.
2. **Métodos de ensamblaje**: clasificación, marco teórico, aplicaciones prácticas, referencias, algoritmos utilizados, objetivos de los métodos (tipo de secuencias que ensambla, longitud y heurísticas).
3. **Evaluación de los programas de ensamblaje**: evaluación de los programas más conocidos de ensamblaje de secuencias mediante la instalación (cuando sea posible) y prueba, con el objetivo claro de contrastar los distintos programas (clasificación, objetivo, ventajas, inconvenientes, rendimiento, usabilidad, SO, mercado, uso, historia y futuro del programa)
4. **Conclusión** del estudio sobre el estado del arte de los algoritmos y programas de ensamblaje de secuencias con un énfasis en la evaluación de los métodos durante los últimos años y la futura evaluación de los mismos en los próximos años, teniendo en cuenta las nuevas técnicas de NGS y nuevas tecnologías emergentes.

2. INTRODUCCIÓN AL ENSAMBLAJE DE SECUENCIAS

2.1. DEFINICIÓN

Podemos definir el ensamblaje de secuencias en Bioinformática como la superposición de varios fragmentos de una secuencia de ADN para alinearlos y reconstruir la secuencia original. La necesidad de realizar un ensamblaje de secuencias viene dada porque actualmente no es posible secuenciar un genoma completo mediante una sola lectura, sino que es necesario realizarlo mediante pequeños trozos de la secuencia original con una longitud de entre 20 y 1000 bases, según la tecnología que se utilice. Normalmente, los fragmentos cortos, llamados «lecturas» o *reads* vienen dados por una secuenciación genómica por perdigonada (Figura 2) o una transcripción de genes (ESTs) [147].

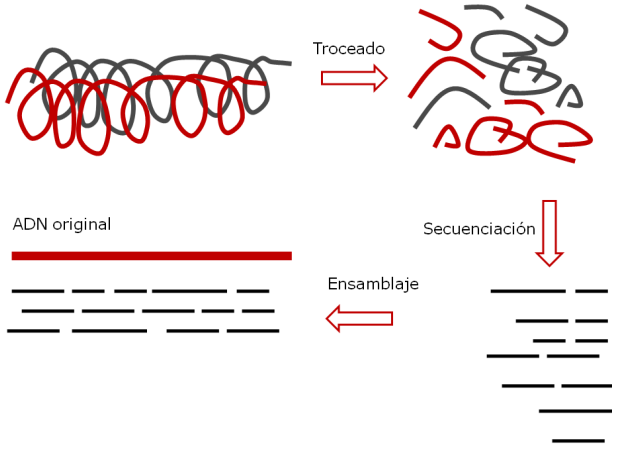


Figura 2 Secuenciación por perdigonada y ensamblaje posterior de los fragmentos o lecturas

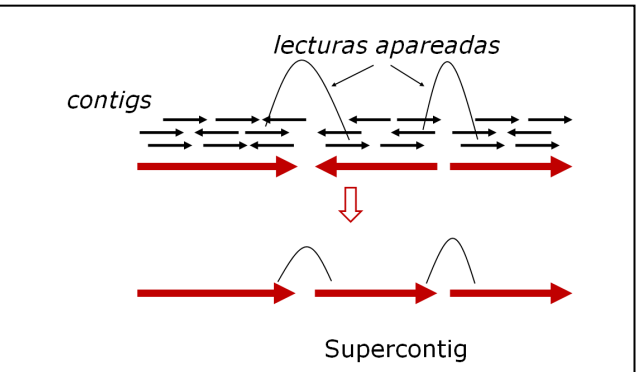


Figura 1 Formacion de *supercontigs* a partir de los *contigs* y las lecturas apareadas

Para realizar el proceso de ensamblaje, estas lecturas se agrupan en *contigs* y estos *contigs* a su vez se ensamblan llenando los huecos que pueda haber entre ellos (debido a la existencia de secuencias de nucleótidos que se repiten muchas veces en el ADN) mediante análisis de la coincidencia de los extremos de dos *contigs* con los de algún inserto que tengan en común (método de los «extremos apareados» (*paired-ends*, *mate-pairs* o *paired-end reads*), produciendo esqueletos (*scaffolds*), también llamados *supercontigs* o *metacontigs* (Figura 1). El alineamiento múltiple de secuencias en un *contig* produce la secuencia consenso. Los *supercontigs* definen el orden de los *contigs*, su orientación y el tamaño de los huecos (*gaps*) entre los *contigs*. La topología del *supercontig* puede ser, a su vez, una secuencia simple (*path*) o una red. El formato más utilizado actualmente para el ensamblaje de secuencias es el ACE, aunque existen otros como AFG, MAQ, SOAP2, QUAL, SAM y BAM (algunos de ellos derivados del propio programa) donde los *contigs* consenso se pueden representar por cadenas de caracteres A, C, G y T más otros caracteres que pueden tener un significado especial. La secuencia esqueleto consenso suele contener N en los huecos entre los *contigs* y el número de N indica normalmente la longitud estimada de los huecos entre finales emparejados.

El resultado de los ensambladores se puede medir por el tamaño y la precisión de sus *contigs* y *supercontigs*. 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. El *contig* 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 genoma está contenido en *contigs* de tamaño mayor o igual que X. Los estadísticos N50 no son comparables entre ensamblajes a menos que se calculen usando el mismo valor de sumatorio de longitudes.

2.2. PROBLEMAS PRINCIPALES

Las tecnologías de secuenciación comparten una limitación fundamental: las lecturas son mucho más cortas que el genoma del que proceden. Actualmente, las tecnologías WGS (*Whole-Genome Sequencing*) intentan superar esta limitación con la realización de un sobremuestreo o cobertura del genoma, la generación de lecturas cortas de posiciones aleatorias y, posteriormente, con la reconstrucción de la secuencia original mediante el programa de ensamblaje, pero este enfoque no asegura que los datos secuenciados sean completos por lo que es necesario tener en cuenta que pueden existir huecos no secuenciados. Igualmente los sistemas de secuenciación pueden generar errores de transcripción que se intentan resolver mediante la cobertura

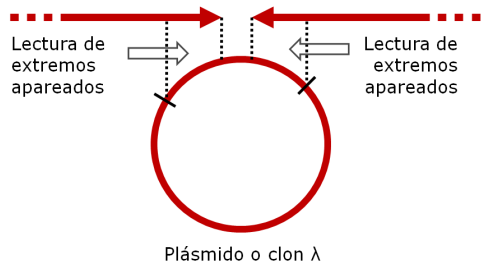


Figura 4 Los huecos de las secuencias se localizan entre lecturas de extremos apareados (un par de minisecuencias de los dos extremos de un solo fragmento clonado) y, por lo tanto, se pueden cerrar por secuenciación adicional del ADN clonado

Existen otros problemas a resolver que vienen determinados porque el ADN contiene secciones repetidas llamadas «repeticiones» que han de tratarse correctamente para no generar errores (Figura 5, Figura 3, Figura 6).

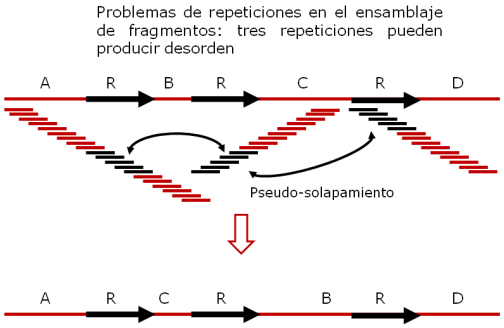


Figura 3 Problemas de repeticiones en el ensamblaje de secuencias. Tres repeticiones pueden producir desorden

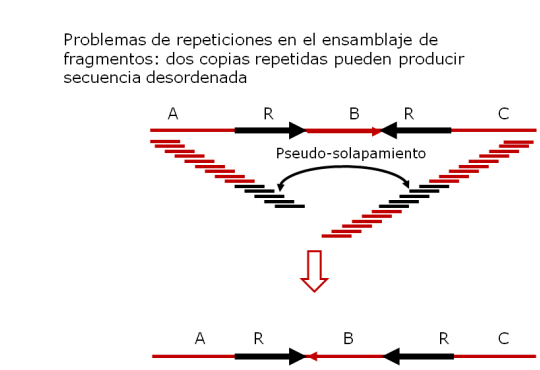


Figura 5 Problemas de repeticiones en el ensamblaje de secuencias. Dos copias repetidas pueden producir una secuencia desorodenada

Las repeticiones son complejas de resolver debido a que no es fácil distinguir entre secuencias repetidas o solapamientos. La forma heurística de solucionar este problema es correlacionar las lecturas mediante patrones de diferentes bases e intentar ampliar la cobertura de la secuenciación, lo cual puede

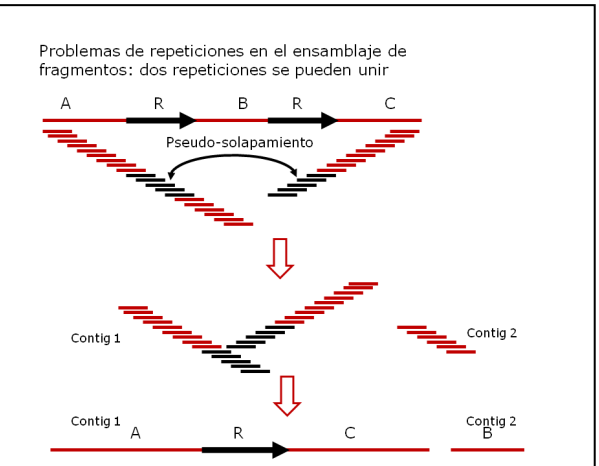


Figura 6 Problemas de repeticiones en el ensamblaje de secuencias. Dos repeticiones se pueden unir

inducir errores de secuenciación. Para repeticiones cuya fidelidad excede la de las lecturas, la resolución de las repeticiones depende de la longitud de las lecturas, es decir, si una lectura abarca una repetición con una única secuencia en cualquier posición de la repetición. Las repeticiones que resulten más largas que las lecturas se pueden resolver con la extensión de los extremos emparejados (Figura 4), pero el análisis es complicado y la resolución completa requiere normalmente dos fuentes: pares que abarquen la repetición en cada extremo con una única secuencia y pares que coincidan exactamente con un extremo en la repetición. Los límites para la resolución de repeticiones se pueden configurar para genomas completos mediante suposiciones. El problema de las «repeticiones» no fue resuelto correctamente hasta la propuesta de **EULER** [92] en 2001 donde se propone, como alternativa al enfoque de solapamiento por consenso, reducir el problema de ensamblaje de secuencias a una variación del problema clásico del camino euleriano. Este enfoque tiene una complejidad computacional polinómica frente a la complejidad exponencial de otras aproximaciones. Actualmente las plataformas de secuenciación NGS generan lecturas más cortas que a su vez generan repeticiones con más probabilidad pero a cambio se obtiene una cobertura mayor que incrementa la posibilidad de abarcar repeticiones cortas y que condiciona el uso de nuevos enfoques para el correcto ensamblaje de las lecturas.

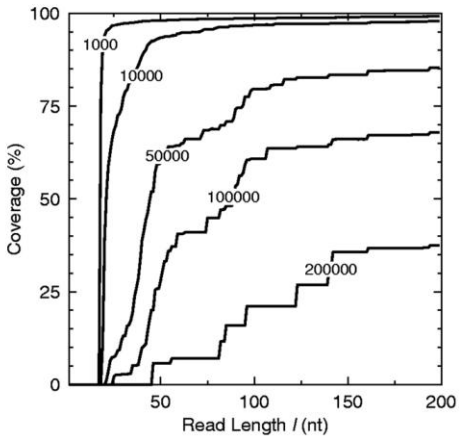


Figura 7 Porcentaje del genoma [106] de la *E. Coli* cubierto por *contigs* mayores que una longitud umbral en función de la longitud de las lecturas. Whiteford N et al. Nucl. Acids Res. 2005;33:e171-e171. ©Oxford University Press

El cálculo de la longitud óptima de las lecturas depende de varios factores como pueden ser la cobertura, el tamaño de los *contigs* generados (lo cual ayuda al proceso de ensamblaje) y el genoma a ensamblar. Ya en 2005 se trató este tema [82, 106] y con lecturas de 20nt (Figura 7) podemos reensamblar el 98% del genoma de la bacteria E. Coli con *contigs* de más de 100nt o el 91% con *contigs* de más de 1.000nt pero sólo se cubre el 10% con *contigs* mayores de 10.000nt. Si aumentamos el tamaño de la lectura a 30nt mejoramos la cobertura a un 75% con *contigs* mayores de 10.000nt. Esta relación entre la cobertura y el número de *contigs* que pueden ser reconstruidos mediante un programa de ensamblaje teórico fue estudiado por Eric Lander y Michael Waterman [88] con lo que podemos conocer *a priori* la cobertura necesaria para el ensamblaje teórico de un genoma concreto. El modelo Lander-Waterman parte de la asunción de que las lecturas están uniformemente distribuidas en la secuenciación y propone que el número aproximado de *contigs* necesarios $m = N*e^{-C}$. Algunos ejemplos de aplicación de este modelo dan los siguientes datos de cobertura:

- Drosophila: C=14x
- Humano: C>7x
- Delfín: C=2.59x
- Lemur: C=1.93x

Sin embargo, estas estimaciones son teóricas y en experimentos concretos se producen ciertas divergencias [118].

Actualmente muchos procesos de secuenciación de genomas son muy complejos e incluso se realizan de manera distribuida entre varios centros de investigación, por lo que se hace necesaria la colaboración [1] para unir los distintos ensamblajes. Haciendo un poco de historia, los primeros ensambladores comenzaron a aparecer en la década de los 80 y principios de los 90 como variantes de los programas de alineamiento de secuencias para alinear las grandes cantidades de fragmentos que generaban automáticamente los secuenciadores de ADN (método de Sanger). A medida que los organismos secuenciados crecieron en tamaño y complejidad (desde pequeños virus o plásmidos a bacterias y finalmente a seres eucarióticos), los programas de ensamblaje han ido necesitando estrategias más sofisticadas para manipular:

- Grandes cantidades de datos (terabytes) a procesar en *clusters* de computación.
- Secuencias idénticas o casi idénticas (conocidas como repeticiones) que pueden, en el peor de los casos, incrementar exponencialmente el tiempo y la complejidad de los algoritmos.
- Errores en los fragmentos producidos por los instrumentos de secuenciación, que pueden provocar errores en el ensamblaje.

Cuando los científicos se enfrentaron con el objetivo de ensamblar los primeros genomas eucarióticos, entre ellos el de la levadura (*Saccharomyces cerevisiae*) en 1996, la mosca de la fruta (*Drosphila melanogaster*) en el año 2000 y el genoma humano un año más tarde, se desarrollaron ensambladores como el **CELERA ASSEMBLER** [37, 79] y **ARACHNE** [12] que eran capaces de manipular genomas de 100 a 300 millones de pares de bases. Como consecuencia de estos esfuerzos, otros grupos, pertenecientes principalmente a los principales centros de secuenciación de genomas, siguen desarrollando ensambladores a gran escala, incluso basados en software de código abierto, como pueden ser el conocido como **AMOS** [40], el cual pretende llevar todas las innovaciones tecnológicas del ensamblaje de secuencias hacia el software libre.

Por otro lado, tenemos que tener en cuenta que el ensamblaje de EST difiere del genoma en varios puntos. Las secuencias que se ensamblan proceden de fragmentos de los mRNA que se expresan en una célula, por lo que representan sólo un subconjunto del genoma completo. A primera vista, los problemas subyacentes y algoritmos difieren entre el ensamblaje de un genoma y de las EST. A menudo, los genomas tienen grandes cantidades de secuencias repetidas que no aparecerán en las EST, precisamente porque representan porciones de genes expresados. Por otro lado, las células tienden a tener un conjunto de genes expresados constantemente, otros sobreexpresados y otros subexpresados, lo cual conduce nuevamente al problema de que aparecen secuencias similares junto a otras muy escasas en el conjunto de datos que se tienen que ensamblar.

3. ENSAMBLAJE DE SECUENCIAS DE NGS (*NEXT GENERATION SEQUENCING*)

3.1. INTRODUCCIÓN

Los avances técnicos tales como el desarrollo de la clonación molecular, la secuenciación Sanger, la PCR y las micromatrices (*microarrays*) de oligonucleótidos, son una clave importante a tener en cuenta para la capacidad actual de secuenciar, anotar y estudiar genomas completos de organismos [97]. En los últimos años se han producido avances muy importantes en las plataformas denominadas NGS (*Next Generation Sequencing*) que se están comercializando. La tecnología, gracias a trabajar a muy pequeña escala, junto con la aceptación entusiasta de la comunidad científica y la posibilidad en la precisión y la longitud de las lecturas (*reads*), sugieren que estas tecnologías están destinadas a producir un gran impacto en las ciencias biológicas relacionadas con la genómica y la post-genómica, sobre todo porque está sustituyendo a la técnica tradicional de secuenciación de Sanger. Sin embargo, al igual que el análisis de los datos de micromatrices y el ensamblaje y anotación de secuencias de genomas completos de los datos de la secuenciación convencional, la gestión y el análisis de los datos de NGS requieren del desarrollo de herramientas informáticas que sean capaces de ensamblar, identificar e interpretar cantidades ingentes de datos de secuencias de nucleótidos extremadamente cortas. En este documento proporcionamos una amplia visión de los avances bioinformáticos que se están introduciendo y desarrollando para numerosas aplicaciones genómicas y de genómica funcional basadas en NGS.

El desarrollo de estas tecnologías de secuenciación [28] masiva en paralelo ha surgido de distintos avances tecnológicos, entre ellos los que se han producido en el campo de la nanotecnología, mediante la cual se dispone de instrumentos ópticos capaces de detectar y diferenciar millones de fuentes de luz o fluorescencia en la superficie de pequeños cristales, y, cómo no, de la aplicación de los principios clásicos de biología molecular al problema de la secuenciación. Otra consideración importante es que, en el contexto de una secuencia genómica disponible, muchos problemas (tales como la identificación de SNP) no requieren la generación de lecturas más largas, debido a que, con mucha probabilidad, las «palabras» de más de 25 o 30 nt no aparecen más de una vez en la secuencia completa del genoma, lo que permite, en la mayoría de los casos, una asignación precisa de, incluso, las lecturas más cortas, a un locus de origen en un genoma de referencia. De esta manera, las tecnologías NGS disponibles producen un gran número de lecturas cortas que se utilizan habitualmente en las aplicaciones de resecuenciación, lo que implica la disponibilidad de una secuencia de referencia idéntica o bastante similar a la fuente de material genético que se está estudiando.

Además de los objetivos tradicionales de resecuenciación de genomas o descubrimiento de SNP, las características de estas tecnologías permiten aplicarlas con eficiencia a un gran número de investigaciones. Por ejemplo, la NGS de ADNc se puede utilizar para proporcionar una visión completa del transcriptoma para facilitar la anotación de genes e identificación de ayustes (*splicing*) alternativos. Estas tecnologías [3] también se están aplicando para la caracterización de poblaciones de pequeños ARN, la identificación de dianas de los microARN en las plantas, la caracterización de regiones genómicas limitadas por factores de transcripción y otras proteínas de unión, la identificación de patrones de metilación del genoma, la caracterización de patrones de edición de ARN y proyectos de metagenómica. Es lógico pensar también que en los próximos años estas tecnologías seguirán avanzando y nos encontraremos en breve con, a su vez, nuevos métodos de NGS. En la actualidad, los secuenciadores disponibles de nueva generación se basan en distintos métodos químicos para generar datos que producen lecturas de diferentes longitudes, aunque, por su naturaleza, todos son paralelizables masivamente, con lo que presentan nuevos desafíos en términos de los soportes de Bioinformática requeridos para maximizar su potencial informativo.

En este trabajo, intentaremos proporcionar una descripción detallada de las mismas tecnologías de secuenciación. Además, revisaremos algunas de las aplicaciones de estas tecnologías que han emergido en la investigación genómica, focalizándonos particularmente en las herramientas bioinformáticas que se han desarrollado para la gestión y el análisis de datos.

Dado el grado de avance de los desarrollos en este campo, no intentaremos mencionar cada instrumento que se ha presentado, sino que intentaremos proporcionar una visión general de tendencias y enfoques en herramientas con las que los autores de este artículo han tenido un contacto directo.

3.2. PLATAFORMAS DE NGS

Actualmente existen tres plataformas principales de NGS [33] con una amplia difusión y disponibilidad, aunque existen varias iniciativas adicionales [85,86,89] que están entrando en el mercado recientemente. En la tabla 1, se presentan algunas características de su rendimiento, longitudes de lectura y coste (a la fecha de realizar este trabajo).

Tabla 1 Características principales de las plataformas NGS [85,86]

Tecnología / Empresa	**Roche 454**			**Illumina**		**ABI SOLiD**			**Helicos Biosciences**	**Pacific Biosciences**	**ZS Genetics***
Plataforma	*GS 20*	*GS-FLX*	*GS-FLX Titanium*	*GA*	*GA II*	*1*	*2*	*3*	**Helicos tSMS**	**Pacific Biosciences**	**ZS Genetics***
Lecturas (M)	0,50	0,50	1,00	28	100	40	115	400			
Longitud Lectura	100	200	350	35	75	25	35	50	25-45	1.000	N/A
Tiempo ejecución (d)	0,20	5,00	0,30	0,40	4,50	6	5	40.730			
Imágenes (TB)	0,01	0,01	0,03	0,50	1,70	1,80	2,50	3,00	192,00	N/A	N/A
Proceso Químico			Pirosecuenciación (Polimerasa)		Secuenciación por sintesís (Polimerasa).	Secuenciación por ligación (octámeros con código de dos			Polimerasa	Polimerasa	Microscopía electrónica
DNA Molde	PCR Emulsión			PCR Puente		PCR Emulsión			Molécula Única	Molécula Única	Molécula Única
Ventajas	Tiempo de ejecución mejor. Lecturas más largas lo cual es mejor para secuenciación *de novo* .			Plataforma NGS más ampliamente utilizada. Requiere menos ADN. Rápido. Bajo coste por		Ratio de error bajo (doble codificación)					
Inconvenientes	Coste. Dificultadades secuenciando homopolimeros			Menor cobertura. Produce baja calidad en los extremos 3'		Tiempo de ejecución mayor. Se ha demostrado que ciertas lecturas no coinciden con la referencia					
Cobertura Típica	5x-30x			30x-200x		50x-500x					
Imágenes (TB)	0,01	0,01	0,03	0,50	1,70	1,80	2,50	3,00			

* Actualmente no hay mucha información sobre ZS Genetics

Un hilo común de estas tecnologías en los últimos años ha sido la mejora continua en el rendimiento (incremento del número y longitud de lecturas y la consecuente reducción de costes por base secuenciada), por lo que los datos que se muestran quedarán rápidamente obsoletos aunque sirven para ilustrar que la tecnología Roche 454 [83, 44] (Figura 8) proporciona ya un sustituto real para muchas de las aplicaciones de la secuenciación Sanger tradicional a un coste muy reducido, mientras que las plataformas del Illumina Genome Analyser (Figura 10) y el ABI SOLiD (Figura 9, Figura 11) generan más lecturas de longitud menor, características que las hacen, por ahora, más específicas para otras aplicaciones.

Todos los métodos anteriormente mencionados se basan una fase de amplificación anterior al propio proceso de secuenciación. Sin embargo, existen otras tecnologías, como la de Helicos, que suprimen el paso de amplificación y proporciona datos de secuencia para moléculas individuales, minimizando el riesgo de introducir sustituciones durante la amplificación. Los principales enfoques bioinformáticos desarrollados para el análisis de datos generados por las plataformas Illumina GA y ABI SOLiD deben ser adecuados también para los datos generados por el método Helicos (basado en molécula única tSMS, *True Single Molecule Sequencing*, que evita la etapa de amplificación con su Heliscope), ya que las tres plataformas proporcionan lecturas de longitudes similares y comparables. Por último la empresa Pacific Biosciencies (con su PacBio) también utiliza la tecnología de molécula única SMRT (*Single Molecule Real-Time*) y comienzan a aparecer otros métodos basados en la tecnología de nanoporos (Figura 12) (Oxford Nanopore tiene firmado un acuerdo de comercialización con Illumina para que esta pueda distribuir las innovaciones relacionadas con los nanoporos y las corrientes iónicas que fluyen a través de estos para ir identificando secuencialmente las bases de una cadena de ADN) o la microscopía de túneles de electrones (con ZS Genetics y Ion Torrent que ha sido recientemente adquirida por Applied Biosystems y reconvertida en Life Technologies) que están empezando a ser viables.

La información detallada de estos últimos avances no está todavía disponible de manera relativamente fácil, aunque se espera que se puedan producir lecturas individuales con una longitud media de megabases. Dado que estos métodos están prácticamente inaccesibles a la hora de escribir este documento y que la naturaleza de los datos generados deberían ser fundamentalmente diferentes de los proporcionados por las plataformas actuales, podemos considerar que los desarrollos bioinformáticos relacionados con estos métodos quedan fuera del alcance de este documento y no los trataremos de igual manera que lo haremos con los otros.

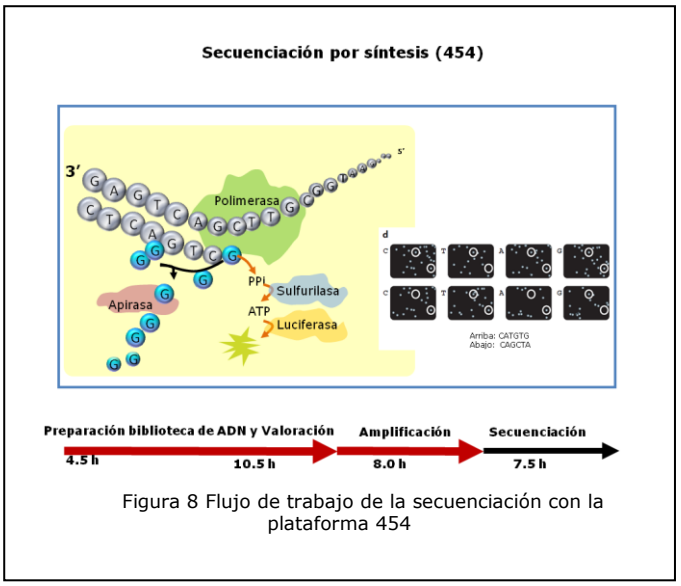


Figura 8 Flujo de trabajo de la secuenciación con la plataforma 454

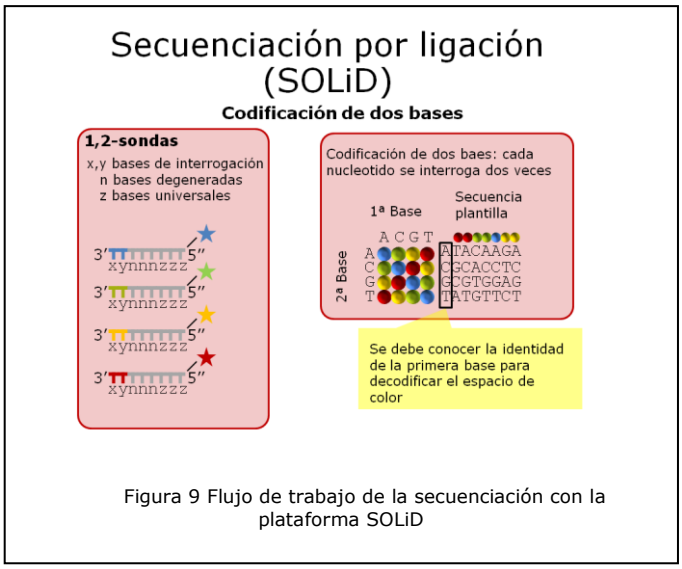


Figura 9 Flujo de trabajo de la secuenciación con la plataforma SOLiD

Las plataformas Illumina GA y Roche 454 utilizan técnicas innovadoras para amplificar y secuenciar, y comparten el principio denominado «secuenciación por extensión» usado en la metodología Sanger. Así se secuencian las bases complementarias a la molécula patrón al agregarlas a una nueva cadena cuya identidad se determina por medios químicos. Sin embargo, la tecnología de secuenciación de ABI-SOLiD utiliza un único proceso químico por el que los oligonucleótidos complementarios se ligan a una nueva molécula y la identidad de las dos primeras bases del oligonucleótido ligado se especifica por un código degenerado de cuatro colores (cada color específica cuatro diferentes oligonucleótidos). Esta aproximación proporciona algunos beneficios en términos de precisión, ya que cada base del patrón se interroga dos

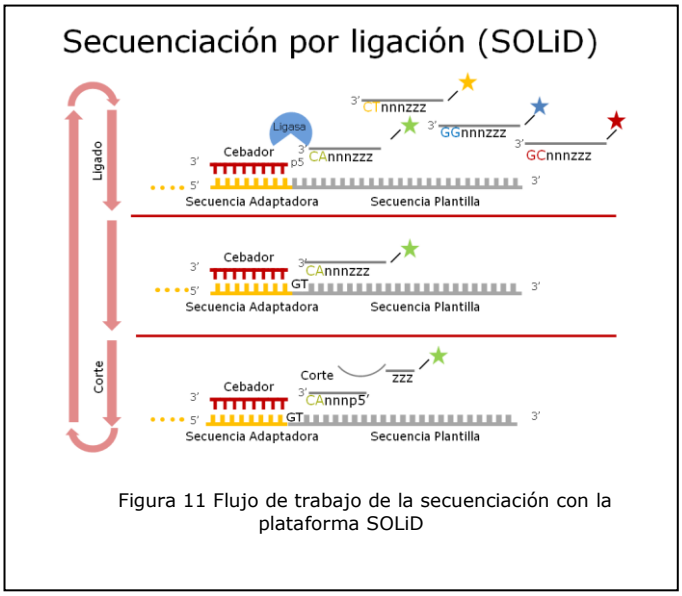


Figura 11 Flujo de trabajo de la secuenciación con la plataforma SOLiD

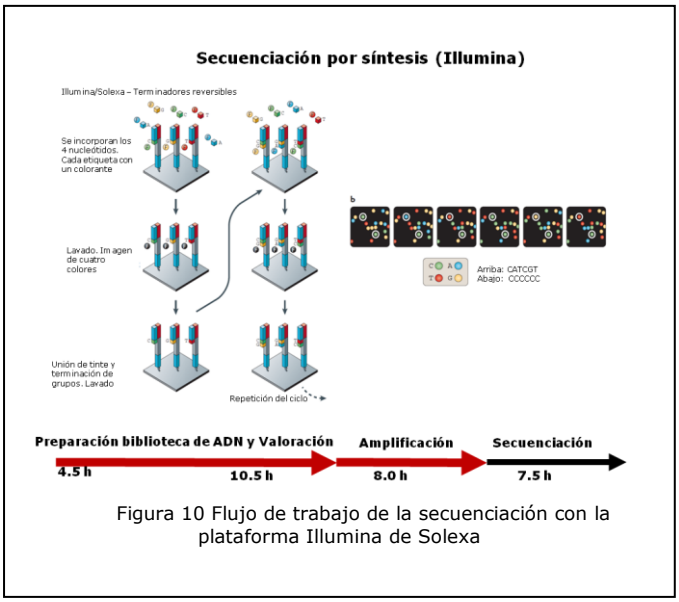


Figura 10 Flujo de trabajo de la secuenciación con la plataforma Illumina de Solexa

veces en rondas de cebadores independientes. Como consecuencia, las lecturas de colores se pueden trasladar a la lectura de la base solamente si se conoce la primera base de la secuencia (o de otra manera, la última base del cebador utilizado). En las aplicaciones de resecuenciación, mediante un estudio detallado de los datos de secuenciación de SOLiD, se puede diferenciar entre errores de secuenciación y SNPs, ya que los errores [7, 85] en la secuencia producen más cambios en las bases cadena abajo (mientras que la variación entre patrones y la secuencia de referencia causan un único error en las lecturas mapeadas) (Figura 13).

De manera análoga a la secuenciación automatizada Sanger, las plataformas de NGS proporcionan indicadores o puntuaciones de calidad basadas en la probabilidad de que una base sea incorrecta. El algoritmo PHRED [17], por ejemplo, asigna un valor de calidad para cada base en una lectura Sanger según QV =-10 log 10(p), donde p es la predicción de probabilidad de que el color sea incorrecto, de manera que un valor alto de QV indica una menor probabilidad de error. Por ejemplo, un valor QV = 20 corresponde a un error de probabilidad de 1 en 100 y un QV = 30 de 1 en 1000. Las plataformas NGS disponen de diferentes perfiles de error de tal manera que se pueden generar valores de calidad en función de las necesidades. El significado de los valores de calidad del Illumina GA son relativamente cercanos a los secuenciadores capilares. Por otra parte, las puntuaciones de Illumina se calculan de otra forma, pero son asintóticamente idénticas la los valores de calidad de PHRED más altos. El intervalo de puntuaciones de calidad en PHRED para Sanger varía entre 0 y 93, mientras que los de Illumina están entre 0 y 40. En el sistema SOLiD, las puntuaciones de calidad se asignan a cada color y se calculan con un método similar al usado en PHRED. Los valores de calidad de SOLiD varían generalmente en un rango de 0 a 45 aunque no se conoce exactamente la relación directa entre las puntuaciones de color y los valores de PHRED. Para las lecturas de 454, los valores de calidad están en un intervalo de 0 a 40 y, de manera similar, se utiliza un algoritmo similar al PHRED aunque la probabilidad de error en las lecturas 454 están principalmente relacionadas con que una base esté sobrevalorada. Las lecturas de Roche 454 son más propensas a errores de inserción y deleción mientras que las de Sanger están más relacionadas con la identificación de bases. Adicionalmente, tenemos que tener en cuenta, que se utilizan diversos tipos de formatos de ficheros de texto para las distintas plataformas NGS, aunque el más frecuente es el FASTQ [116].

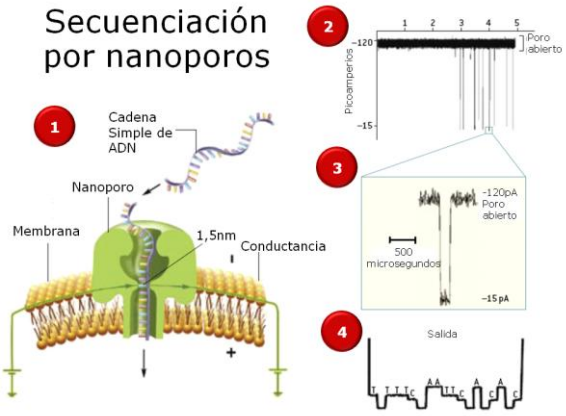


Figura 12 Secuenciación por nanoporos. Se hace pasar la cadena simple de ADN por nanoporos en una membrana suspendida en una solución salina con un voltaje aplicado a través de ella. Los iones que se desplazan de un lado de la membrana al otro crean una corriente eléctrica. Al tiempo que cada una de las cuatro bases de ADN diferentes pasan a través del poro, la intensidad de la corriente disminuye en un grado diferente, lo que permite una rápida secuenciación de las bases

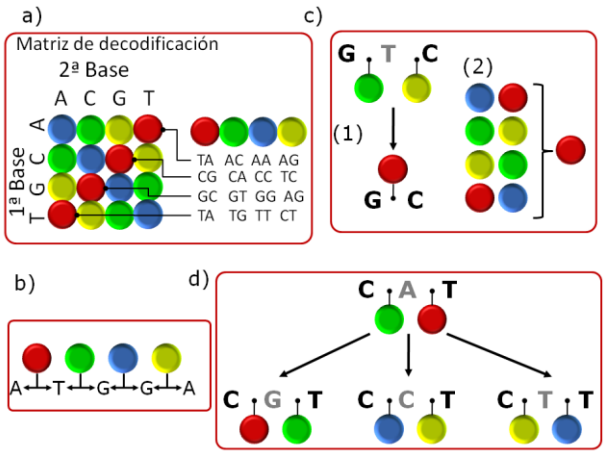


Figura 13 a) la matriz de decodificación permite codificar hasta 16 potenciales combinaciones de dos base mediante 4 colorantes b) Doble interrogación: cada base está definida dos veces. C) si ocurre una deleción en la secuencia GTC el resultado tiene que ser GC. El número de transiciones observadas decrecerá de 2 a 1. La transición individual debe ser de G a C. La situación inversa es cierta si una se produce una inserción de una sola base con el resultado de que solo pueden ocurrir 4 posibilidades de las transiciones adyacentes potenciales. d) si ocurre un SNP en la secuencia CAT existen solo tres resultados posibles: CGT, CCT y CTT. Esto significa que solo se permiten 3 combinaciones de dibases. Debido a que cualquier base se define mediante dos nucleótidos (por ej.: CA y AT), entonces se tienen que observar dos cambios adyacentes para que se produzca cualquier SNP. De esta manera, los errores se representan por cambios individuales. Como hay sólo tres bases como alternativas de ocurrencia cuando se observa un SNP (es decir, una A puede ir a C, G o T) hay solo tres combinaciones de dibases permitidas para cualquier combinación de transición adyacente. Las otras seis combinaciones adyacentes posibles son, por lo tanto, inválidas por definición. De esta manera, cuando se miden dos errores adyacentes, solo 1/3 de ellos pueden serlo.

4. CLASIFICACIÓN DE ALGORITMOS Y PROGRAMAS DE ENSAMBLAJE DE SECUENCIAS

Como se ha comentado anteriormente, el ensamblaje de secuencias es un proceso posterior a la secuenciación cuyo resultado depende de varios factores, como pueden ser el origen de la secuencia a ensamblar, su tamaño y el algoritmo utilizado.

La evolución de los programas de ensamblaje de secuencias ha ido acompañada de la mejora y propuesta de distintos enfoques para resolver el problema. Aunque comienzan a surgir nuevas tecnologías de secuenciación, actualmente las principales se basan en la secuenciación del genoma mediante el método de «perdigonada» mediante el cual el genoma se parte en multitud de pequeños fragmentos (lecturas) que posteriormente se alinean obteniendo la secuencia original. Una primera aproximación para resolver el problema del ensamblaje de estas lecturas fue asumir que cada lectura estaba representada en el genoma original por lo que se deriva que dicho genoma original debe ser la secuencia más corta compuesta por el solapamiento de todas las lecturas, que se conoce como el problema de la Supercadena Común más Corta (*Shortest Common Superstring SCS*) que es NP-completo pero tiene una aproximación algorítmica relativamente eficiente calculando todos los solapamientos posibles entre las lecturas y asignando una puntuación a cada solapamiento potencia. Posteriormente el algoritmo une las lecturas mediante un procedimiento iterativo combinando aquellas lecturas cuyo solapamiento tiene mayor puntuación. Este proceso continúa hasta que no se puedan unir más lecturas.

Aunque esta primera aproximación al ensamblaje de secuencias no modela correctamente el ensamblaje comenzaron a surgir diversos algoritmos que aplicaban esta heurística voraz en su diseño, como por ejemplo **TIGR** [41], **PHRAP** [55] y **CAP3** [144].

Los algoritmos han ido evolucionando y actualmente existen distintas aproximaciones al ensamblaje de secuencias que podemos clasificar atendiendo a los siguientes criterios: origen de las secuencias, tamaño de las lecturas y los algoritmos utilizados. En los siguientes puntos explicamos cada uno de ellos.

4.1. ORIGEN DE LAS SECUENCIAS

Atendiendo a si existe un genoma de referencia del genoma del organismo a secuenciar podemos distinguir dos tipos de ensamblaje de secuencias:

1. Referencia, asignación o identificación (*mapping*): se secuencia el genoma teniendo en cuenta un genoma secuenciado de referencia. Aquí podemos encontrar: **BOWTIE**, **MAQ** (*Mapping and Assembly with Qualities*), **BWA** (*Burrows-Wheeler Alignment Tool*), **SOAP** (*Short Oligonucleotide Analysis Package*), SOAP2 (version mejorada de **SOAP**), **YAGA** (*Zillions Of Oligos Mapped*), **ELAND** (*Efficient Large-Scale Alignment of Nucleotide Databases*), **PASS**, **SHRIMP** (*Short Read Mapping Package*), **AB MAPREADS**, **RMAP**, **gnumap** [65], **GenomeMapper** [59], **NOVOCRAFT** [70], **SLIDER** [60] y **MOSAIK** [49].
2. De-novo: se ensamblan lecturas sin disponer de un genoma de referencia. Aquí podemos encontrar: el **CELERA ASSEMBLER** [37, 79], **PHRAP**, **CABOG** (**CELERA ASSEMBLER** *with the Best Overlap Graph*), **NEWBLER** [48], **ARACHNE** [12], **AMOS**, **ABBA**, **MIRA** (*Mimicking Intelligent Read Assembly*), **ABYSS** (Assembly By Short Sequences), **EULER**, **Velvet**, **EDENA** (*Exact De novo Assembler*) y **SOAP*denovo*** (*Short Oligonucleotide Analysis Package for De novo*), **ALLPATHS**, **SHARCGS,** **LaserGene** [66] y **VCAKE**.

4.1.1. ENSAMBLAJE POR REFERENCIA, ASIGNACIÓN O IDENTIFICACIÓN (*MAPPING*)

El ensamblaje por referencia asigna los *supercontigs* a una estructura los más definida posible del genoma del organismo que se pretende secuenciar. Esta asignación de lecturas es una manifestación clara del problema bioinformático más antiguo: el alineamiento de secuencias. Sin embargo, los métodos clásicos como la programación dinámica de Smith-Waterman, la indexación de las combinaciones de longitud *k* (*K-meros*) en la secuencia patrón (BLAT) [140], o las combinaciones de ambos sistemas (BLAST) [4], no son muy adecuados para el alineamiento de un número muy grande de secuencias cortas contra una secuencia de referencia [23], por lo que se requiere el desarrollo de nuevos métodos matemáticos y heurísticos para conseguir sistemas óptimos de ensamblaje.

Para evitar la necesidad de utilizar excesiva computación, el objetivo global de la asignación de lecturas cortas es obtener resultados satisfactorios de la manera más eficiente posible (en términos de requisitos de memoria y tiempo). Como resultado obtenemos que muchos métodos están basados en principios y algoritmos similares pero difieren en la implementación y en la aplicación de heurísticas concretas con el fin de incrementar la velocidad con la mínima pérdida de precisión [113]. Los desarrollos en estos campos están actualmente en auge y se producen casi semanalmente herramientas nuevas o modificadas [2]. De esta manera, en este documento mostraremos sólo una descripción de los principios generales que contemplan la mayoría de los algoritmos más exitosos.

Podemos decir que la mayoría de las herramientas de asignación de lecturas cortas contempla la creación de un índice de las posiciones de todos los *K-mer*os distintos tanto de las lecturas de la secuencia como de la secuencia del genoma. La diferencia fundamental entre los distintos algoritmos de asignación se basa en si el genoma o las lecturas de la secuencia están indexadas y el método de indexación aplicado. Adicionalmente, aparecen métodos que pueden o no permitir la presencia de *indels* (palabra que proviene de la contracción de 'inserción' y 'deleción') en los alineamientos.

Siguiendo este esquema (Figura 14) se han construido diversas herramientas de asignación, unas para alguna plataforma de secuenciación concreta y otras de carácter general. Entre ellas, podemos destacar las siguientes:

- **ELAND** [5], que ha sido desarrollada en paralelo a la tecnología de secuenciación de Solexa y se distribuye gratuitamente para aquellos que compren el secuenciador. **ELAND** indexa etiquetas y se basa en la estrategia de división de etiquetas; tolera la presencia de errores. Es uno de los programas más rápidos y menos consumidor de memoria.
- **SEQMAP** [56] construye un índice para las lecturas usando la subcadena más larga posible con coincidencia exacta y la compara con el genoma. Permite inserciones y deleciones en los alineamientos.
- **YAGA** [89] está basado en los mismos principios que **ELAND,** con la diferencia que las lecturas se indexan usando semillas espaciadas que se pueden denotar con una cadena binaria. Por ejemplo en una cadena 1110100101011, los 1 significan que en esa posición se ha encontrado una coincidencia y los 0 indican posiciones no seguras. Solamente se indexan las posiciones con un 1. Los autoress indican es que es más rápido que **ELAND** a costa de consumir más memoria.
- **SOAP** [52] (*Short Oligonucleotide Alignment Program*) fue uno de los primeros métodos publicados para la asignación de etiquetas cortas, en el que tanto las etiquetas como el genoma se convierten primero a números mediante una codificación de 2 bits por base. Para admitir dos faltas de coincidencia se parte la lectura en dos fragmentos como en **ELAND** y se permiten o bien faltas de coincidencias o *indels*. [33]. A veces ocurre que el genoma no se puede alinear debido a que las lecturas, por razones técnicas, siempre muestran un mayor número de errores de secuenciación en el extremo 3'. En este caso, **SOAP** puede recortar iterativamente varias bases de ese final 3' y rehacer el alineamiento hasta que se detecte un alineamiento correcto o la secuencia restante sea demasiado corta para la alineación específica que el alineamiento sea específico. El principal inconveniente es el consumo de memoria, ya que se consume más de 10 GB para el genoma humano.
- **PASS** [30] mantiene una tabla *hash* de las posiciones genómicas de las subcadenas semilla (normalmente 11 y 12 bases) en memoria RAM, así como un índice de las puntuaciones precalculadas de palabras (cadenas) cortas (normalmente de 7 y 8 bases)

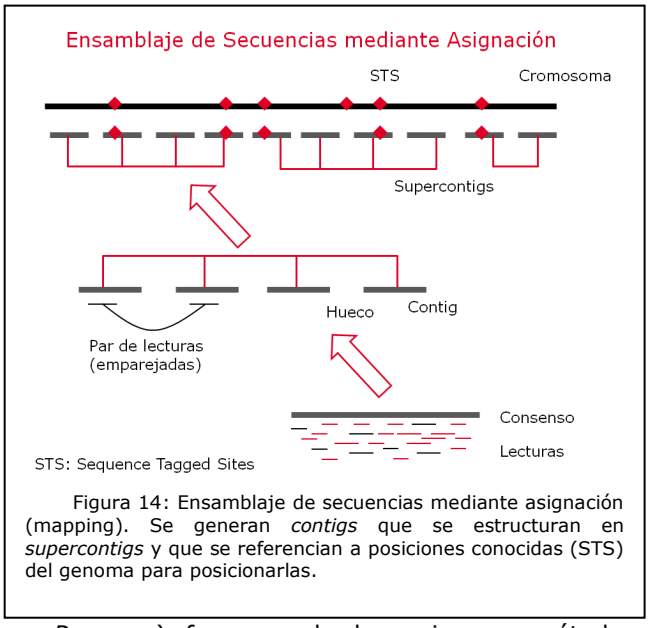


Figura 14: Ensamblaje de secuencias mediante asignación (mapping). Se generan *contigs* que se estructuran en *supercontigs* y que se referencian a posiciones conocidas (STS) del genoma para posicionarlas.

alineadas entre ellas. El programa asigna cada etiqueta mediante tres pasos:

- Primero encuentra palabras coincidentes en el genoma,
- Segundo, por cada coincidencia, chequea los alineamientos precalculados de las regiones laterales (así se incluyen inserciones y deleciones),
- Y Tercero, si se pasa el paso 2 se ejecuta un alineamiento dinámico de una región pequeña alrededor de la coincidencia inicial.

Este algoritmo es más rápido que **SOAP** pero, de nuevo, a costa de consumir gran cantidad de memoria (10 GB) para el índice genómico.

- El algoritmo **MOM** [39] (*Maximun Oligonucleotide Mapping*) [69] busca coincidencias exactas de subcadenas cortas (semillas) entre el genoma y las etiquetas de secuencia y las intenta ampliar con semillas para encontrar la secuencia que coincida más larga posible teniendo en cuenta un número determinado por usuario de divergencias. Para buscar semillas coincidentes, **MOM** crea una tabla *hash* de subsecuencias de longitud fija *k* (*K-mer*os) ya sea del genoma o de las secuencias etiqueta y secuencialmente lee las secuencias sin indexar en busca de coincidencias con los *K-meros* de la tabla *hash*. Al igual que en **SOAP**, dado un número máximo de errores, se produce un recorte de las etiquetas que no coinciden completamente. En términos del número de etiquetas emparejadas correctamente produce un rendimiento mejor que **SOAP**, aunque necesita más de 10 GB para funcionar.
- **BOWTIE** [62] (y una nueva versión de **SOAP**) indexa el genoma mediante un esquema basado en la transformada de Burrows-Wheleer y el índice FM (*Full-text Minute-space*). **BOWTIE** es bastante eficiente ya que solo necesita aproximadamente 1,3 GB de memoria, aunque hay que tener en cuenta que siempre devuelve las coincidencias exactas, pero no cuando la coincidencia mejor sea inexacta.
- **MAQ** [32] (*Mapping and Assembly with Qualities*) es una de las herramientas [67] con más éxito en este campo, ya que está específicamente ideado para hacer uso de los QV de nucleótido que generan las lecturas de Illumina. La idea es que los desajustes producidos por errores en la secuenciación deben aparecer principalmente en aquellas posiciones de las etiquetas que tienen los indicadores de calidad más bajos, mientras que los debidos a SNPs deben aparecer siempre en la misma posición de la secuencia genómica. Los desajustes tienen, por tanto, un peso relativo a sus indicadores de calidad. Por defecto, **MAQ** utiliza seis tablas *hash* para asegurar que una secuencia con dos desajustes o menos se pueda alinear, de manera equivalente a como lo hace **ELAND**. Las seis tablas *hash* corresponden a seis semillas espaciadas, de manera análoga a las que usa **YAGA**. Por defecto, **MAQ** indexa los primeros 28 nt de las lecturas y es muy rápido, pero está basado en una serie de heurísticas que no siempre garantizan encontrar la mejor coincidencia para una lectura.
- **RMAP** [71] al igual que **MAQ** hace uso de los indicadores de calidad generados por Illumina. A las posiciones de las lecturas les asigna una calidad alta o baja en función de si inducen una coincidencia o no, es decir, actúan como asteriscos. Para prevenir la posibilidad de coincidencias triviales se utiliza un paso intermedio de control que elimina las lecturas con excesivas posiciones con baja calidad (un filtro similar también se ha implementado en la herramienta **PASS**).
- **CloudBurst** [96] es un algoritmo paralelizado de asignación de lecturas optimizado para datos NGS que se puede utilizar en numerosos análisis biológicos, que incluyen el descubrimiento de SNP, el genotipado y la genómica personalizada. Está basado en **RMAP** y genera todos los alineamientos para cada lectura con cualquier número de desajustes o diferencias. Llegar a este nivel de sensibilidad puede ser prohibitivo en ejecución, pero **CloudBurst** (está disponible mediante licencia GNU [75]) utiliza la implementación Hadoop del modelo de programación *MapReduce* de Google para paralelizar la ejecución haciendo uso de múltiples nodos de computación. El tiempo de ejecución de **CloudBurst** escala linealmente con el número de lecturas asignadas casi linealmente cuando se incrementa el número de procesadores.

Algunas herramientas de asignación, incluidas **MAQ**, **BWA** [63, 86], **PASS**, **SHRIMP** [73] y **AB MAPREADS,** utilizan espacio de color tanto para la secuencia de referencia como para las lecturas. De esta manera es posible utilizar algoritmos convencionales de alineamiento que se han desarrollado para utilizar con lecturas cortas en Illumina GA y Roche 454.

El rendimiento de los diferentes métodos se tiene que medir de acuerdo con diferentes parámetros: tiempo consumido, ocupación de memoria, espacio en disco y en el caso de herramientas heurísticas, el número actual de lecturas que se han asignado correctamente en su posición en el genoma.

4.1.2. ENSAMBLAJE *DE NOVO*

Las principales herramientas y aplicaciones relacionadas con el ensamblaje de secuencias se basan en utilizar genomas o secuencias genómicas de referencia y contra ellas identificar y asignarlas distintas secuencias que se pretenden ensamblar. No obstante comienzan a surgir herramientas para ensamblar secuencias genómicas *de novo* [86], sin una referencia concreta (Figura 15) y aunque a veces no es posible completar el ensamblaje completo del genoma se pueden construir *contigs* fiables a partir de los datos suministrados cuando las secuencias repetidas no resultan excesivas. El continuo incremento en la longitud de las lecturas de las plataformas NGS permite que, mientras la tecnología 454 ya está completando borradores de genomas microbianos, en un futuro cercano serán viables los proyectos de secuenciación *ab initio* de algunos genomas eucarióticos con tecnologías tales como Illumina o ABI SOLiD.

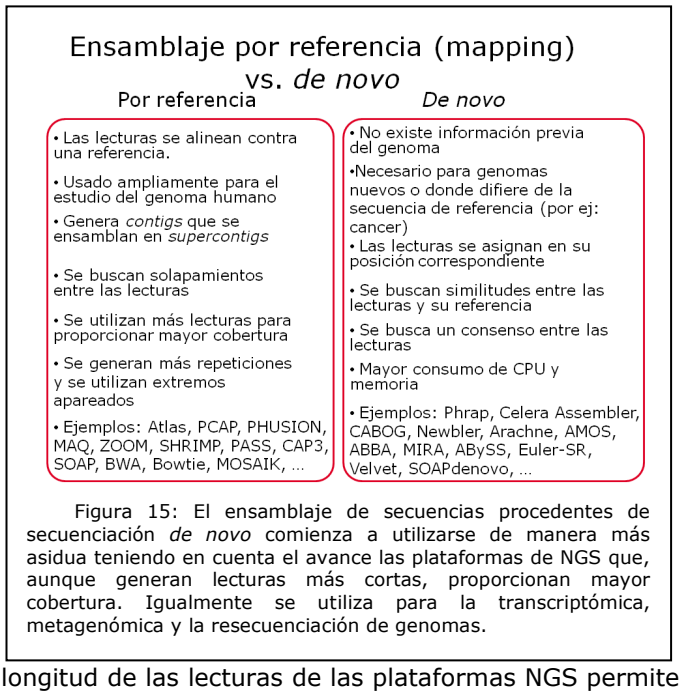


Figura 15: El ensamblaje de secuencias procedentes de secuenciación *de novo* comienza a utilizarse de manera más asidua teniendo en cuenta el avance las plataformas de NGS que, aunque generan lecturas más cortas, proporcionan mayor cobertura. Igualmente se utiliza para la transcriptómica, metagenómica y la resecuenciación de genomas.

El ensamblaje de secuencias *de novo* es más compleja que el ensamblaje por asignación ya que es un problema NP-completo y se puede considerar como una generalización del problema de la supercadena común (dado un conjunto de secuencias, encontrar la secuencia más corta que contenga a todas ellas).

El ensamblaje *de novo* se suele utilizar para resecuenciar un genoma conocido, para secuenciación de genomas pequeños, para transcriptómica y para la metagenómica que trata el estudio de los metagenomas. La metagenómica es una herramienta muy útil para acceder a la elevada biodiversidad de las muestras ambientales y está contribuyendo a caracterizar de forma eficaz la diversidad genética presente en dichas muestras. La información que proporcionan las librerías metagenómicas está enriqueciendo el conocimiento, y por tanto, las aplicaciones prácticas en campos como la industria, la investigación terapéutica o la sostenibilidad del medio ambiente. Este enfoque es muy importante porque produce mucha información acerca de la comunidad de organismos cultivables (en laboratorio) y no cultivables (en el medio ambiente), la frecuencia de especies procariotas y familias de genes sobreexpresados, lo cual permite estudiar la respuesta de los ecosistemas en conjunto ante determinados factores y comprobar cómo se modifican los genomas en respuesta a los estímulos. Aunque tiene como desventaja que la probabilidad de ensamblar un genoma completo a partir de un metagenoma disminuye en relación a la complejidad de la comunidad.

Para el ensamblaje *de novo*, contamos con herramientas que se diseñaron para el ensamblaje convencional de secuencias de datos y orientadas a genomas completos que aunque no están orientadas para manipular grandes cantidades de lecturas producidas por las plataformas NGS ni para gestionar la alta tasa de error que producen dichas plataformas, si están demostrando su utilidad para el desarrollo de ensambladores *de novo*. Entre estas herramientas podemos destacar **ATLAS** [45], **ARACHNE** [12], **PCAP** y **PHUSION** [103].

Entre las aplicaciones específicas disponibles para ensamblaje *de novo* de lecturas de NGS podemos destacar **QSRA** [18], **ALLPATHS**, **Velvet** [29], **EDENA** [64], **VCAKE** [143], **SHARCGS** [36], **EULER-SR** [93, 94] y **SSAKE** [89]. **VCAKE** y **Velvet** utilizan grafos de Bruijn para compendiar la distribución de solapamientos de lecturas, mientras que **EULER-SR** utiliza un enfoque diferente. Estos enfoques tienden a requerir una compensación entre la producción de escasos *contigs* de gran longitud con una baja tasa de cobertura genómica o un alto número de *contigs* cortos con una alta tasa de cobertura genómica.

Se han realizado varios estudios comparativos sobre distintos algoritmos, tales como **QSRA**, **EDENA**, **Velvet**, **SSAKE** y **VCAKE** y destaca que **QSRA** se

comporta mejor que los demás en la velocidad de ejecución [18]. En estas pruebas **EDENA** y **Velvet** generaron *contigs* más largos con baja cobertura genómica mientras que **QSRA**, **SSAKE** y **VCAKE** generaron un número mayor de *contigs* pero más cortos y **QSRA** generó la tasa más alta de cobertura genómica.

Por otro lado, el algoritmo **SHARCGS** es capaz de ensamblar millones de lecturas cortas a la vez que gestiona los errores de manera bastante eficiente. El rendimiento de este algoritmo se evaluó comparándolo con **SSAKE** y **EULER-SR**. Parece ser que **SSAKE** es particularmente vulnerable a la presencia de errores de secuenciación. **EULER-SR** es un gran consumidor de CPU, particularmente cuando existen muchos errores de secuenciación que alteran la complejidad del grafo. Por último el algoritmo **ALLPATHS** permite el análisis de lecturas tanto emparejadas como no emparejadas [119] para ensamblaje *de novo* de genomas completos con microlecturas (25-50 bases).

4.2. TAMAÑO DE LAS LECTURAS

Las primeras técnicas de secuenciación producían fragmentos o lecturas de entre 400 a 800 pb como consecuencia de la secuenciación *Sanger*. Actualmente, las técnicas más comunes (segunda generación de secuenciadores) empleadas producen lecturas más cortas pero a cambio ofrecen una mayor velocidad que se utiliza para dar mayor cobertura, generando un número muy superior de lecturas más corta. Como ejemplo podemos considerar el tamaño de la técnica de secuenciación denominada pirosecuenciación desarrollada por *454 Life Sciences* (empresa adquirida por Roche) que es capaz de producir lecturas de entre 200 a 400 pb y la técnica de *Illumina*, utilizando también una técnica de secuenciación masiva en paralelo del ADN basada también en la polimerización del ADN que produce lecturas de entre 20 y 100 pb, o la tecnología SOLiD (Sequencing by Oligonucleotide Ligation and Detection) de Applied Biosystems que secuencia por ligación de octámeros marcados de secuencia conocida a la cadena de ADN, con la detección posterior de la señal fluorescente emitida tras cada ligación producida y que genera igualmente lecturas de un tamaño de entre 25 y 75pb aproximadamente.

A estas tres últimas tecnologías (454, Illumina y SOLiD) se les denomina de segunda generación o NGS y son las que han producido un avance cualitativo y cuantitativo en nuevos algoritmos de ensamblaje ya que mientras la primera generación, con lecturas largas, permitía una mejor aproximación a la secuenciación *de novo*, las nuevas tecnologías, con la reducción en el tamaño de las secuencias, están mejor preparadas para resecuenciación y transcriptómica. Estas nuevas longitudes más pequeñas de las lecturas producidas han supuesto un giro muy importante en los algoritmos y heurísticas utilizadas en el ensamblaje de secuencias. Es cierto, no obstante, que algunas herramientas y programas se siguen utilizando en el ensamblaje de NGS pero también es cierto que podemos realizar una clasificación de las herramientas en función del tamaño de las lecturas (y su cantidad) para las cuales están mejor orientados. De esta manera tenemos:

1. Ensambladores para lecturas largas, que son aquellos ensambladores que soportan lecturas largas y de los que podemos destacar **PHRAP**, **CAP3**, **TIGR** [41] [146], **PCAP** y **MIRA** [100]
2. Ensambladores para lecturas cortas, que son aquellos que soportan mejor el ensamblaje de muchas lecturas cortas y de los que podemos destacar **SSAKE** (*Short Sequence Assembly by K-mer search and 3' read Extensión*), **Velvet**, **EDENA** (*Exact De novo Assembler*), **EULER-SR** y **ABYSS** (*Assembly By Short Sequencing*).

La evolución de las tecnologías de secuenciación sigue de manera constante y las nuevas tecnologías de secuenciación de tercera generación (ver apartado 3.2) esperan generar longitudes de lecturas del orden de las 1000pb [54].

4.3. TIPOS DE ALGORITMOS

La mayoría de los programas de ensamblaje de secuencias utilizan diferentes aproximaciones [81] algorítmicas de las que podemos destacar: aproximaciones voraces (*greedy*), solapamiento-diseño-consenso *(overlap-layout-consensus*), grafos de de Bruijn y basados en la transformación de Burrows-Wheeler. Existen, no obstante, multitud de algoritmos que se utilizan en el ensamblaje de secuencias como puede ser el algoritmo de Smith-Waterman [90] para alineamientos locales y que se utiliza en multitud de programas de ensamblaje (**ABYSS** [89], **MOSAIK** [49], **PHRAP** [55], **SSAKE** [89] y **TIGR** [41]). Los tres primeros que se han mostrado anteriormente se basan en grafos que son una abstracción bastante usada en informática para el desarrollo de algoritmos. Se trata de un conjunto de nodos (vértices) y sus enlaces (arcos) entre dichos nodos. Si los enlaces poseen una única dirección entre los nodos, se dice que son grafos dirigidos. Los conjuntos de enlaces forman lo que se denomina camino y un camino simple es aquel que tiene nodos distintos (cada nodo se visita una sola vez).

Para representar los ensamblajes de secuencias se utilizan los grafos de solapamiento que representan tanto las lecturas de ensamblaje como sus solapamientos [38]. Los solapamientos deben ser precalculados mediante una serie de emparejamientos de secuencias.

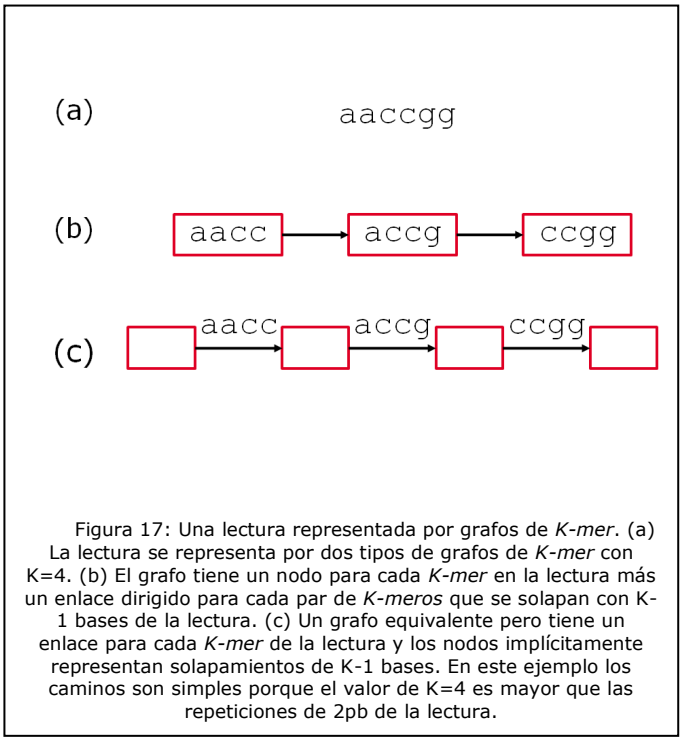


Figura 17: Una lectura representada por grafos de *K-mer*. (a) La lectura se representa por dos tipos de grafos de *K-mer* con K=4. (b) El grafo tiene un nodo para cada *K-mer* en la lectura más un enlace dirigido para cada par de *K-meros* que se solapan con K-1 bases de la lectura. (c) Un grafo equivalente pero tiene un enlace para cada *K-mer* de la lectura y los nodos implícitamente representan solapamientos de K-1 bases. En este ejemplo los caminos son simples porque el valor de K=4 es mayor que las repeticiones de 2pb de la lectura.

Conceptualmente el grafo se utiliza en ensamblaje de secuencias porque tiene nodos que representan a las lecturas y los enlaces representan los solapamientos. En la práctica, el grafo puede tener distintos elementos o atributos para distinguir los extremos 5' y 3' de las lecturas, las secuencias complementarias inversas y directas de las lecturas, las longitudes de las lecturas, las longitudes de los solapamientos y los tipos de solapamientos (sufijo a prefijo y de contención). Los caminos a través del grafo son los *contigs* potenciales y los caminos pueden convertirse en secuencias. Los caminos pueden tener imágenes especulares que representan el complemento inverso de la secuencia. Hay dos maneras para simular la semántica de la doble hélice de ADN:

- Si el grafo tiene nodos separados para los finales de la lectura entonces los caminos se generan a partir del final opuesto de la lectura por la se entra a los nodos.
- Si el grafo tiene enlaces separados para las cadenas directa e inversa entonces los caminos sale de un nodo en la misma cadena por la que entran los enlaces.

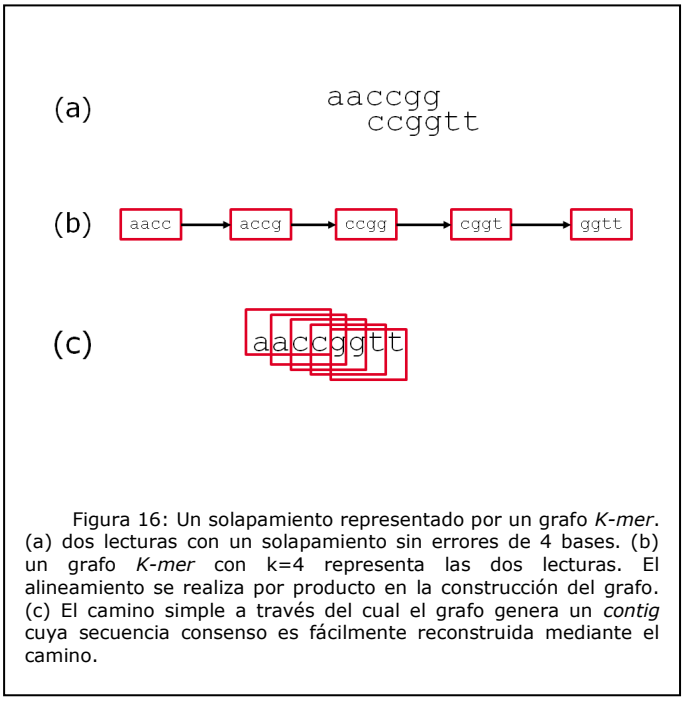


Figura 16: Un solapamiento representado por un grafo *K-mer*. (a) dos lecturas con un solapamiento sin errores de 4 bases. (b) un grafo *K-mer* con k=4 representa las dos lecturas. El alineamiento se realiza por producto en la construcción del grafo. (c) El camino simple a través del cual el grafo genera un *contig* cuya secuencia consenso es fácilmente reconstruida mediante el camino.

El grafo de de Bruijn fue desarrollado fuera del ámbito del ADN para representar cadenas dentro de un alfabeto finito. Los nodos representan todas las cadenas posibles de longitud fija. Los enlaces representan solapamientos perfectos sufijo a prefijo.

Un grafo de *K-mer* es una forma de grafo de Bruijn ya que sus nodos representan a todas las subsecuencias de longitud fijas extraídas de una secuencia mayor. Por su construcción, el grafo contiene un camino que se corresponde con la secuencia original (Figura 16).

Un grafo de *K-meros* puede representar varias secuencias, de tal manera que en su aplicación al ensamblaje de secuencias representa las lecturas de entrada. Cada lectura induce un camino y las lecturas con solapamientos perfectos inducen un camino común, por lo que los solapamientos perfectos se detectan implícitamente sin necesidad de cálculos de alineamientos de secuencias emparejadas (Figura 17). Comparados con los grafos de solapamiento, los grafos de *K-meros* son más sensitivos a las repeticiones y a los errores de secuenciación. Los caminos en los grafos de solapamientos convergen en repeticiones más largas que una lectura pero los caminos en los grafos *K-meros* convergen en repeticiones perfectas de longitud K o mayores de K y K debe ser menor que la longitud de la lectura. En los grafos de *K-meros*, cada error de secuenciación de una base induce K nodos falsos. Cada nodo falso tiene la posibilidad de coincidir con algún otro nodo y por lo tanto inducir a una falsa convergencia de caminos.

En situaciones reales los datos de WGS producen problemas tanto en grafos de solapamientos como en grafos de *K-mer*.

En este tipo de grafos se utilizan algunos conceptos básicos:

- Salientes o espolones, que son divergencias terminales del camino principal (Figura 18a). Se generan debido a los errores de secuenciación en los entremos de una lectura y también cuando la cobertura tiende a cero.
- Burbujas son caminos que divergen y después convergen (Figura 18b). Se generan por errores de secuenciación hacía la mitad de una lectura y por polimorfismos. La detección de burbujas no es trivial [25].
- El patrón deshilachado son caminos que convergen y después divergen (Figura 18c). Se generan por repeticiones en el genoma.
- Los ciclos son caminos que convergen en sí mismos. Se generan por repeticiones en la secuencia.

En general, las divergencias y convergencias incrementan la complejidad del grafo y conducen a enredos difíciles de resolver. Esta complejidad proviene principalmente de repeticiones y errores de secuenciación en las lecturas.

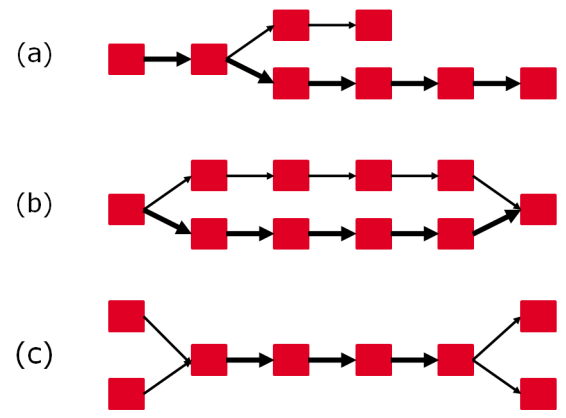


Figura 18: La complejidad de los grafos de *K-meros* se puede diagnosticar debido a la diversidad de información que se puede extraer. En estos grafos los enlaces destacados representan más lecturas. (a) Una base «descolgada» o errante hacía el final de una lectura genera un «espolón» o una rama corta muerta. Este mismo patrón puede ser inducido por una coincidencia de una cobertura nula después de un polimorfismo cerca de una repetición. (b) Una base «descolgada» o errante hacía la mitad de una lectura provoca una «burbuja» o camino alternativo. Este mismo efecto se puede producir en polimorfismos entre cromosomas donantes generando una «burbuja» con paridad de multiplicidad de lecturas en caminos divergentes. (c) Las secuencias repetidas conducen al patrón de «cuerda deshilachada» con caminos convergentes y divergentes [87].

En el contexto de grafos, el ensamblaje es un problema de reducción de grafos y las reducciones de grafos más óptimas son problemas NP-completos por lo que no se conoce una solución eficiente [105]. Por lo tanto los ensambladores se tienen que apoyar en algoritmos heurísticos para eliminar las redundancias, reparar errores, reducir la complejidad, alargar caminos simples y, en cualquier caso, simplificar el grafo.

Por último, hay que tener en cuenta también los algoritmos de los ensambladores desarrollados para o adaptados para ensamblaje de ESTs que se distinguen en varios aspectos. Las secuencias para ensamblajes de ESTs son los transcritos de mRNA de una célula y representan sólo un subconjunto del genoma completo. A primera vista, los algoritmos para ensamblaje de ESTs difieren de los orientados al ensamblaje del genoma ya que por ejemplo, la secuenciación genómica trabaja con muchas secuencias repetidas y, sin embargo, los transcritos no poseen tantas repeticiones. Por otro lado, las células tienen siempre un cierto número de genes que están constantemente expresándose en grandes cantidades con lo que de nuevo aparece el problema de tener secuencias similares que se presentan con un número alto de ocurrencias.

4.3.1. VORACES

Los primeros ensambladores de NGS utilizaban algoritmos voraces [99, 98] que aplican heurísticas en cada fase del algoritmo mediante la cuales pretenden buscar soluciones parciales óptimas y una operación básica: dada una lectura o *contig*, añade una lectura o *contig* más. Esta operación se repite hasta que no sea posible. Cada operación utiliza el solapamiento con más puntuación para genera la siguiente unión. La función de puntuación mide, entre otros indicadores, el número de bases coincidentes en el solapamiento. De esta manera, los *contigs* crecen por extensión tomando la lectura que se encuentra por el siguiente solapamiento con más puntuación. Los algoritmos voraces pueden atascarse cuando un *contig* puntual toma lecturas que deben ser de otros *contigs*.

Los algoritmos voraces son implícitamente algoritmos de grafos ya que simplifican drásticamente el grafo al considerar sólo los enlaces con más puntuación, como optimización pueden instanciar sólo un solapamiento por cada lectura que examinan y pueden también descartar cada solapamiento inmediatamente después de la extensión del *contig*.

Al igual que todos los ensambladores, los algoritmos voraces necesitan mecanismos para evitar incorporar solapamientos falsos positivos en los *contigs* y un ensamblador que se base en solapamientos falsos

positivos unirá secuencias no relacionadas a cualquier lado de una repetición.

El primer ensamblador para lecturas cortas fue **SSAKE** [89] que se diseñó para lecturas cortas no emparejadas de longitud uniforme. Está basado en el concepto de que una alta cobertura debe proporcionar un mosaico con las lecturas que no tienen errores. **SSAKE** no utiliza un grafo de manera explícita sino que utiliza una tabla de búsqueda de lecturas indexadas por sus prefijos. **SSAKE** busca de manera iterativa lecturas que se solapen al final del contig. Las lecturas candidatas deben tener un solapamiento idéntico prefijo a sufijo cuya longitud está definida mediante un umbral y si se da el caso de que existen varias con solapamientos de igual longitud, escoge una mediante heurísticas. De tal manera que primero escoge lecturas con confirmación fin a fin en otras lecturas (lo cual favorece a las lecturas sin errores). En segundo lugar el software detecta cuando un conjunto de candidatas presenta múltiples extensiones y en particular detecta cuando el sufijo de la lectura candidata presenta diferencias que se confirman en otras lecturas. Esto es equivalente a encontrar una rama en un grafo y en este punto el software termina la extensión del *contig*. El usuario puede elegir no utilizar el comportamiento «estricto» donde **SSAKE** utiliza la extensión con mayor puntuación. Cuando no existen lecturas que satisfagan el umbral mínimo inicial, el programa disminuye el umbral hasta que se alcance un mínimo. De esta manera, se puede configurar el comportamiento del programa tanto para los posibles límites del tratamiento de repeticiones como para regiones de baja cobertura. **SSAKE** se ha extendido para explotar las lecturas emparejadas y lecturas que no coinciden perfectamente [121].

SHARCGS [36] también trabaja con longitudes uniformes, alta cobertura y lecturas cortas no emparejadas, pero añade una funcionalidad de pre y postprocesado a **SSAKE**. El preprocesado filtra lecturas erróneas mediante comprobación de un número mínimo de coincidencias exactas de máxima longitud en otras lecturas. Adicionalmente tiene un segundo filtro opcional que requiere que los indicadores de calidad de lecturas coincidentes excedan un umbral mínimo. **SHARCGS** filtra los conjuntos de lecturas en bruto tres veces cada una con un rigor diferente, para generar tres conjuntos filtrados. El programa ensambla cada conjunto de manera separada mediante extensión iterativa de *contigs*. Tras estos pasos, en un postproceso, une los tres conjuntos de *contigs* utilizando alineamiento de secuencias. Esta unión tiene como objetivo ampliar *contigs* de lectura confirmadas mediante la integración de *contigs* más largos que se han generado mediante los filtros menos estrictos.

VCAKE [143] es otro algoritmo de extensión iterativo. Distinto a **SSAKE** y a **SHARCGS**, puede incorporar coincidencias imperfectas durante la ampliación de los *contigs*. **VCAKE** se utilizó en combinación con **NEWBLER** en una integración para datos híbridos de Solexa+454 [125]. Otra integración combinó **NEWBLER** y el **CELERA ASSEMBLER** para datos híbridos de 454+Sanger [42]. Ambas integraciones rompen los *contigs* del primer ensamblador para producir pseudolecturas adecuadas para el segundo ensamblador. Esta última integración ajusta la cobertura de lectura y los indicadores de calidad en las pseudolecturas que genera lo cual ayuda al segundo ensamblador a dar un peso a los *contigs* con gran cobertura desde el primer ensamblaje.

4.3.2. SOLAPAMIENTO-DISEÑO-CONSENSO

El enfoque OLC (Overlap/Layout/Consensus) fue ampliamente utilizado en los ensambladores para datos Sanger y fue optimizado para genomas grandes en diversos tipos de software incluyendo el **CELERA ASSEMBLER** [37, 79], **ARACHNE** [12] y **CAP** y **PCAP** [145], **EDENA**, **CABOG**, **TIGR**, **ATLAS**, **PHRAP**, **PHUSION** y **SHORTY** y ha sido estudiado ampliamente [84].

Los ensambladores que utilizan este enfoque están muy orientados a *de novo* y utilizan un grafo de solapamientos y operan ejecutando tres fases:

1. Para buscar los solapamientos realizan una comparación de lecturas emparejadas todas contra todas. El software precalcula los *K-mer* contenidos en todas las lecturas, selecciona los solapamientos candidatos que comparten dichos *K-mer* y calcula las alineaciones utilizando *K-mer* como semillas de alineamientos. El descubrimiento de solapamientos es sensitivo al tamaño de los *K-mer*, a la longitud mínima de solapamiento y al porcentaje mínimo de identidad requerido en dichos solapamientos. El comportamiento de estos parámetros se ve afectado por los errores de secuenciación y una baja cobertura. Valores mayores conducen a una mayor precisión pero con *contigs* más cortos.
2. La construcción y manipulación de un grafo de solapamientos conduce a un diseño del grafo basado en un conjunto de lecturas representativo, es decir, el grafo no necesita incluir todas las secuencias base por lo que estos grafos pueden utilizarse en grandes genomas ya que pueden ajustar

(mediante el tamaño del grafo) la cantidad de memoria que utilizan.
3. La secuencia consenso se genera mediante un alineamiento múltiple de lecturas aunque no hay un método eficiente para calcularla [90]. Esta fase se puede ejecutar en paralelo, por *contigs*.

Existen dos ensambladores que aplican el enfoque OLC a lecturas cortas de las plataformas Solexa y SOLiD: el software **EDENA**, que fue desarrollado para lecturas no emparejadas de longitud uniforme, descarta lecturas duplicadas y busca todos los solapamientos perfectos libres de errores. Elimina solapamientos individuales que son redundantes con otros solapamientos mediante una aplicación del algoritmo de reducción de solapamientos transitivos [85]. El otro software es **SHORTY** [72, 88] que trata el caso especial donde unas pocas lecturas pueden actuar como semillas para conseguir lecturas cortas y sus extremos apareados, y mediante iteraciones utiliza *contigs* como semillas para generar nuevos *contigs*.

4.3.3. ENFOQUE MEDIANTE GRAFOS DE DE BRUIJN

Esta tercera aproximación para ensamblaje de secuencias se utiliza principalmente para lecturas cortas de las plataformas de SOLiD y Solexa. Se basa en grafos de *K-mer* que los hacen muy eficientes para tratar grandes cantidades de lecturas cortas, ya que los grafos de *K-mer* no requieren un descubrimiento de todos los solapamientos mediante la comprobación de todos contra todos, tampoco es necesario almacenar las lecturas individuales en sus solapamiento y, adicionalmente, comprime las secuencias redundantes. Por el contrario, los grafos *K-mer* no mantienen secuencias en un momento determinado y son grandes consumidores de memoria para grandes genomas, aunque con los sistemas de memoria distribuida se comportan bastante bien [81].

El enfoque de utilizar un grafo de de Bruijn se conoce también como enfoque euleriano y está basado en un escenario ideal en el que, dado un conjunto de datos libres de errores con cobertura total y para cada repetición, el grafo de *K-mer* es un grafo de de Bruijn que contiene un camino euleriano [114], es decir, un camino que cubre todo el grafo y pasa por cada nodo una sola vez. La realidad es que con datos reales no tiene por qué existir este camino y el ensamblaje es realmente un producto del proceso más amplio que la construcción del grafo y generación del camino euleriano. Realmente, la fase de construcción se realiza de manera rápida utilizando una búsqueda en una tabla *hash* para cada ocurrencia de *K-mer* en los datos de entrada y aunque este proceso consume bastante memoria, el grafo *K-mer* almacena cada *K-mer* como mucho una sola vez con lo que no es relevante ocurrencias de *K-mer* se producen en cada lectura. En términos de consumo de memoria, el tamaño del grafo es más pequeño que los datos de entrada ya que hay lecturas que comparten *K-mer*.

Pevzner [88] ya estudió los problemas que las repeticiones genómicas producen en el ensamblaje de secuencias de tal manera que generan bucles en los grafos *K-mer* por lo que permiten más de una posible reconstrucción de la secuencia objetivo. Idury y Waterman [86] también estudiaron los problemas con datos reales y añadieron dos tipos de información extra al grafo *K-mer* y denominaron al resultado un grafo de secuencias: cada enlace se etiqueta con las de las secuencias se generan a partir de las primeras. A su vez los nodos tienen una posición de entrada y una posición de salida y estos tres elementos se adjuntan (comprimidos) en cada enlace y a esto se le denomina eliminación de semifallos (*singletons*). Estos avances junto con nuevos desarrollos produjeron la implementación del ensamblador **EULER** para datos Sanger. Aunque este desarrollo no era utilizable para proyectos de secuenciación Sanger a gran escala, **EULER** y el enfoque de grafos de de Bruijn se posicionaron cuando la plataforma Illumina comenzó a generar datos compuestos por lecturas muy cortas no emparejadas de tamaño uniforme.

Principalmente son tres factores los que complican la aplicación de grafos de *K-mer* para el ensamblaje de secuencias de ADN.

1. El ADN tiene una doble cadena por lo que la secuencia de una lectura dada puede solaparse con la secuencia complementaria inversa o del mismo sentido de cualquier otra lectura. La implementación de un grafo de *K-mer* contiene nodos y enlaces para las dos cadenas intentando evitar el resultado de ensamblar la secuencia dos veces [86]. Existen otras implementaciones que almacenan las dos secuencias (directa y reversa) como medio-nodos afines con la restricción de que los caminos se recorren de la entrada a la salida por una de las mitades [85]. Por último, otras implementaciones representan cadenas alternativas en un único nodo con dos lados, restringiendo los caminos de entrada y salida a los lados opuestos.
2. Los genomas reales presentan repeticiones en estructuras complejas incluyendo repeticiones en tándem, repeticiones invertidas, repeticiones imperfectas y repeticiones insertadas en otras repeticiones. Las repeticiones más largas que K conducen a grafos de *K-mer* confusos que

complican el problema del ensamblaje. Las repeticiones perfectas de longitud K o mayores generan problemas en el grafo de tal manera que se generan estructuras excesivamente ramificadas (estructura de cuerda deshilachada) (Figura 18c): los caminos convergen hasta la longitud de la repetición y a partir de ahí divergen. Para finalizar un ensamblaje con éxito se requiere la separación de los caminos que han convergido que representan una repetición sin solución, es decir, el grafo contiene información insuficiente para solucionar la repetición. Los ensambladores normalmente lo que hacen es consultar las lecturas y los posibles extremos apareados para intentar resolver estas regiones conflictivas.

3. Un palíndromo es una secuencia de ADN que es idéntico a su complemento inverso. Los palíndromos generan caminos que vuelven sobre sí mismos, aunque **Velvet** [85] resuelve este problema de manera elegante escogiendo K impar así un *K-mer* de tamaño impar no puede coincidir con su complemento inverso.
4. Los datos reales incluyen errores de secuenciación. Los ensambladores que utilizan grafos de de Bruijn utilizan varias técnicas para reducir la sensibilidad a este problema. Primero preprocesan las lecturas para eliminar errores; segundo, dan un peso a los enlaces del grafo en función del número de lecturas que soportan y eliminan caminos del grafo; tercero, convierten caminos a secuencias y utilizan algoritmos de alineamiento de secuencias para eliminar caminos idénticos o casi idénticos. Muchas de estas técnicas derivan de la familia de ensambladores **EULER**.

4.3.4. USO DE LA TRANSFORMACIÓN DE BURROWS-WHEELER

La transformación de Burrows-Wheeler [84] (BWT – *Burrows-Wheeler Transform*) es un algoritmo (permutación reversible) utilizado en programas de ensamblaje de secuencias ya que permite dar una solución parcial a uno de los principales problemas del ensamblaje de secuencias que es el consumo de memoria. Este algoritmo permuta el orden de las bases y consigue que la misma base se repita varias veces de manera consecutiva lo cual es útil como paso previo para la compresión y almacenamiento de los datos (Figura 19).

La transformación de Burrows-Wheeler se utiliza en las tareas de alineación contra un genoma de referencia ya que permite reducir la memoria necesaria para indexar el genoma.

Junto a este algoritmo se utiliza un índice comprimido denominado FM-Index [88] que es autocontenido y se utilizó inicialmente para la indexación de textos. Está basado en la Transformación de Burrows-Wheeler junto a una serie de métodos de compresión y una matriz de sufijos comprimida que está cerca del mínimo teórico dado por la entropía de orden cero [89] por lo que con un FM-Index se pueden indexar un conjunto de lecturas sin tener almacenadas las lecturas originales [90, 88].

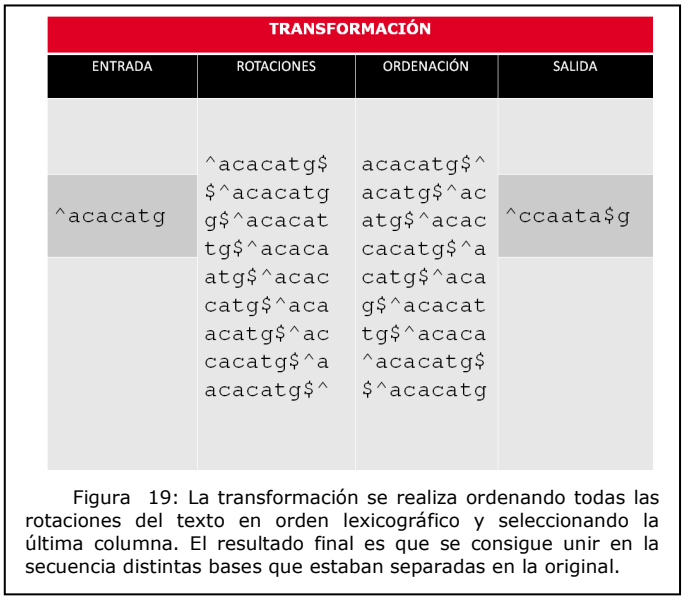


Figura 19: La transformación se realiza ordenando todas las rotaciones del texto en orden lexicográfico y seleccionando la última columna. El resultado final es que se consigue unir en la secuencia distintas bases que estaban separadas en la original.

Existen varios programas de ensamblaje de secuencias que utilizan la transformación de Burrows-Wheeler para optimizar el uso de memoria y mejorar en la eficiencia del algoritmo. Entre ellos podemos destacar **BOWTIE** [85, 86] que utiliza un índice BWT para ajustar el tamaño de la memoria y requiere 2.9GB para genoma humano, **BWA** [86] que tiene versiones tanto para lecturas cortas como largas, **MAQ** [32] que indexa las lecturas con una tabla *hash* y **SOAP2** que utiliza una modificación de BWT denominada *2way-BWT* [86] para los alineamientos que tolera más diferencias en las lecturas más allá de 35pb y ofrece mejoras para el tratamiento de huecos.

5. ALGORITMOS DE ENSAMBLAJE DE SECUENCIAS MAS CONOCIDOS

A continuación vamos a enumerar las características principales de los algoritmos y herramientas de ensamblaje de secuencias más conocidos hasta el momento.

Tabla 2 Tabla resumen algoritmos de ensamblaje

ENSAMBLADORES																
Item	NOMBRE	Website	Fecha Publicación	Tipo	Referencias	Lenguaje	Transcriptómica	Plataforma de Secuenciación	Formato Entrada	Formato de Salida	Algoritmo	Licencia SW	SO	Computación	Creador	Objetivo Principal
1	AB MAPREADS	http://solidsoftwaretools.com/gf/project/mapreads/		Mapping		C++		SOLID	FASTA, CSFASTA	FASTA		GPL				
2	ABBA	http://sourceforge.net/apps/mediawiki/amos/index.php?title=ABBA	sep-08	De Novo assembly for Short Reads	http://www.ploscompbiol.org/article/info:doi/10.1371/journal.pcbi.1000186											
3	ABByS	http://www.bcgsc.ca/platform/bioinfo/software/abyss	feb-09	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/pubmed/19251739 http://bioinformatics.oxfordjournals.org/content/25/21/2872.full	C++	Si				De Bruijn Graphs				Inanc Birol, Steven Jones, et al., Genome Sciences Center, British Columbia Cancer Agency	
5	ALLPATHS	http://www.broadinstitute.org/science/programs/genome-biology/computational-rd/computational-research-and-development	oct-09	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18340039	C++		Illumina			Assembly, De Bruijn Graphs				Jonathan Butler, Iain MacCallum, David Jaffe, et al., Broad Institute of MIT and Harvard	
4	AMOS	http://sourceforge.net/apps/mediawiki/amos/index.php?title=AMOS		De Novo assembly for Short Reads												
6	ARACHNE	http://www.broadinstitute.org/crd/computational-research-and-development	dic-02	Long reads (Sanger Method). WGS	http://www.ncbi.nlm.nih.gov/pubmed/11779843	C										
7	ATLAS		feb-04	Mapping	http://www.ncbi.nlm.nih.gov/pmc/articles/PMC383319/											
8	Bowtie	http://bowtie-bio.sourceforge.net/index.shtml	mar-09	Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=19261174						Burrows-Wheeler Transform			Paralelo		
9	CABOG		oct-08		http://bioinformatics.oxfordjournals.org/content/24/24/2818.full.pdf+html											
10	CAP3	http://deepc2.psi.iastate.edu/aat/sas.html	oct-02	Mapping	http://genome.cshlp.org/content/9/9/868.long	C	Si	N/A	FASTA					Secuencial	Xiaoqiu Huang (Iowa State University)	Ensamblaje de secuencias mediante búsqueda de la cadena común más larga
11	Celera Assembler	http://sourceforge.net/apps/mediawiki/wgs-assembler/index.php?title=Main_Page	mar-00													
12	Edena	http://www.genomic.ch/edena.php	oct-08	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18332092		Si	Illumina	FASTA, FASTQ		Overlap layout				David Hernandez et al., Genomic Research Laboratory, Geneva University Hospitals	
13	ELAND	http://bioinfo.cgrb.oregonstate.edu/docs/solexa/	feb-08	Mapping				Illumina								
14	EULER	http://euler-assembler.ucsd.edu/portal/	jun-01	Short Read Assembly	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18083777	, Perl					Assembly, De Bruijn graph		Linux		University of California, San Diego	
15	EULER-SR	http://euler-assembler.ucsd.edu/portal/	nov-08	De Novo assembly for Short Reads	http://genome.cshlp.org/content/early/2008/12/03/gr.079053.108.abstract	C++					Assembly, De Bruijn graph					
16	MAQ	http://sourceforge.net/projects/maq/	nov-08	Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18714091			Illumina (Funciones básicas para SOLiD)			Burrows-Wheeler Transform					
17	MIRA3	http://chevreux.org/projects_mira.html	may-04	De Novo, Mapping			Si				Smith-Waterman		MS, Mac OS X, Lunix (32 y 64 bits), Solaris		Boston College	
18	MOSAIK	http://bioinformatics.bc.edu/marthlab/Mosaik	mar-08	Mapping		C++					Smith-Waterman	Dual				
19	NEWBLER	http://454.com/products-solutions/analysis-tools/gs-de-novo-assembler.asp		De Novo				454				Comercial	Linux			
20	PASS	http://pass.cribi.unipd.it/cgi-bin/pass.pl	feb-09	Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=19218350	C++		Illumina, SOLiD, 454								
21	PCAP	http://seq.cs.iastate.edu/	oct-02	Mapping	http://www.ncbi.nlm.nih.gov/pubmed/12952883									Paralela		
22	PHRAP		oct-99			C	Si				Smith-Waterman			Secuencial		
23	PHUSION		nov-02													
24	SeqMap	http://www.stanford.edu/group/wonglab/jiangh/seqmap/	ago-08	Mapping	http://bioinformatics.oxfordjournals.org/content/24/20/2395.full.pdf+html				FASTA	Varios			Mac OS X, Windows, Linux			
25	SHARCGS	http://sharcgs.molgen.mpg.de/	ago-07	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=17908823	Perl		Illumina			Assembly		Linux		Juliane Dohm, Heinz Himmelbauer, et al., Max-Planck-Institute for Molecular Genetics, Berlin	De novo assembly
26	SHORTY	http://www.cs.sunysb.edu/~skiena/shorty/	may-07	De Novo assembly for Short Reads	En revisión						Assembly, De Bruijn graph				Juliane Dohm, Heinz Himmelbauer, et al., Max-Planck-Institute for Molecular Genetics, Berlin	
27	SHRiMP	http://compbio.cs.toronto.edu/shrimp/		Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=19461883			SOLiD, Helicos, 454, Illumina			Mapping, Colorspace					
28	SOAP	http://soap.genomics.org.cn/	ene-08	Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18227114	C++					Alignment, Mapping, Burrows-Wheeler Transform		Unix		Beijing Genomics Institute	SNP Discovery
29	SOAPdenovo	http://soap.genomics.org.cn/	jul-09	De Novo		C	Si									
30	SSAKE	http://www.bcgsc.ca/platform/bioinfo/software/ssake	dic-06	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=17158514			Illumina, SOLiD, 454			Assembly				Rene Warren, Robert Holt, et al., Genome Sciences Center, British Columbia Cancer Agency	
31	VCAKE	http://sourceforge.net/projects/vcake/	ago-07	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=17893086			Illumina, SOLiD, 454	FASTA	FASTA	Assembly, K-mer extension	GPL	Linux, MacOS		William Jeck, Corbin Jones, et al., University of North Carolina at Chapel Hill	De novo assembly
32	Velvet	http://www.ebi.ac.uk/~zerbino/velvet/	mar-08	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18349386	C	Si	Illumina	FASTA, FASTQ, ELAND, GERALD, fasta.gz, fastq.gz		Assembly, De Bruijn graph	GPL			Daniel Zerbino and Ewan Birney, European Bioinformatics Institute	De novo assembly
33	YAGA	http://www.bioinformatics.iastate.edu/News/YAGA_Alura_2010.htm	ene-09	De Novo assembly for Short Reads	http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2648799/?tool=pubmed	C++	Si								Benjamin G Jackson, Patrick S Schnable, and Srinivas Aluru	
34	ZOOM	http://www.bioinformaticssolutions.com/products/zoom/index.php	ago-08	Mapping	http://www.ncbi.nlm.nih.gov/sites/entrez?Term=18684737			454, Illumina					Unix, Windows		Bioinformatics Solutions Ic.	

5.1. AB MAPREADS

SOLiD System Color Space Mapping Tool (**AB MAPREADS**) [35] es la herramienta de Applied Biosystems para el sistema de secuenciación SOLiD y está desarrollado para referenciar lecturas de espacio de color de SOLiD a genomas humanos completos.

La entrada para el sistema es un genoma de referencia en formato FASTA y un fichero de lecturas del sistema SOLiD en formato CSFASTA. El sistema busca todas las concordancias entre las lecturas y la secuencia de referencia permitiendo hasta N no coincidencias. La asignación se ejecuta en espacio de color realizando en tiempo real la traslación de la secuencia de referencia.

El sistema utiliza varias heurísticas para mejorar los resultados, tales como puntuar de manera distinta las discordancias adyacentes de las lecturas en función de indicadores de calidad, o normalizar las puntuaciones de calidad para tratar mejor las discordancias.

El sistema está desarrollado en C++ y la última versión conocida es la 2.4.1.

5.2. ABBA

ABBA (*Assembly Boosted By Amino acid sequences*) es un ensamblador denominado

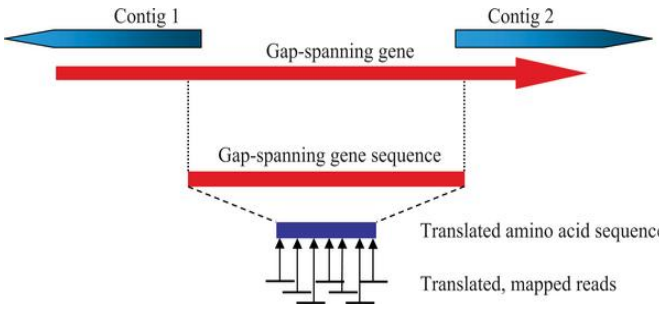


Figura 21 Ensamblaje de genes con **ABBA** [135]. Todos los *contigs* se alinean con secuencias de genes predefinidos para identificar genes que abarcan dos o más *contigs*. Las secuencias de ADN de estos últimos genes se cortan con un pequeño margen en los extremos. Posteriormente se busca la traslación a aminoácido de cada fragmento de gen en la lista de lecturas que todavía no han sido ensambladas. Finalmente, se ensamblan las lecturas que se hayan identificado por este proceso y se añaden los dos *contigs* para rellenar el hueco.

comparativo (se utilizan secuencias similares como referencia) para lecturas cortas que utiliza una secuencia de aminoácidos para dirigir el ensamblaje de genes. Las secuencias de aminoácidos están más conservadas que el ADN lo cual permite aumentar la distancia relativa que se usa como referencia (Figura 21).

Actualmente **ABBA** está integrado dentro del paquete **AMOS** de libre distribución que también dispone del ensamblador **AMOScmp**.

5.3. ABYSS

ABYSS [89] son las siglas de *Assembly By Short Sequences* y se trata de un ensamblador para secuencias *de novo* que está diseñado para lecturas

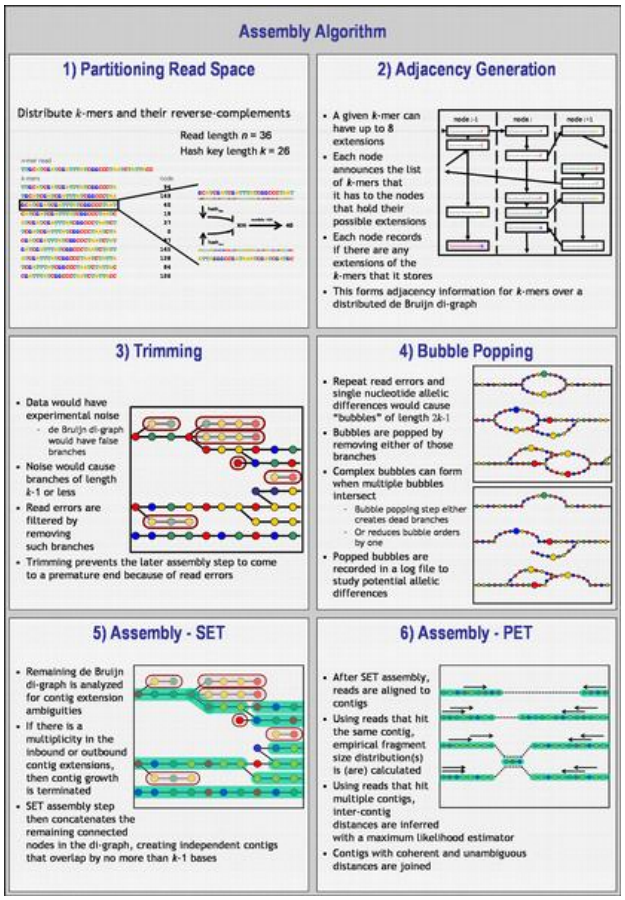


Figura 20 Fases del algoritmo utilizado en **ABYSS**

muy cortas. La versión de un solo procesador es útil para ensamblar genomas de hasta 100 Mbases. Existe una versión que está implementada utilizando librerías MPI y es capaz de ensamblar genomas mayores. Está desarrollado por Simpson JT y otros en el Canada's Michael Smith Genome Sciences Centre en lenguaje C++.

En Internet existe más información y la posibilidad de descarga [133].

ABYSS es una implementación distribuida [89] diseñada para poder trabajar con ensamblaje de genomas del tamaño humano con grafos de de Bruijn

teniendo en cuenta las limitaciones de memoria. **ABYSS** distribuye el grafo de *K-meros* y su computación correspondiente en varios procesadores utilizando la memoria de manera compartida (Figura 20). Esta implementación permite ensamblar hasta 35.000 millones de lecturas Solexa de un genoma humano.

Existen varios problemas a los que se enfrenta un algoritmo diseñado para una sola CPU cuando se traslada a una matriz de computación ya que debe ser posible partir el problema para que se pueda resolver mediante computación en paralelo distribuyendo estas particiones entre la matriz y posteriormente ordenar los resultados. Las particiones de **ABYSS** llegan a la granularidad de nodos individuales del grafo (cada nodo se procesa separadamente asignándolo a una CPU) y esto se acompaña convirtiendo los *K-mer* a un entero. Esta fórmula de conversión es neutral de tal manera que un *K-mer* y su complemento inverso se asignan al mismo entero y se utiliza para asignar *K-meros* vecinos a la misma CPU.

ABYSS implementa para computación paralela los mecanismos de simplificaciones de grafos de **EULER** y **Velvet** y elimina los ramales cortos terminales cuando son más cortos que un umbral definido. **ABYSS** trata las burbujas mediante búsquedas acotadas y escoge el camino más utilizado por las lecturas. También transforma en *contigs* caminos simples no interconectados y ejecuta un seguimiento de pares. Utiliza una representación compacta de grafos de *K-mer* donde cada nodo representa tanto un *K-mer* como su complemento inverso y mantiene 8 bits de información extra donde almacena la existencia o no de cada una de las posibles extensiones (bases) de cada final donde los enlaces del nodo mantienen implícita esta información. **ABYSS** recorre los caminos en paralelo comenzando en nodo arbitrario del grafo y desde este busca sucesores: las bases K-1 del nodo más una base indicada por un enlace se convierten (incluyendo la asignación de la CPU) numéricamente en la dirección del nodo sucesor. Cuando un camino un nodo que está en una CPU diferente, el proceso emite una solicitud de información y mientras espera la respuesta, el programa contínua de manera asíncrona el tratamiento de otros nodos del grafo.

En un posproceso **ABYSS** utiliza lecturas emparejadas para unir *contigs* pero no construye *supercontigs*. En resumen **ABYSS** es un software de ensamblaje escalable para lecturas cortas y emparejadas Solexa.

5.4. ALLPATHS-LG

ALLPATHS-LG es un ensamblador para genomas completos que genera ensamblajes de alta calidad a partir de lecturas cortas basado en grafos de de Bruijn. Fue publicado con resultados en datos simulados [76] y revisado para datos reales [57]. La versión LG (*Large Genome*) es capaz de ensamblar grandes genomas (hasta el tamaño del genoma humano) utilizando las últimas 100 bases emparejadas de las lectura generadas por Illumina.

Los autores de **ALLPATHS-LG** indican que el proyecto está en continuo desarrollo por lo que no existe una versión «oficial» del mismo (desde la página web se puede descargar continuamente la última versión estable).

Está desarrollado por el Broad Institute en Cambridge por Sante Gnerre, Iain MacCallum y otros. En Noviembre de 2010 se realizó una revisión del programa que se ha publicado en PNAS [89] indicando las mejoras y el nuevo enfoque de la versión LG, que se pueden resumir de la siguiente manera:

- Manipulación de secuencias repetitivas. Las secuencias repetitivas generan grandes problemas para ensamblajes de calidad. La versión original **ALLPATHS** dejaba de funcionar correctamente para repeticiones >= K, donde K es un número lo suficientemente pequeño para que existan bastantes lecturas con solapamientos de longitud K. En **ALLPATHS-LG** lo que se ha hecho es poder trabajar con un K mayor ejecutando un paso inicial denominado «doble lectura», en el cual las dos secuencias finales de un fragmento se unen y se detecta si una tercera genera un solapamiento o un hueco, aunque realmente se pueden genera otras ocurrencias tales como que un SNP aparezca entre los dos extremos del par.
- Mejora en la corrección de errores. Para ello cíclicamente se revisan lecturas individuales que indiquen claramente que se trata de un error y se editan para corregirlo.
- Se mejora el tratamiento de saltos entre datos.
- Uso más eficiente de memoria.
- Mejora para el ensamblaje de regiones con baja cobertura. En tal caso, **ALLPATHS-LG** puede utilizar un K menor de 15 para tratar mejor los solapamientos.

ALLPATHS utiliza un preprocesador para corrección de lecturas relacionado con el alineamiento espectral del **EULER** de manera que confía en *K-meros* que tienen una alta frecuencia y calidad en las lecturas. El filtro funciona con *K-meros* para tres valores de K y las lecturas se recuperan si ocurren hasta dos sustituciones de bases con baja calidad lo cual genera *K-meros* confiables. S pueden recuperar *K-meros* posteriormente si son esenciales para construir el camino entre lecturas de extremos apareados.

ALPATHS invoca un segundo preprocesador que crea caminos únicos (*unipaths*). Este proceso comienza

con el cálculo de solapamientos perfectos de lecturas donde los *K-meros* se utilizan de semillas asignando identificadores numéricos a dichos *K-meros* de tal manera que reciben identificadores consecutivos cuando aparecen muchos manera consecutiva en lecturas y solapamientos. También rellena una base de datos con intervalos de identificadores y con los enlaces de las lecturas entre ellos, y posteriormente une dichos intervalos con una medida que es consistente con todas las lecturas, operación que es equivalente a la construcción de un grafo de de Bruijn seguido de la eliminación de los únicos (*singletons*). Esta implementación de la base de datos pretende reducir los requisitos de memoria para la propia construcción del grafo que se realiza en el paso siguiente.

ALLPATHS construye un grafo de de Bruijn a partir de la base de datos anterior y la primera operación es la eliminación de los ramales pequeños terminales (*spur*). También utiliza particiones en el grafo para resolver repeticiones mediante ensamblaje de regiones que no están repetidas localmente, para ello aplica heurísticas de cara a escoger las particiones que formen un camino consistente a lo largo del genoma. También utiliza semillas en particiones con nodos que se corresponden con *contigs* largos, moderadamente cubiertos y ampliamente separados y da más valor a las particiones que tienen nodos y lecturas enlazados por alineamientos y extremos emparejados. Por otro lado busca rellenar huecos entre lecturas emparejadas realizando una búsqueda en el grafo de instancias donde exactamente un solo camino satisface una restricción de distancia. **ALLPATHS** ensambla cada partición de manera separada y en paralelo para, posteriormente, unir los grafos locales donde tienen una estructura solapada. En el grafo global **ALLPATHS** elimina de manera heurística los ramales pequeños terminales, los pequeños componentes desconectados y los caminos no creados por extremos apareados.

En resumen, **ALLPATHS** está desarrollado para el ensamblaje de grandes genomas utilizando lecturas cortas de la plataforma Solexa con extremos apareados. Sus filtros de lecturas utilizan valores de calidad para ajustar los errores de sustitución y simplifica el grafo que inicialmente está basado en lecturas y solapamientos. Más allá de las propias lecturas y el seguimiento de emparejamientos, aplica los datos generados por dichas lecturas y extremos aparados en un enfoque divide y vencerás ejecutado externamente al grafo.

5.5. **AMOS**

AMOS (*A Modular, Open-Source whole genome assembler*) [61] es un conjunto de herramientas, clases e interfaces para la construcción de ensambladores de secuencias de ADN. El paquete incluye tanto el software base como las interfaces para ensamblar distintos módulos y herramientas adicionales para solapamiento, generación de consenso, *contigs* y gestión de ensamblajes.

AMOS también se denomina al consorcio constituido para el desarrollo de dicho software con licencia de código abierto. Los principales colaboradores del consorcio son la Universidad de Maryland (Center for Bioinformatics and Computational Biology), El Institute for Genomic Research, el Karolinska Instituted y el Marine Biological Laboratory (Woods Hole).

El conjunto de herramientas que constituyen **AMOS** se pueden clasificar de la siguiente manera:

5.5.1. ENSAMBLADORES

- **ABBA** (Assembly Boosted By Amino Acid Sequences)
- **AMOScmp**: ensamblador comparative
- **AMOScmp-shortReads**: ensamblador comparative para lecturas cortas (Solexa, 454)
- **AMOScmp-shortReads-alignmentTrimmed**: ensamblador comparativo para lecturas cortas que utilizan alineamientos basados en recortes.
- **minimus**: ensamblador básico para pequeños conjuntos de datos.
- **minimus2**: ensamblador básico que trabaja con dos conjuntos de datos que pueden usarse para genera un único ensamblaje.
- **Minimo**: es el ensamblador *minimus* pero con más opciones.

5.5.2. VALIDACIÓN Y VISUALIZACIÓN

- **Hawkeye**: visor de ensamblaje.
- **AMOSvalidate**: herramienta de análisis forense del ensamblaje.
- **Benchmark**: herramienta para pruebas de datos

5.5.3. *SCAFFOLDING*

- **Bambus**: herramienta de para la creación de *supercontigs* de manera jerárquica y de código abierto.
- **Bambus2**: andamiaje para genomas polimórficos y metagenomas.

5.5.4. RECORTE, SOLAPAMIENTO Y CORRECCIÓN DE ERRORES

- **Figaro**: vector de recorte estadístico.
- **UMD Overlapper**: para cálculo de solapamientos.

- **KI Overlapper**: para la manipulación de problemas creados por errores de secuenciación.
- **AutoEditor**: corrección automática de errores de secuenciación.
- **FastQC**: composición de lecturas y calidad.

5.5.5. UTILIDADES

- **File Conversion utilities**: para convertir datos a y desde **AMOS**.
- **AMOS Utilities**: utilidades generales.
- **runAMOS**: *script* de ejecución.

5.6. **ARACHNE**

ARACHNE [12] es un ensamblador propuesto por el profesor S. Batzoglou en el año 2002 [13] para el ensamblaje de secuencias genómicas usando lecturas largas con extremos emparejados que utiliza el enfoque de solapamiento por consenso con varias características principales:

- Incluye un procedimiento eficiente y preciso para la búsqueda de solapamientos en las lecturas.
- Posee un sistema para unir las lecturas mediante enlaces dinámicos rehaciendo el grafo hacía adelante y atrás (*forward-reverse*).
- Detecta *contigs* repetidos mediante inconsistencia de los enlaces anteriores.

El sistema comparte algunas características con el **CELERA ASSEMBLER** tales como el algoritmo unión de lecturas para formar *contigs* y, por el contrario, posee otras diferencias como es el uso de frecuencias de *K-meros* para identificar repeticiones y mediante su ordenación es capaz de detectar solapamientos (Celera lo que hace es filtrar repeticiones predefinidas). **ARACHNE** también comparte algunos aspectos con otros ensambladores tales como **PHRAP** [55] y **CAP3** [144] como es en la creación, ajuste y evaluación de alineamientos de lecturas mediante puntuaciones de calidad.

ARACHNE está diseñado para analizar lecturas de secuencias obtenidas de ambos extremos de plásmidos clonados, es decir, lecturas emparejadas en cualquier orden de secuencia del plásmido. A cada base de una lectura se le asocia una puntuación de calidad que la genera a su vez el programa PHRED [17] de tal manera que una puntuación q corresponde a la probabilidad de $10^{-q/10}$ de que la base sea incorrecta y una puntuación de 40 al 99,99% de que sea correcta. En un paso inicial **ARACHNE** elimina regiones concretas de lecturas, lecturas completas con baja probabilidad y secuencias que pudiesen estar contaminadas por las anteriores.

A partir de esta fase inicial, el algoritmo sigue los siguientes pasos:

- Detección de solapamientos y alineamiento: ordenación y extensión.
- Corrección de errores
- Evaluación de las alineaciones
- Identificación de secuencias emparejadas (*paired pairs*)
- Ensamblaje de *contigs*
- Detección de *contigs* repetidos
- Creación de *supercontigs*
- Relleno de los huecos en los *supercontigs*
- Refinamiento de consenso

5.7. **ATLAS**

ATLAS [45] es un conjunto de programas desarrollados por Havlak et al. (2004) para el ensamblaje de genomas mediante un enfoque combinado que utiliza lecturas de secuencias tanto de BACs como WGS. Los clones BAC presentan la ventaja de un ensamblaje localizado por lo que se reduce la carga computacional a la vez que se proporciona un método eficaz para trabajar con secuencias repetidas. La inclusión de secuencias WGS facilita el uso de insertar clones de diferente tamaño a la vez que reduce los costes de producción. Una de las funciones principales de **ATLAS** es el agrupamiento de secuencias WGS en BACs concretos, lo cual se realiza mediante secuencias solapadas. El procedimiento va ensamblando simultáneamente sólo pequeñas unidades del genoma debido a que la construcción de las secuencias consenso se realiza desde un ensamblaje en local de las lecturas. Una vez ensamblado, cada BAC se utiliza para obtener una capa genómica. Este crecimiento del mapa del genoma basado en secuencias base tiene mayor precisión que los que utilizan métodos no basados en secuencias. El uso de BACs permite también la corrección de errores debido a repeticiones en cada una de las etapas del proceso para lo cual se utilizan diversas estrategias y heurísticas. **ATLAS** ha sido utilizad para el borrador de un ensamblaje de ratón e incluye herramientas de gestión de solapamientos y andamiajes que se pueden utilizar en proyectos WGS.

5.8. **BOWTIE**

BOWTIE [84] es lo que los autores denominan un alineador de lecturas cortas eficiente en uso de memoria y ultrarápido. Está orientado a pequeños conjuntos de datos de grandes genomas. Es capaz de alinear secuencias cortas (lecturas) de genoma humano a un ratio de 25 millones por hora con lecturas de 35pb. **BOWTIE** utiliza un índice Burrows-Wheeler para gestionar el uso de memoria, utilizando alrededor de 2.2GB para el genoma humano. Permite el uso de

varios procesadores para trabajar en paralelo y forma parte de otras herramientas tales como **TopHat**, **Cufflinks** (herramienta para ensamblaje de transcriptoma y cuantificación isomorfa de lecturas RNA-seq), **Crossbow** (herramienta software para resecuenciación en *cloud computing*) y **Myrna** (herramienta *cloud computing* para el cálculo diferencial de expresión génica en grandes conjuntos de datos RNA-seq).

5.9. CAP3

CAP3 [144] fue desarrollado por Huang y Madan en 1999 y es la tercera generación del programa de ensamblaje de secuencias **CAP** que incluye mejoras y nuevas funcionalidades. Está diseñado para trabajar con secuencias por perdigonada e implementa el enfoque de diseño por consenso de solapamiento. Las etapas operacionales de **CAP3** se describen en la siguiente ilustración. El programa tiene la capacidad de cortar regiones finales (tanto 5' como 3') de lecturas de baja calidad. Utiliza valores base de calidad en el cálculo de solapamientos entre lecturas, construcción de múltiples alineamientos de secuencias de lectura y generación secuencias consenso. Se utiliza también para el ensamblaje de ESTs. El programa también utiliza restricciones hacía adelante y hacia atrás (forward-reverse) para corregir errores de ensamblaje y enlaces de *contigs*. Los autores compararon el rendimiento de **CAP3** con respecto a **PHRAP** para cuatro conjuntos de datos de BAC y observaron que **PHRAP** produce *contigs* más largos donde **CAP3** a menudo produce menos errores en secuencias consenso. Se utilizan los valores base de calidad generados por PHRED para calcular los solapamientos entre lecturas, construcción de múltiples alineamientos de secuencias de lectura y generación de secuencias consenso. También se utilizan algoritmos eficientes para identificar y calcular solapamientos entre lecturas.

Una característica inusual de **CAP3** es el uso de restricciones *forward-reverse* en la construcción de *contigs* que se producen a menudo por la secuenciación de los dos finales de un subclon. Una restricción *forward-reverse* específica que dos lecturas deberían estar en hebras opuestas de una molécula de ADN dentro de un rango de distancia específico. Al secuenciar las dos terminaciones de cada *subclon*, se producen un gran número de restricciones *forward-reverse* de las cuales algunas son incorrectas debido a errores en el *lane tracking* y en la clonación. La estrategia de **CAP3** para hacer frente a esta dificultad se basa en la observación de que la mayoría de las restricciones son correctas y el resto suele ocurrir aleatoriamente. Por lo tanto, algunas restricciones que no satisfacen un *contig* no pueden ser suficientes para indicar un error en el montaje de dicho *contig*. Sin embargo, si un número suficiente de todas las restricciones son incompatibles con una combinación en un contig, es probable que dicha unión sea un error por lo que hay que buscar alternativas.

El algoritmo de ensamblaje consiste de tres pases principales. En la primara fase se identifican y eliminan las regiones pobres terminales (5' y 3') de cada lectura. Se calculan los solapamientos entre lecturas, se identifican y se eliminan los falsos. En la segunda fase, se unen las lecturas para formar *contigs* en orden decreciente de puntuaciones de solapamientos. Después, se utilizan las restricciones *forward-reverse* para realizar correcciones en los *contigs*. En la tercera fase se construye un alineamiento múltiple de secuencias de lecturas y una secuencia consenso teniendo en cuenta los valores de calidad de cada base calculados para cada *contig*. Los valores base de calidad se utilizan en el cálculo de solapamientos y en la construcción de alineamientos múltiples de secuencias.

5.10. CELERA ASSEMBLER

CELERA ASSEMBLER [37, 79] es un software científico para desarrollo biológico. Es un ensamblador de secuencias de ADN generadas por perdigonada para genomas completos (WGS) *de novo*. Reconstruye secuencias largas de ADN desde datos fragmentados generados por un proceso de secuenciación por perdigonada de genoma completo. Fue inicialmente desarrollado por Gene Myers, Granger Sutton, Art L. Delcher y otros en Celera Genomics desde 1998 y versionado como software libre en 2004 como el **wgs-assembler** bajo licencia GNU [22]. El **CELERA ASSEMBLER** ha contribuido de manera importante al avance de la Genómica, incluyendo la primera secuencia completa por perdigonada de un genoma de un organismo multi-celular [85] y la primera secuencia diploide de un individuo humano [89]. El software del **CELERA ASSEMBLER** es un sistema modular compuesto de múltiples programas que interactúan a través de interfaces bien definidas de tal manera que se pueden modificar el orden secuencias de los programas y sus parámetros para cubrir necesidades específicas. Está escrito principalmente en lenguaje C para sistemas operativos Unix. Funciona con secuenciadores tipo Sanger tal como el ABI 3730. El software se mantiene gracias a la aportación del JCVI (J. Craig Venter Institute), el cual mantiene y mejora el código fuente en colaboración con científicos de la Universidad de Maryland.

Los algoritmos del **CELERA ASSEMBLER** (o *wgs-assembler*) dependen de tres puntos para su correcto funcionamiento: la construcción de *unitigs* (*contigs* que se ensamblan de manera unitaria, sin información conflictiva), la identificación *unitigs* únicos (aquellos que son copias únicas en el genoma) y suficientes

emparejamientos de los pares (*mate pair*) que conecten los *unitigs* únicos.

A partir de la versión 5 (actualmente está en la versión 6.1 que se liberó en Abril del 2010) el **CELERA ASSEMBLER** se denomina **CABOG** (**CELERA ASSEMBLER***'s Best Overlap Graph*) y es la versión que dispone de soporte para la plataforma FLX de 454 (pirosecuenciación). **CABOG** está descrito en *Bioinformatics* [80]. Se puede considerar que **CABOG** funciona de manera similar a **NEWBLER**, tanto para conjuntos de bacterias como en lecturas no emparejadas. Según información directa del **CELERA ASSEMBLER** se observa que en conjuntos de datos que contienen tanto FLX como extremos apareados (*paired-end mates*) de cualquier fuente (FLX o Sanger) **CABOG** produce *contigs* y *supercontigs* más largos que cualquier otro software similar.

CABOG reduce los problemas con las lecturas homopolímeras, es decir, repeticiones de bases únicas, para ello, **CABOG** construye *unitigs* iniciales que excluyen lecturas que son subcadenas de otras lecturas e intenta evitar inicialmente las lecturas de estas subcadenas ya que son más susceptibles a repeticiones que inducen a solapamientos falsos.

CABOG aplica un esquema para corregir bases que se describe por primera vez en **ARACHNE** [12], para ello compara cada lectura con su conjunto de lecturas solapadas y se detectan errores de secuenciación. Lo que hace es no corregir la lectura sino que asigna un ratio de error en solapamientos que abarquen el error inferido. Posteriormente aplica un umbral definido por el usuario para estos ratios de error y de los solapamientos que superen el filtro de error y un filtro para una mínima longitud de alineamientos, **CABOG** selecciona el «mejor» solapamiento por lectura. El «mejor» se define como el que alinea más bases. Se estima que este filtro de «mejor solapamiento» elimina muchos de los solapamientos eliminados por el algoritmo de eliminación de enlaces del grafo que es más costoso computacionalmente [85] y que se utilizó en la versión original del **CELERA ASSEMBLER**. **CABOG** construye un grafo de solapamientos de lecturas y de mejores solapamientos. Dentro del grafo, se construyen *unitigs* de los caminos simples máximos que están libres de ramas e intersecciones. Posteriormente **CABOG** construye un grafo de *unitigs* más las restricciones de lecturas emparejadas y dentro de ese grafo une los *unitigs* en *contigs* y los conecta en *supercontigs*. También aplica una serie de reducciones al grafo incluyendo la eliminación de enlaces que se infieren transitivamente. Finalmente, **CABOG** genera las secuencias consenso mediante el cálculo de alineamiento múltiple de secuencias a partir del diseño de los *supercontigs* más las secuencias de lectura.

5.11. EDENA

EDENA (Exact *De novo* Assembler) [31, 64] es un ensamblador para lecturas cortas orientado al Genome Analyzer de Illumina. Está desarrollado en el Genomic Research Laboratory de Ginebra en Suiza. Está basado en el paradigma de diseño por solapamiento de tal manera que todos los solapamientos exactos detectados entre parejas de lecturas se estructuran en un grafo (fase de solapamiento), posteriormente las lecturas se indexan en una matriz y los solapamientos se marcan mediante búsquedas dicotómicas en dicha matriz. De esta manera en el grafo generado se eliminan los enlaces transitivos y detectados como falsos (fase de diseño). Finalmente se genera como salida los *contigs* que pueden ensamblarse siguiendo un camino no ambiguo en el grafo. **EDENA** es capaz de generar *contigs* bacterianos de varios miles de bases con cobertura cercana al total.

5.12. ELAND

ELAND (Efficient *Large-Scale Alignment of Nucleotide Databases*) [5] ha sido desarrollado por Anthony J. Cox (Solexa) y forma parte del software de Illumina. Es un algoritmo para lecturas cortas de programación dinámica orientado a identificar (mapping) secuencias de un genoma y contrastarlo contra un genoma de referencia. Para ello se basa en tres puntos principales: dada una secuencia de longitud N, se puede dividir en cuatro subsecuencias (A, b, C y D) de longitud similar. Si se asume que no hay más de dos errores en la secuencia inicial, se deduce que, al menos, dos de la subsecuencias están libres de errores. A partir de aquí se construyen la seis subsecuencias formadas por parejas de dos subsecuencias (AB, CD, AC, BD, AD, BC) y se buscan en la base de datos que contiene todas las secuencias del genoma. Para acelerar el procedimiento se utilizan lecturas indexadas en tablas *hash* mediante operaciones binarias.

ELAND identifica secuencias de hasta 32pb y trabaja con secuencias de la misma longitud. Adicionalmente tiene otras limitaciones como son que alinea secuencias que tengan como máximo dos errores (las que tengan más se ignoran) y devuelve secuencias que coinciden en una única posición del genoma.

5.13. EULER

EULER [92] es un algoritmo para ensamblaje de secuencias cortas y de los primeros que utiliza un camino euleriano implementado mediante un grafo de Bruijn. Según sus autores tiene la ventaja con respecto a los mecanismos de diseño por solapamiento de que es mucho más eficaz con las repeticiones ya que en vez de enmascararlas genera un grafo que permite crear una estructura de repeticiones del genoma para poder

tratarlas. El software **EULER** fue desarrollado para lecturas Sanger [88, 89, 88] y fue posteriormente modificado y denominado **EULER**-SR [93, 94] para lecturas cortas de 454 GS20 [87], lecturas cortas no emparejadas de Illumina/Solexa [88] y emparejadas finales de Solexa [88].

EULER se basa en tres principios fundamentales:

- Representa las lecturas como enlaces y los solapamientos como nodos en un grafo de Bruijn.
- Realiza un ensamblaje eficiente mediante la reducción a un problema de camino euleriano: cada enlace debe visitarse una sola vez.
- Las repeticiones se tratan mediante el uso de múltiples enlaces para una lectura repetida.

Antes de construir el grafo **EULER** aplica un filtro a las lecturas que detecta bases erróneas teniendo en cuenta *K-meros* que se dan de manera poco frecuente. El filtro se basa en la redundancia de las lecturas: la mayoría de los *K-meros* correctos deben estar más repetidos en las lecturas. El filtro también se basa en la aletoriedad de los errores de secuenciación: para cualquier K donde 4^k supera en dos veces el tamaño del genoma, la mayoría de los *K-meros* deben ser únicos. El filtro **EULER** se implementa con una lista de *K-meros* y sus frecuencias correspondientes en las lecturas. El filtro excluye o corrige *K-meros* con baja frecuencia de ocurrencia ya que la corrección es especialmente importante para lecturas cortas con un error y cobertura altos. Esta corrección reduce el número total de *K-meros* y, por tanto, el número de nodos del grafo, aunque también es cierto que puede generar problemas adicionales:

- tiene el riesgo de enmascarar polimorfismos [89]
- puede invalidar *K-meros* correctos porque habían tenido una baja cobertura
- puede dejar como válida una lectura incorrecta porque aparezcan *K-meros* que ocurren en varias lecturas pero nunca juntos en una sola lectura.

Para solucionar estos problemas los ensambladores que utilizan *Overlap-Layout-Consensus* tienen un sistema de corrección de bases erróneas similar a este pero utilizan solapamientos en vez de *K-meros*.

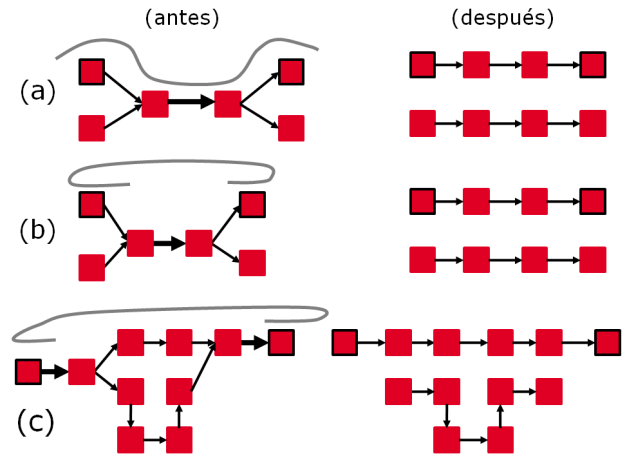


Figura 22: Tres métodos para resolver la complejidad de grafos. (a) El hilo de lectura une caminos a través de repeticiones terminales colapsadas que son más cortas que las longitudes de las lecturas. (b) Los hilos de nodos apareados une caminos a través de repeticiones colapsadas que son más cortas que las distancias de los finales emparejados (*paired-end*). (c) Se escoge un camino cuando la longitud se ajusta a la restricción de los finales emparejados (*paired-end*). No todos los enredos se pueden resolver con lecturas y emparejamientos. Los caminos sin ramas son ilustrativos y pueden simplificarse con enlaces o nodos simples [87].

El proceso de filtrado de **EULER** se denomina alineamiento espectral [89] y lo que hace es identificar errores de secuenciación comparando los *K-meros* contenidos entre lecturas individuales y todas las lecturas y desconfía de *K-meros* de lecturas individuales cuya frecuencia en todas las lecturas es inferior a un umbral escogido tras el cálculo de las frecuencis de distribución (normalmente bi-modal) de *K-meros* presentes en las lecturas. El primer pico representa los *K-meros* que ocurren una o dos veces debido a errores de secuenciación (o baja cobertura). El segundo pico representa los *K-meros* redundantes inducidos por la cobertura de lecturas (o repeticiones). **EULER** selecciona el umbral entre los dos picos y etiqueta todos los *K-mer* como «buenos» o «malos». Posteriormente examina cada lectura y para la que tiene *K-meros* «malos» ejecuta un algoritmo que sustituye las bases y reduce el número de estos *K-mer* [88]. Finalmente puede aceptar o rechazar la lectura corregida (aunque lecturas rechazadas se pueden introducir posteriormente para ayudar tras el ensamblaje en regiones con poca cobertura). Hay que notar que **EULER** corrige errores de sustitución pero no inserciones ni *deleciones*, es decir, *indels*; aunque es cierto que las sustituciones son el tipo de error más común en datos Solexa [78].

Una vez en este punto, **EULER** construye un grafo de *K-mer* a partir de las lecturas filtradas y corregidas; y aplica una serie de manipulaciones al grafo para

evitar los efectos de los errores de secuenciación y las repeticiones.

Como se procesan *K-meros* y no lecturas, la construcción del grafo descarta mucha información continua de las lecturas por lo que **EULER** tiene que reparar este defecto realizando un seguimiento de las lecturas a través del grafo (ya que asignar una lectura en el grafo es relativamente fácil). Posteriormente, en esta fase, los *K-meros* de las lecturas se asignan a nodos únicos y las lecturas se constituyen por caminos. A partir de aquí, la explotación del grafo es más compleja ya que las lecturas que terminan dentro de una repetición no son consistentes con ningún camino que tenga una salida de la repetición, pero las lecturas que superen una repetición sí que son consistentes con caminos menores. Por otro lado, el seguimiento de la lectura a través del grafo genera una cadena de secuencias del patrón típico «cuerda deshilachada» con lo que se puede resolver una repetición (Figura 22a) por lo que el seguimiento de las lecturas restringe el conjunto de caminos válidos en el grafo permitiendo la resolución de repeticiones cuya longitud está entre K y la longitud de las lecturas.

Teniendo en cuenta que una lectura emparejada es una lectura larga que ha perdido alguna base en medio de su secuencia, **EULER** utiliza estas lecturas emparejadas para resolver repeticiones más largas que las propias lecturas individuales. La técnica se podría denominar «seguimiento de parejas» (*mate-threading*) ya que los finales emparejados que expanden una repetición proporcionan una evidencia para unir un camino que entra en una repetición a un camino que sale de la misma repetición (Figura 22b). Los finales emparejados pueden resolver también algunos enredos complejos inducidos por repeticiones.

Un grafo complejo puede tener múltiples caminos entre dos nodos que se corresponden a los finales opuestos de emparejamientos (*mate pair*). Cada camino implica una secuencia de ADN válida y en muchos casos solo uno de los caminos implica una secuencia cuya longitud satisface las restricciones de finales emparejados (Figura 22c). Entre cualquier para emparejado, podrían darse demasiados caminos para que la búsqueda sea factible y **EULER** intenta restringir el espacio de búsqueda utilizando restricciones de emparejamientos como límites de la longitud del camino. Hay que tener en cuenta que el número de caminos en un grafo general escala con N^E para N nodos y E enlaces por nodo. Los grafos de secuencias de ADN con *K-mer* pueden restringirse a $E \leq 4$ para representar las cuatro extensiones (bases) posibles de una secuencia y aunque esta restricción puede alterarse utilizando ciertas simplificaciones en el grafo, N^E sigue siendo no resoluble computacionalmente.

Una vez definidos los caminos, **EULER** implementa ciertas simplificaciones al grafo en regiones que tienen, tanto alta como baja, cobertura en las lecturas y la eliminación de ramales cortos o espolones *(spurs*) reduce el conjunto de ramas y caminos del grafo por lo que se generan caminos simples pero más largos. Los ramales cortos son presumiblemente debidos a errores de secuenciación que han sobrevivido al filtro de alineamiento espectral.

Varias plataformas generan baja calidad en los finales 3' e intentan solucionar el problema mediante protocolos basados en lecturas largas. **EULER** plantea la resolución al problema enfocándose en los prefijos confiables (cuya longitud es variable por lectura) de las lecturas en vez de en sus sufijos, escogiéndolos durante la fase de corrección de errores. Durante el seguimiento de las lecturas para su asignación, **EULER** asigna los prefijos y los sufijos a diferentes caminos del grafo en función de una heurística propia y como los sufijos añaden cobertura a múltiples alineamientos de secuencias, también añaden mayor conectividad al grafo por lo que este mayor número de secuencias conduce a obtener una tamaño mayor de *contig*. Por otro lado **EULER** no utiliza los sufijos para alterar la secuencia consenso sino sólo contribuyen a la conectividad.

Al igual que los grafos de solapamiento son sensitivos al umbral de la longitud del solapamiento mínimo, los grafos de *K-meros* son sensitivos al parámetro K. Valores mayores de K resuelven repeticiones más largas pero también fracturan los ensamblajes en regiones de baja cobertura de lecturas. **EULER** intentan solucionar esto con una heurística construyendo y simplificando dos grafos de *K-meros* con diferentes valores de K, identificando enlaces en el grafo de K pequeño que se pierden en el grafo de K mayor. Esto añade los correspondientes pseudo-enlaces al segundo grafo extendiendo caminos en este último y generando *contigs* mayores. Esta técnica usa de manera efectiva grandes *K-meros* para construir *contigs* iniciales fiables y poder rellenar huecos con el mayor número de *K-meros* pequeños, lo cual es análogo al relleno de huecos utilizado por los ensambladores que utilizan en enfoque solapamiento-diseño-consenso [85].

Algún software **EULER** incorpora una estructura adicional denominada grafo A-Bruijn cuyo nombre proviene de la combinación del grafo de de Bruijn y una matriz de adyacencia (contigua) donde los nodos del grafo representan columnas consecutivas de alineamientos de secuencias múltiples. Comparado con los nodos de *K-meros* que representan lecturas individuales, los nodos de adyacencia pueden ser menos sensitivos a los errores de secuenciación y han sido desarrollados [88] para poder clasificar repeticiones dentro del grafo.

En resumen **EULER** utiliza varios grafos de de Bruijn con distintos tamaños de *K-meros* realizando comparaciones entre ellos, aplicando heurísticas para simplificar la complejidad de los grafos inducida por los errores de secuenciación y explotando las restricciones de los finales de lecturas con baja calidad y los finales emparejados para reducir los «enredos» inducidos por las repeticiones. El objetivo por el cual se desarrolló el software fue el ensamblaje *de novo* para lecturas cortas de la plataforma Solexa incluyendo finales emparejados (*paired-ends*).

5.14. MAQ

MAQ [32] son las iniciales de *Mapping and Assembly with Quality* y ensambla mediante un proceso de identificación lecturas cortas contra secuencias de referencia. El programa está específicamente diseñado para Illumina-Solexa/AB-SOLiD pero no para 454, fue desarrollado por Heng Li del centro Sanger en C++ y utiliza secuencias en formato FASTA. **MAQ** proporciona una puntuación de calidad (*mapping quality*) a cada alineamiento, de manera similar a como lo hace *Phred*. **MAQ** es un proyecto hospedado como software de código abierto en Sourceforge.net. La página del proyecto está disponible en [32] y anteriormente se conocía como **MAPASS2**.

El funcionamiento de **MAQ** se basa en indexar todas las lecturas de entrada y escanearlas contra la secuencia de referencia durante varios ciclos, almacenando los dos mejores resultados en memoria. Posteriormente identifica la posición donde la suma de las puntuaciones de calidad de los nucleótidos no coinicidentes es mínima. En el caso de que se varias posiciones con la misma puntuación, **MAQ** elige una de ellas de manera aleatoria. El resultado final es un identificador de calidad por cada alineamiento de lectura.

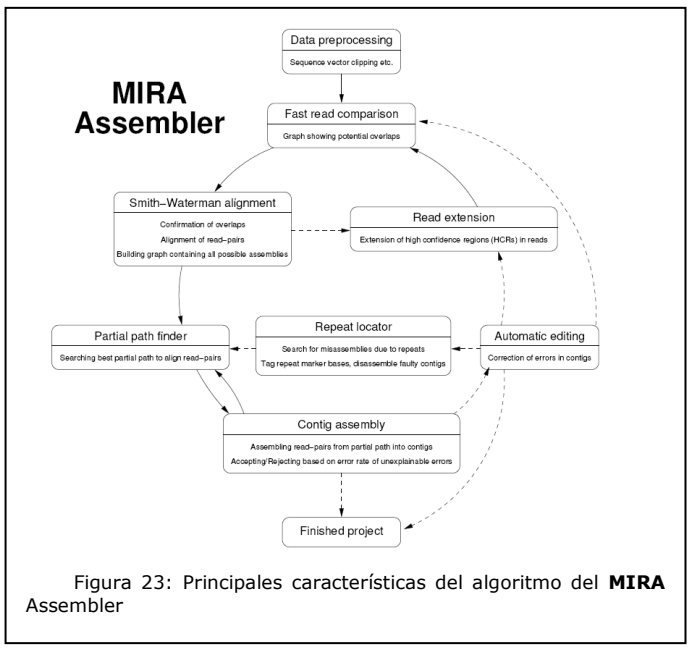


Figura 23: Principales características del algoritmo del **MIRA** Assembler

MAQ soporta un máximo de 63pb de longitud de lecturas.

5.15. MIRA3

MIRA3 [68, 8] (*Mimicking Intelligent Read Assembly*) es un ensamblador de secuencias de genoma y ESTs [8] para Sanger, 454 e Illumina. Ha sido desarrollado por Bastien Chevreux et al. que lo introdujo a través de una tesis [100] en 1997 y se encuentra actualmente en su versión 3.2.1. Desde 2007 está publicado bajo licencia de código abierto.

Las características (Figura 23) más importantes de **MIRA3** son, entre otras, las siguientes:

- Es capaz de ensamblar genomas completos. Se han hecho pruebas con hasta 50 megabases.
- Diseñado para ensamblaje *de novo* y asignación (*mapping*) con Sanger, 454 y Solexa o cualquiera combinación de estas.
- Soporta ensamblaje de ESTs [9]: detección de SNPs; ensamblaje de transcritos por cepas, rutinas especiales para cobertura extrema que permite ensamblaje de familias de genes con miles de secuencias similares.
- Soporte para plantilla de clones / secuenciación de extremos apareados (*paired-end*).
- Utiliza indicadores de calidad para mejorar el consenso resultante.
- Incorpora editores de secuencia automática con gestión de errores para datos Sanger y 454.
- Orientado a un fácil uso y con distintos parámetros para predefinir tareas.
- Incorpora rutinas de preprocesamiento de datos. Soporta diversos formatos de entrada como FASTA, FASTQ, CAF, EXP, PHD y etiquetados de Poly-A al final de las secuencias EST.
- Soporta diversos formatos de salida: FASTA (con indicadores de calidad), ACE, CAF, ensamblaje directo gap4, GenBank (sólo ensamblajes de referencia), HTML y otros.
- Genera diversos ficheros de salida para genera estadísticas, información del ensamblaje y el propio ensamblaje.
- Soporta la carga de ficheros GenBank (gbf/gbff/gbk).
- Soporte multiprocesador.

5.16. MOSAIK

MOSAIK [49] es un ensamblador para identificación mediante un genoma de referencia que está basado modularmente en cuatro programas:

- **MOSAIK**Build
- **MOSAIK**Aligner
- **MOSAIK**Sort
- **MOSAIK**Assembler

MOSAIKBuild convierte varios formatos de secuencias en un formato nativo para **MOSAIK**. **MOSAIK**Aligner empareja lecturas contra secuencias de referencia. **MOSAIK**Sort resuelve las lecturas *paired-end* y ordena los alineamientos por las coordenadas de la secuencia de referencia. Para terminar, **MOSAIK**Assembler analiza los alineamientos ordenados y genera un una secuencia de alineamientos múltiple que se guarda en un formato de ensamblaje.

De esta manera, se puede decir que el flujo de trabajo consiste en proporcionar secuencias de entrada en formato FASTA, FASTQ, Illumina Bustard & Gerald o SRF y producir ficheros de ensamblaje (en formatos **PHRAP, ace** y **gigabayes**) que pueden ser visionadas mediante herramientas tales como **EagleView**.

Actualmente **MOSAIK** tiene licencia de código abierto y está alojado en Google [50]

Actualmente se han añadido diversas características para mejorarlo:

- Para hacerlo más rápido se ha introducido una implementación del algoritmo Smith-Waterman por lo que las lecturas 454 se alinean bastante más rápido.
- Se utilizan modelos de regresión para incrementar la precisión y la utilidad de los indicadores de calidad que genera.
- Se ha incorporado una opción de búsqueda de alineación.
- Se soporta SOLiD para lo cual importa y alinea lecturas SOLiD en *colorspace.*
- Soporta los formatos de alineación SAM y BAM.

MOSAIK está escrito en C++ y soportado en las siguientes plataforma: Microsoft Windows, Apple MAC OS S, FreeBSD y Linux.

Sus características de desarrollo hacen que esté orientado a la búsqueda de SNPs e *indels*.

5.17. NEWBLER

NEWBLER [48, 87] es un paquete de software comercial para ensamblaje de secuencias *de novo*. Realmente es el Genome Sequencer *de novo* para plataformas 454 (Roche) para lo que acepta:

- Lecturas estándar 454 FLX y 454 Titanium (más largas).
- lecturas únicas (*single*) y *paired-end*.
- Y opcionalmente puede incluir lecturas Sanger.

Se ejecuta en plataformas Linux de 32 y 64 bits tanto mediante línea de comandos como mediante una interfaz Java (trabaja directamente con los datos .SFF generados por el secuenciador) y se conoce más como **gsAssembler** en la documentación de Roche.

Es uno de los primeros que utiliza el enfoque Solapamiento/Diseño/Consenso. La primera versión emparejaba lecturas de aproximadamente 100pb que generaba el secuenciador GS 20. Desde entonces, el ensamblador **NEWBLER** se ha revisado para construir *supercontigs* con restricciones de lecturas emparejadas. En 2005 se generó una nueva versión de **NEWBLER** que implementa dos OLC de tal manera que en la primera fase genera *unitigs* a partir de las lecturas [85]. Estos *unitigs* sirven de manera preliminar pero fiable como semillas para el resto del proceso de ensamblaje. En la segunda fase OLC genera *contigs* más largos a partir de los unitigs iniciales uniéndolos mediante un diseño basado en solapamientos emparejados. Igualmente es posible partir los *unitigs* si el prefijo y el sufijo se alinean con diferentes *contigs*. Esta división de unitigs puede, a su vez, dividir lecturas (que normalmente vienen de repeticiones) lo cual conduce a situar en varios *contigs*.

NEWBLER hace uso de la cobertura para eliminar errores provenientes de la secuenciación usando métricas para poder detectar lecturas inexactas provenientes de repeticiones homopolímeras.

El paquete **NEWBLER** ofrece funcionalidades adicionales al ensamblaje *de novo* e incluye una guía de usuario bastante extensa. Las versiones iniciales se enfocaban en ensamblaje de lecturas genómicas pero a partir de la versión 2.3 incluye optimizaciones para datos transcriptómicos. El software se distribuye con los secuenciadores de 454, actualizándose cada cierto tiempo y la descripción de estas versiones indica que el software actual difiere bastante del algoritmo publicado. El software no está disponible.

5.18. PASS

PASS (A Program to Align Short Sequences) [30] es un programa que alinea secuencias cortas de ADN contra una secuencia genómica de referencia. Está diseñado para manipular grandes cantidades de lecturas como las generadas por las tecnologías de Solexa, SOLiD o 454. El algoritmo está basado en una estructura de datos que mantiene en RAM el índice de las puntuaciones genómicas de palabras semilla

(normalmente 11 o 12 bases) así como otro índice de las puntuaciones precalculadas de palabras cortas (normalmente siete y ocho bases) alineadas unas contra otras. Después de construir el índice genómico, el programa explora cada secuencia realizando tres pasos: (1) busca palabas semilla que coincidan en el genoma; (2) para cada coincidencia comprueba los alineamientos precalculados de los bordes de las regiones; (3) si se pasa el paso 2 entonces se ejecuta un alineamiento dinámico exacto de una región restringida alrededor de la coincidencia.

El programa está disponible [51] como código abierto, está desarrollado en C++ y soportado en Linux y Windows.

5.19. **PCAP**

PCAP (Paralell Contig Assembly Program) [145] es un programa para ensamblaje de genoma completo que posee varias características orientadas a la eficiencia y la exactitud del ensamblaje. Es multiprocesador y utiliza un método para evitar solapamientos que se pueden perder debido a errores de secuenciación, detecta regiones repetidas en forma de solapamientos con otras lecturas (en vez de encontrar muchas coincidencias cortas entre lecturas). Es capaz de identificar y eliminar regiones de lecturas finales contaminadas y la generación de la secuencia consenso para un *contig* se basa en una alineación de lecturas en el *contig* y tanto los indicadores de calidad como la información de cobertura se utilizan para determinar cada consenso.

El programa **PCAP** ha sido probado en un conjunto de datos de un genoma completo de ratón consistente en 30 millones de lecturas y en el cromosoma 20 humano con 1,7 millones de lecturas. El programa está disponible gratuitamente para uso académico.

El algoritmo utilizado consiste en dos fases principales:

1- En la primera fase se identifican las regiones de lecturas repetidas y se calculan los solapamientos entre lecturas.
2- En la segunda fase se identifican y eliminan las regiones pobres para cada lectura. Las lecturas se ensamblan en *contigs* usando solapamientos únicos y se corrigen y enlazan los *contigs* en armazones (*scaffolds*) con restricciones. Se construye una secuencia de alineamientos de lecturas múltiple y se genera una secuencia consenso para cada *contig*.

5.20. **PHRAP**

PHRAP [55] es un algoritmo diseñado para ensamblar secuencias por perdigonada (*shotgun*). Fue desarrollado por Phil Green en 1999 y es ampliamente utilizado en proyectos de ensamblaje de secuencias. Es un algoritmo que soporta lecturas largas y fue diseñado para ensamblar secuencias Sanger e implementa el enfoque de diseño por consenso de solapamiento. **PHRAP** utiliza el algoritmo de alineamiento de secuencias de Smith-Waterman, para comparar todas las secuencias y encontrar pares que coincidan en subsecuencias. Tras esto, genera puntuaciones basadas en las lecturas y en las alineaciones anteriores. El proceso del ensamblaje de secuencias de **PHRAP** se describe en la siguiente figura. **PHRAP** preensambla lecturas en grupos antes de mezclarlas en grupops dentro de una secuencia *contig*, lo cual puede ser util para reducir el riesgo de unir lecturas incorrectamente como resultados de repeticiones (secuencias repetidas).

Las claves de **PHRAP** son las siguientes:

- Combina información de calidad proporcionada por el usuario que se computa internamente para mejorar la precisión de los *contigs* producidos.
- Construye secuencias de *contig* basadas en una superposición de partes de lecturas con más alta puntuación (esto aplica principalmente cuando la calidad de los datos se suministra al comienzo del proceso de ensamblaje).
- Proporciona información exhaustiva sobre el ensamblaje para ayudar en la resolución de problemas.
- Habilidad para manipular grandes conjuntos de datos, y
- Portabilidad entre varios S.O: Unix/Linux, Mac OSX y MS

5.21. **PHUSION**

El ensamblador **PHUSION** [103] está basado en el enfoque de consenso por solapamiento y es el ensamblador del Sanger Center. **PHUSION** es modular y se basa en **PHRAP** para el ensamblaje a bajo nivel y utiliza diversos algoritmos para reducir la complejidad de los datos generados por **PHRAP** y corregir errores producidos por este.

El sistema se basa en las siguientes fases:

1- Preparación de datos: detección de contaminación, vectores y calidad del recorte.
2- Agrupación de lecturas: las lecturas que comparten un bajo número de copias de

«palabras» se agrupan juntas en *clusters* que pueden ensamblarse independientemente.

3- R**PHRAP**: Cada cluster de lecturas y sus lecturas asociadas se ensamblan utilizando **PHRAP** de una forma iterativa que permite que los *contigs* se extiendan o rompan utilizando información de las lecturas.
4- RPjoin: Agrupa *contigs* basándose en lecturas compartidas, solapamientos de secuencias en información de las lecturas.
5- RPono: Construye un armazón de *contigs* basándose en la información de las lecturas.
6- Filtrado de contaminación: Se rechazan *contigs* basándose en ciertos desequilibrios de las lecturas del patrón original.

5.22. SEQMAP

SEQMAP [56] es una herramienta para la identificación de grandes cantidades de secuencias cortas en un genoma de referencia. Utiliza FASTA como formato de entrada y puede generar la salida en varios formatos (tal como **ELAND**).

El algoritmo base utilizado por **SEQMAP** se basa en el «principio de encasillamiento» (tal como los ensambladores **ELAND**, **SOAP** y RMAP) que se utiliza para trocear cada lectura en varias partes. Posteriormente algunas partes de las lecturas se pueden referenciar directamente contra la referencia y otras pueden ser descartadas rápidamente.

SEQMAP está escrito en C++, utiliza operaciones binarias para acelerar la identificación de tal manera que cada nucleótido está codificad mediante dos bits en memoria. Tal como lo hacen **ELAND** y RMAP, **SEQMAP** indexa y genera un *hash* para cada lectura antes de referenciarla contra el genoma, lo cual es diferente a como lo realizan otros programas tales como BLAT y **SOAP** donde lo que se indexa y se realiza el *hash* correspondiente es en el propio genoma de referencia. La justificación de esto se basa en que la indexación del propio genoma consume más recursos de memoria y **SEQMAP** tiene como objetivo poder ejecutarse en sistemas cercanos a PC de usuario.

SEQMAP es de libre distribución y actualmente está en la versión 1.0.13 generada en Enero de 2.009.

5.23. SHARCGS

SHARCGS [36] (*Short read Assembler based on Robust Contig extensión for Genome Sequencing*) es un programa para la generación de *contigs* de secuencias genómicas basado en *K-meros* de longitudes comprendidas entre 25 y 40 que elimina errores producidos en el proceso de secuenciación. Los datos de entrada consisten en lecturas de tamaño similar que pueden contener un máximo de un 2% de errores.

El algoritmo filtra los datos de entrada quedándose con las lecturas que cumplen el siguiente criterio: todas las lecturas tienen extremos que se solapan con otras. Cuando se proporcionan indicadores de calidad en el fichero de entrada el primer paso del filtrado se modifica de tal manera que una lectura se lee varias veces sólo se almacena una vez (guardando el indicador de calidad más alto). Si dos lecturas son complementos inversos entre sí se les suma su indicador de calidad de la otra lectura.

El algoritmo de ensamblaje asume que sólo pasan el primer filtro lecturas correctas a pesar de la falta de fiabilidad de los datos originales. El algoritmo pasa entonces a generar *contigs* con lecturas que se unen sin ambigüedades. Si no existe ninguna lectura para una posición concreta se genera un hueco (*gap*) que puede consistir de varias posiciones en una fila para la cual ninguna lectura ha pasado el filtro.

SHARCGS puede ejecutar varios ciclos del algoritmo principal cada uno de ellos con distintos parámetros y el ensamblaje final se almacena en un fichero que contiene una entrada en formato FASTA por cada contig generado.

La implementación de **SHARCGS** está realizada en Perl y se han realizado pruebas para ensamblar varios genomas (Figura 24), entre ellos una BAC de 100kpb que tarda menos de 15 minutos en un procesador Intel Xeon a 2,8Ghz (32 bits) usando menos de 1GB de RAM. El ensamblaje de una E. coli tarda menos de 10 usando 24Gb de RAM en un AMD Opteron a 2,8Ghz (64 bits).

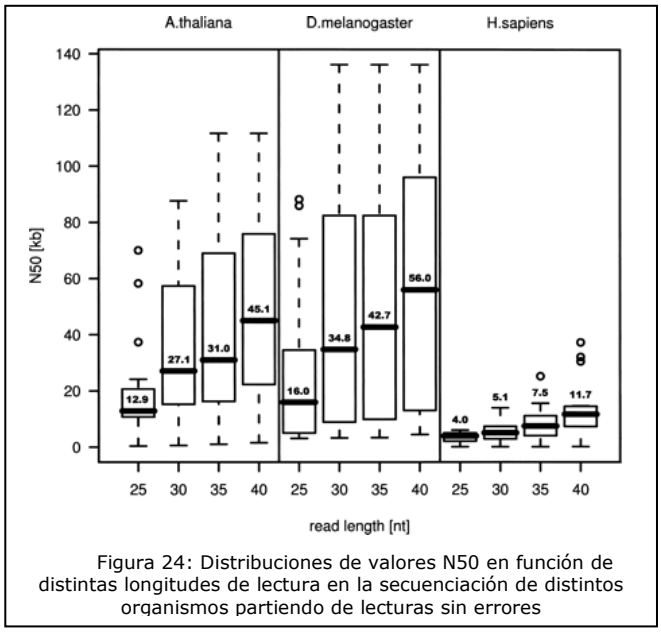


Figura 24: Distribuciones de valores N50 en función de distintas longitudes de lectura en la secuenciación de distintos organismos partiendo de lecturas sin errores

Los autores compararon mediante simulación (faltando lecturas y con errores de secuenciación) el algoritmo con **SSAKE** [89] y observaron que este último no pudo alinear al 25% de los *contigs* contra la secuencia BAC de referencia mientras que **SHARCGS** alineó el 100%. También se comparó con el programa **EULER2** [92] (sin una adaptación precisa de este para el experimento) pero observaron que cuando existen muchos errores de secuenciación **EULER2** genera un grafo de Bruijn excesivamente complejo que produce problemas de rendimiento aunque los resultados obtenidos mostraron que los *contigs* generados por **EULER**2 cubrían entre el 33% y el 95% de la secuencia objetivo con un N50 entre 53pb y 51kpb.

5.24. SHORTY

SHORTY [72, 88] es un programa para el ensamblaje de lecturas cortas mediante la construcción de un grafo de Bruijn que está actualmente en revisión y se basa en las siguientes fases:

- Filtrado de los pares de lecturas de entrada para corregir errores de secuenciación mediante un análisis de frecuencia y corrección de las lecturas consenso.
- Construcción de un subgrafo de Bruijn de lecturas «a la izquierda» así como de su grupo asociado de lecturas «a la derecha»
- Construcción del subrafo opuesto
- Selección de los *contigs* de suficiente tamaño mediante distintas heurísticas
- Extensión de los *contigs* mediante un algoritmo de post-ensamblaje

SHORTY está desarrollado en C++ y es de libre distribución.

5.25. SOAP*DENOVO*

SOAP*denovo* [52, 122] (Short Oligonucleotide Analysis Package) es un ensamblador para lecturas cortas *de novo* que tiene como objetivo el ensamblaje de oligonucleótidos en *contigs* y *supercontigs* utilizando un algoritmo (Figura 25) de ensamblaje paralelizable basado en grafos de Bruijn aunque a diferencia de **Velvet** la corrección de errores se realiza antes de construir el grafo. Está preparado para ensamblar grandes genomas y está especialmente diseñado para ensamblar miles de millones de lecturas cortas de de 75pb de longitud y menor [123, 124] provenientes del Illumina GA.

Según los autores, el ensamblador se ha probado satisfactoriamente para el ensamblaje de secuencias (Figura 26) provenientes de genomas humanos de asiático (117,7Gb de datos) y africano, consiguiendo un tamaño de *contig* N50 de 7,4kb y 5,9kb respectivamente y un *supercontig* de 446,3kb y 61,9kb respectivamente.

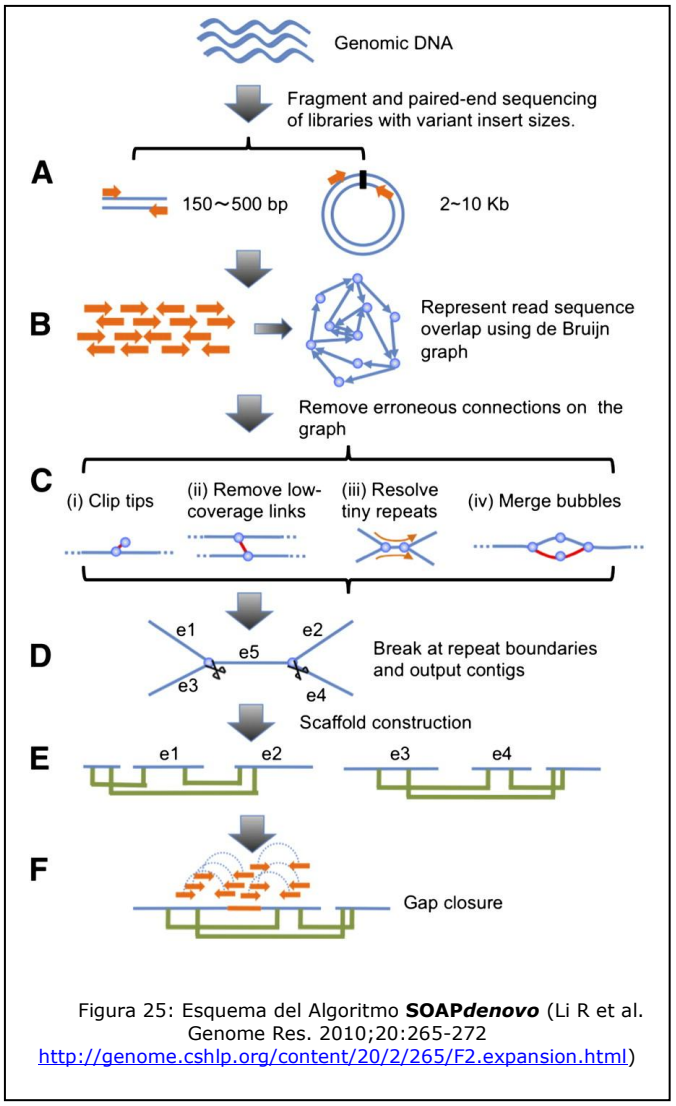


Figura 25: Esquema del Algoritmo **SOAP*denovo*** (Li R et al. Genome Res. 2010;20:265-272 http://genome.cshlp.org/content/20/2/265/F2.expansion.html)

SOAP filtra y corrige las lecturas usando umbrales predefinidos para las frecuencias de *K-mer* construyendo un grafo de de Bruijn y podando las ramas que exceden dichos umbrales. **SOAP** analiza las lecturas y separa los caminos que presentan un patrón simétrico de «cuerda deshilachada». **SOAP** también elimina las burbujas con un algoritmo similar al de **Velvet** con una cobertura mayor que determina el camino a escoger. Aunque la implementación del grafo de de Bruijn que utiliza **SOAP** es prestada de **EULER** y **Velvet**, en este caso es más eficiente ya que no utiliza un seguimiento de lecturas por lo que solo requiere 120 GB RAM para almacenar 5.000 millones de nodos a partir de 3.300 millones de lecturas (después del filtrado).

SOAP construye *contigs* a partir de lecturas mediante el grafo de de Bruijn descartando este grafo para construir los *supercontigs* ya que mapea todas las lecturas emparejadas a las secuencias *contig* consenso incluyendo las lecturas que se usan en el grafo. Posteriormente construye un grafo de *contigs* cuyos enlaces representan las restricciones de los *contigs* emparejados. **SOAP** reduce la complejidad del grafo de *contigs* eliminando enlaces que se pueden inferir transitivamente desde otros y también aísla *contigs* que se recorren por caminos incompatibles. Estas últimas técnicas también se utilizan en **CABOG** y en su predecesor, el **CELERA ASSEMBLER** y al igual que hace **ALLPATHS**, **SOAP** procesa los enlaces en el orden del tamaño del inserto, de menor a mayor. **SOAP** hace esto excluyendo la construcción de *supercontigs* que se intercalan en otros y utiliza lecturas emparejadas para asignar lecturas a huecos entre *contigs* vecinos dentro de un *supercontig*. Esto es similar a las técnicas de **CABOG** denominada «*rocks and stones*» [37, 79] y a las que **Velvet** denomina «migaja de pan» y «*pebble*».

El enfoque de desarrollo del ensamblador y su orientación hacía ensamblaje *de novo* se basa en que la principal dificultad de ensamblaje de lecturas cortas obtenidas por perdigonada en un genoma completo está en la presencia de secuencias repetidas que se presentan de manera idéntica o muy parecida a lo largo del genoma. El estudio del análisis de la estructura y sus repeticiones (idénticas o similares) de un genoma de referencia conocido ayuda de manera inevitable al diseño de la secuenciación objetivo y proporciona, desde el principio, una estimación teórica del ensamblaje esperado.

Para esto, los autores estudiaron detenidamente la estructura del genoma y observaron que en humanos, alrededor de la mitad del genoma se deriva de elementos transponibles (*transponsable elements* TEs) [87] (secuencias que son capaces de transponerse, entendiendo por transposición, al hecho de que una secuencia pueda cambiar de sitio en el genoma). La mayoría de los transposones aparecen mediante selección natural por lo que las copias nueva acumulan mutaciones después de su duplicación por lo que son fácilmente distinguibles de otras copias repetidas. En el análisis del genoma humano alrededor del 79% de las secuencias están compuestas de 25-mers y la distribución de longitud de estas secuencias muestra que alrededor del 47% de repeticiones tienen un tamaño inferior a 1kb, aunque hay dos picos (de 330pb y 6kb) que corresponden a las dos clases de elementos transponibles del genoma humano: los retrotransposones *Alu* y L1, respectivamente. Alrededor del 78% de las agrupaciones se encuentran entre 500pb y 5kb por lo que teóricamente usando un nodo de tamaño 25-mer para ensamblaje tenemos un tamaño del contig N50 de 1,3kb; reduciendo el tamaño a 21-mer obtenemos un contig N50 de 251pb e incrementándolo a 29-mer obtenemos un N50 de 1,9kb. A mayores *K-meros* obtenemos mayores tamaños de *contigs* pero se requiere un mayor profundidad de secuenciación o una mayor longitud de las lecturas para poder garantizar que las lecturas cortas cubren más que un tamaño seleccionado de *K-mer* para cada posición genómica.

Los pasos principales que sigue el algoritmo son los siguientes:

- Corrección de errores en la secuencia preensamblada: este paso es esencial para grandes conjuntos de datos para reducir drásticamente el uso de memoria haciendo posible la carga completa del número de secuencias para construir el grafo de Bruijn. Para pequeños conjuntos de datos, la corrección de errores antes del ensamblaje no

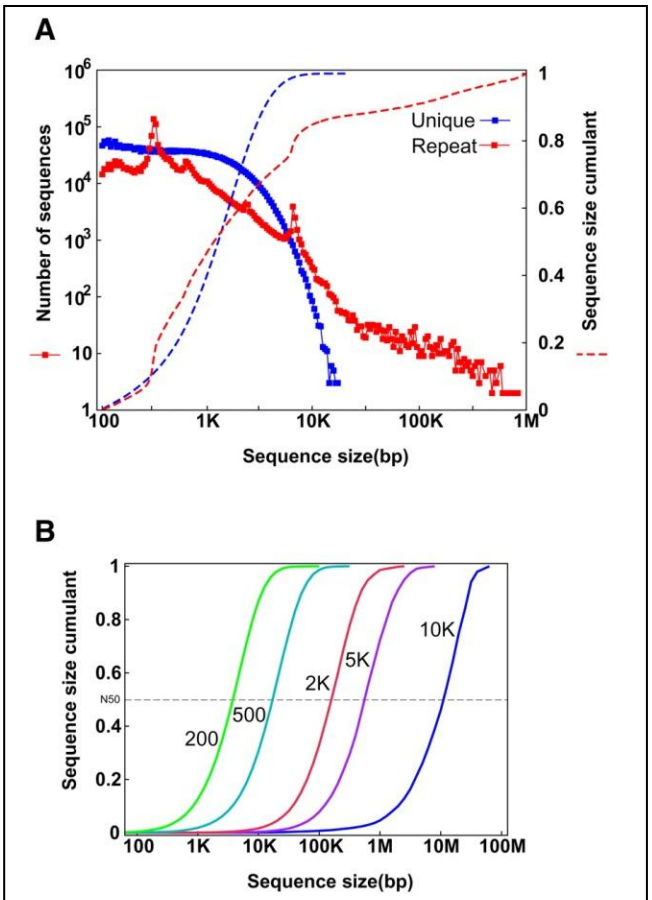


Figura 26: (A) Distribución de la longitud de agrupaciones de secuencias repetidas y únicas (B) Distribución de la longitud de la secuencia de un ensamblaje ideal para cada tamaño de inserto de extremos emparejados (Li R et al. Genome Res. 2010;20:265-272 http://genome.cshlp.org/content/20/2/265/F2.expansion.html)

es tan importante por lo que se considera opcional en el algoritmo.
- Ensamblaje de *contigs*: El grafo de Bruijn se compone de nodos de 25-mers, se cortan los extremos que tienen una longitud menor de 50pb. De esta manera, para el genoma asiático se eliminaron 323 millones de nodos extremos. Finalmente en esta fase se generan *contigs* con longitud igual o superior a 100pb de tal manera que los tamaños de los *contigs* N50 y N90 son de 1050pb y 205pb, respectivamente.
- Construcción de super*contigs*: una vez obtenidos los *contigs*, mediante el grafo cada lectura corta está alineada con un *contig*. Se utiliza un mínimo de tres pares de lecturas como criterio para definir el orden y distancia entre dos *contigs*. El algoritmo comienza desde los insertos de tamaño más pequeño hasta los más grandes para ir construyendo los *supercontigs* usando los extremos emparejados para unir los *contigs*. En este paso se obtuvo que los *supercontigs* construidos tenían 17,3kb y el tamaño del N50 había mejorado a 103,5kb.
- Cierre de huecos: La mayoría de los huecos de los super*contigs* estaban compuestos de repeticiones que habían estado enmascaradas durante la construcción de los super*contigs*. Para desensamblar las copias repetidas y rellenar los huecos se usó la información de los extremos emparejados.

Como resultado práctico de la implantación del algoritmo se obtuvo el ensamblaje de los dos genomas comentados anteriormente en un sistema compuesto de 8 procesadores AMD *quad-core* a 2.3Ghz con 512Gb de memoria. El tiempo total del proceso fue de 48h para el asiático y 40h para el africano.

Comparándolo con un algoritmo similar como puede ser **ABYSS** [89] para un caso similar se tiene que este último consume más tiempo de CPU (87 horas) con un uso menor de memoria (<16Gb de **ABYSS** frente a 140Gb de **SOAP*denovo***).

El programa **SOAP*denovo*** está disponible gratuitamente pero sin el código fuente.

5.26. SSAKE

SSAKE [89] [111] (*Short Sequence Assembly by K-mer) and 3' read Extension*) es una aplicación para el ensamblaje de lecturas cortas de ADN. Actualmente está en la versión 3.7. El programa está escrito en Perl y el algoritmo se basa en búsquedas progresivas de *K-meros* en un árbol tipo *trie* (*prefix tree*) para encontrar el solapamiento idéntico mayor posible entre dos secuencias. El algoritmo ha sido utilizado en el proyecto metagenómico del mar de los Sargazos donde se usaron secuencias de lecturas de entre 25 y 46pb de genomas de virus, bacterias y hongos y creando un grafo simulado de 40 millones de 25-mers.

El funcionamiento del programa comienza con el almacenamiento en memoria de las secuencias de ADN (en formato FASTA), generando una tabla *hash* indexada mediante las lecturas de secuencia única y cuyo valor representa el número de ocurrencias de dicha secuencia en el conjunto de datos. Posteriormente se genera un árbol que se utiliza para organizar las secuencias y sus reversos complementarios por las primeras once bases de extremos 5'. Las lecturas se ordenan por orden decreciente del número de ocurrencias para mostrar el grado de cobertura y minimizar la extensión de las lecturas que contienen errores de secuenciación.

Cada lectura no ensamblada *u* se utiliza a su vez como núcleo de un ensamblaje. A partir de *u* se generan todos los posibles extremos 3' seguidos de un *K-mer* y se utilizan para entrar en un bucle de búsqueda hasta que la longitud de la palabra sea más pequeña que un mínimo *m* definido por el usuario o hasta que un *K-mer* tenga una coincidencia perfecta con el extremo *5'* de las bases de la lectura *l*. En este último caso, *u* se extiende con las bases de extremo 3' contenido en *l* y *l* se elimina tanto de la tabla *hash* como del árbol. Este proceso avanza cíclicamente con extremos 3' más *K-meros* más cortos que se repiten con cada ampliación de *u*. Debido a que sólo se permiten las búsquedas desde la izquierda en los árboles *trie*, cuando se han agotado todas la posibilidades para los extremos 3', se utiliza la cadena complementaria del *contig* generado para ampliarlo por el extremo 5'. El árbol tipo *trie* se utiliza para limitar el espacio búsqueda compartiendo las lecturas de secuencia de manera eficiente. Hay dos maneras de controlar la calidad de los resultados en **SSAKE**, la primera es parar la ampliación cuando un *K-mer* coincida con el extremo 5' de más de una lectura de secuencia, lo cual conduce a *contigs* más cortos pero minimiza el número de secuencias que se quedan sin ensamblar; y la segunda es parar la ampliación cuando un *K-mer* es más pequeño que una longitud de palabra *m* definida por el usuario.

SSAKE genera un fichero de *log* con la información de la ejecución con dos ficheros multi FASTA, uno que contiene la secuencia de todos los *contigs* generados y el otro que contiene las lecturas de secuencia no ensambladas.

Como resultados concretos se tienen que el programa tarda aproximadamente 25 horas en ensamblar las lecturas del genoma del mar de los Sargazos [77] secuenciado mediante Sanger con 40 millones de 25-mers teniendo un m de 16 en un procesador Opteron a 1,4Ghz con 32Gb de memoria

usando un máximo de 19Gb. En este caso el 11% de las lecturas de entrada se ensamblaron en *contigs* de tamaño mayor de 100pb, totalizando 12,8Mb. Por el contrario, no se ensamblaron el 32,5% de la secuencias de entrada. Las lecturas restantes terminaron en *contigs* cortos (26pb a 99pb).

5.27. **VCAKE**

VCAKE [143] (*Verified Consensus Assembly by K-mer Extension*) es un algoritmo orientado a la secuenciación de pequeños genomas *de novo* para lecturas cortas (~30pb).

VCAKE es una mejora del programa **SSAKE** [89] en el que se ha realizado una modificación de la extensión *K-mer* que disminuye el error al proporcionar una mayor profundidad de cobertura. **VCAKE** tiene en cuenta todas las lecturas que se solapan con una secuencia semilla que va ampliando base a base usando la base más representada de las lecturas coincidentes que superan ciertas condiciones. **VCAKE** busca todos los *K-mer* que coinciden exactamente con el extremo final 3' de la secuencia hasta un mínimo *n* definido por el usuario. Las primeras 11 bases del *K-mer* se utilizan para realizar una búsqueda en uno de los ficheros de entrada (denominado *bin*) y todas las claves devueltas se contrastan contra el resto del *K-mer*. Aquellas que coincidan perfectamente se introducen en una matriz un número de veces igual al número por el cual estaban indexados en el fichero *bin*. Tal como ocurre en **SSAKE**, el fichero *bin* es una tabla *hash* orientada a una búsqueda eficiente cuya clave consiste de las primeras 11 bases de cada lectura (o su complemento inverso) seguido por la secuencia con un valor que contiene el número de apariciones de esa lectura o su complemento inverso.

Los autores han conseguido ensamblar perfectamente el genoma original Tano **SSAKE** como **VCAKE** son capaces de ensamblar perfectamente el genoma original del SARS-TOR2 sin error con un 50x de cobertura. Para ensamblajes de lecturas con error **SSAKE** no mostró una cobertura perfecta por *contigs* que coincidieran con el genoma original. **VCAKE**, por el contrario, mostró el 100% de cobertura mediante *contigs* de los que el más largo tenía una longitud de 28.332pb.

Para una simulación de lecturas sin error **VCAKE** muestra una cobertura mayor pero no obtiene un *contig* de mayor longitud que **SSAKE**. Para el ensamblaje de lecturas con un ratio de error del 1% **VCAKE** ya consigue el mayor *contig* con una cobertura mayor.

5.28. **VELVET**

Velvet [29, 85] es un conjunto de algoritmos desarrollados por Daniel R. Zerbino y Ewan Birney para el tratamiento de grafos de de Bruijn para ensamblaje de secuencias genómicas. La aplicación **Velvet** puede producir *contigs* de cierta longitud (hasta un N50 de ~50kb para datos de origen procariotico y un N50 de ~3kb para BAC de mamíferos) con lecturas muy cortas y extremos emparejados. Cuando se utiliza con conjuntos de datos reales tipo Solexa sin lecturas emparejadas, **Velvet** genera *contigs* de ~8kb en procrariotas y de 2kb en una BAC de mamíferos.

Velvet se ha utilizado tanto para simulaciones como para datos reales. Cuando se usan lecturas cortas con extremos emparejados **Velvet** consigue ensamblar genomas bacterianos completos con un *contig* N50 de longitud superior a 50kb.

Velvet hace un uso bastante extenso de herramientas de simplificación de grafos para reducir caminos no intersectantes con nodos simples. Esta simplificación comprime el grafo sin pérdida de información. El programa ejecuta la fase de simplificación durante la construcción del grafo y, de nuevo, en varias ocasiones durante el proceso de ensamblaje. Esta técnica, introducida como «eliminación de únicos» para grafos de *K-mer* [86] es similar a la formación de *unitigs* en los grafos de solapamiento [85] y en los ensambladores de Solapamiento-Diseño-Consenso [85].

Velvet poda el grafo de *K-meros* eliminando ramales cortos de manera iterativa y su algoritmo de eliminación de puntas (*tips*) es similar al procedimiento de erosión de **EULER**. Esta eliminación de ramales cortos reduce de manera drástica el tamaño del grafo con datos reales [85], posiblemente debido a que su flujo de trabajo elimina primero errores de secuencias. **Velvet**, por el contrario, no implementa el filtrado de alineamiento espectral de **EULER** (y las publicaciones de **Velvet** no parecen alentar su uso) sino que utiliza un parámetro para el mínimo número de ocurrencias de lecturas para que un *K-mer* sea calificado como un nodo del grafo.

Velvet reduce la complejidad del grafo mediante una búsqueda acotada de las «burbujas» con un algoritmo de búsqueda en anchura que comienza primero con los nodos que tienen múltiples enlaces salientes ya que, en condiciones reales, una búsqueda exhaustiva de burbujas (que pueden tener a su vez burbujas dentro) es impracticable. Esta búsqueda, por lo tanto, está acotada para hacerla tratable y los caminos candidatos se recorren en fases, moviéndose nodo a nodo desde el comienzo y descubriendo todos los caminos alternativos iterativamente, hasta que la longitud del camino excede el umbral definido. **Velvet**

restringe las burbujas candidatas a aquellas que tengan secuencias similares en los caminos alternativos y una vez que encuentra una burbuja elimina el camino que contenga menos lecturas y realinea las lecturas del camino eliminado al camino que queda. Debido a que una alta multiplicidad de lecturas determina el camino objetivo, la realineación genera secuencias consenso mediante un algoritmo de puntuación. Este algoritmo es similar al que tiene **EULER** para eliminar protuberancias [88] y análogo a la detección y «suavizado» de burbujas en los algoritmos de Solapamiento-Diseño-Consenso [85].

Velvet reduce la complejidad adicionalmente mediante el seguimiento de lecturas eliminando caminos que contienen menos lecturas que un umbral determinado. Esta operación tiene el riesgo de eliminar secuencias con baja cobertura pero está pensado para eliminar principalmente conexiones terminales inducidas por errores de secuenciación convergentes. **Velvet** explota las lecturas largas mediante un algoritmo denominado «Rock Band» que forma nodos exteriores a caminos y que se conforman por dos o más lecturas largas que se escogen cuando ningunas otras dos lecturas largas proporcionan una contradicción consistente.

La reducción final del grafo de **Velvet** involucra extremos apareados aunque las versiones tempranas utilizaban un algoritmo denominado «migaja de pan» [85] que es similar al de extremos apareados utilizado en los algoritmos de de Bruijn [88] y en el relleno de huecos de los algoritmos de Solapamiento-Diseño-Consenso [85]. El algoritmo opera en parejas de *contigs* (caminos simples) conectados por finales emparejados y fijando los *contigs* largos intenta rellenar el hueco entre ellos mediante *contigs* cortos. El algoritmo reúne los *contigs* cortos vinculados a los *contigs* largos y aplica una búsqueda primero en amplitud a través del grafo y por un camino simple se definen los *contigs* largos y mediante transversabilidad los *contigs* cortos. Las últimas versiones de **Velvet** utilizan un algoritmo denominado «*pebble*» [85]. En este algoritmo los *contigs* únicos y repetidos son sustituidos por «migaja de pan» por *contigs* largos y cortos, respectivamente. Por otro lado, **Velvet** utiliza un test similar al A-stat [85] del **CELERA ASSEMBLER** y una expectativa dada de cobertura y utilizando la distribución de longitudes construye un diseño de *contigs*. Por otro lado busca en el grafo el camino que sea consistente con el diseño.

Velvet se puede ejecutar varias veces por cada conjunto de datos para optimizar la selección de tres parámetros críticos. La longitud de los *K-meros* se restringe para que sea impar para excluir nodos que representen repeticiones palindrómicas. La frecuencia mínima esperada de *K-meros* en las lecturas determina que *K-meros* deben ser podados *a priori* y la longitud esperada de cobertura controla las conexiones terminales rotas.

En resumen, **Velvet** ofrece una implementación completa de ensamblaje basado en grafos de de Bruijn sin utilizar un preprocesamiento para la corrección de errores aunque sí dispone de un filtro para evitar errores en lecturas. Aplica una serie de heurísticas que reducen la complejidad del grafo y que explotan topologías locales, cobertura de lecturas, identidad de secuencias y restricciones de extremos apareados. El software está orientado a ensamblaje *de novo* a partir de lecturas cortas con extremos apareados de la plataforma Solexa. Existe una extensión que permite ensamblar conjuntos de datos compuestos exclusivamente de lecturas SOLiD aunque los requisitos de memoria excluyen a **Velvet** para secuenciar grandes genomas.

5.29. **YAGA**

YAGA [15] es el acrónimo de *Yet Another Genome Assembler*. Está basado en un desarrollo en C++ (de más de 12.000 líneas de código) para ejecutar en paralelo el ensamblaje de transcriptomas con grandes cantidades de conjuntos de datos basados en lecturas cortas. La solución utiliza un grafo bidirigido (basado en un grafo de Bruijn) distribuido que genera la información de los solapamientos de las lecturas. Posteriormente genera los *contigs* mediante una compactación del grafo anterior utilizando distintas ponderaciones. Finalmente, mediante el grafo anterior resuelve los huecos teniendo en cuenta la cobertura de las lecturas.

El programa se ha probado en regiones codificantes de *Zea mays* ensamblando 925 millones de secuencias compuestas por 40 mil millones de nucleótidos en varios minutos utilizando 1024 procesadores. Los autores destacan la escalabilidad de la solución y su arquitectura distribuida.

5.30. **ZOOM**

YAGA [89] (Zillions of Oligos Mapped) es un programa diseñado para asignar millones de lecturas cortas contra genomas de referencia y orientado a la tecnología NGS. Según los autores el sistema es capaz de asignar lecturas con un 15x de cobertura de Illumina/Solexa de un genoma humano al genoma de referencia en un día de CPU.

El programa está comercializado por la empresa Bioinformatics Solutions Inc. Pero existe una versión descargable de manera gratuita en la dirección http://www.bioinfor.com/**ZOOM**.

6. EVALUACIÓN DE PROGRAMAS

En este punto vamos a realizar una evaluación de la instalación y pruebas funcionales de programas de ensamblaje para lecturas cortas que hacen uso de grafos de Bruijn, entre los que analizaremos **ABYSS**, **ALLPATHS**, **EULER-SR**, SOPA*denovo* y **Velvet**.

6.1. EVALUACIÓN DE **ABYSS**

La versión de **ABYSS** actual es la 1.2.6 que se descarga de: http://www.bcgsc.ca/platform/bioinfo/software/**ABYSS**

Una vez descargado se descomprime el paquete en el directorio deseado y la manera de compilarlo por defecto es la siguiente:

```
$./configure && make
```

Si se desea compilar en un directorio específico:

```
$ ./configure –prefix=/opt/ABYSS && make && sudo make install
```

Si se desea compilar para procesamiento paralelo con soporte para MPI, se hace de la siguiente manera:

```
$ ./configure –with-mpi=/usr/lib/openmpi && make
```

Las librerías MPI deben estar en `/usr/include` y `/usr/lib` o en su localización específica para configurar.

Si instalamos **Google sparse*hash*** tal como recomienda **ABYSS** para reducir el uso de memoria entonces:

```
$./configure CPPFLAGS=-I/usr/local/include
```

El tamaño máximo de los *K-mer* es de 64 pero se puede ajustar mediante:

```
$./configure –enable-maxk=32 && make
```

Para ejecutar **ABYSS**, los binarios se tienen que encontrar en el `PATH`.

Como resumen, en nuestro caso, hemos compilado **ABYSS** de la siguiente manera (instalando previamente Google sparsh*ash*):

```
$./configure CPPFLAGS=-I/usr/include/google --enable-maxk=32
```

La siguiente opción nos permite ver las opciones del programa:

```
mmhr@ubuntu:~/Ensambladores/ABYSS-1.2.6/ABYSS$ ./ABYSS --help

Usage: ABYSS [OPTION]... FILE...

Assemble the input files, FILE, which may be in FASTA, FASTQ,
qseq, export, SAM or BAM format and compressed with gz, bz2 or xz.


      --chastity                     discard unchaste reads [default]
                                     for qseq, export and SAM files only
      --no-chastity                  do not discard unchaste reads
      --trim-masked                  trim masked bases from the ends of reads
                                     [default]
      --no-trim-masked               do not trim masked bases from the ends of reads
  -q, --trim-quality=THRESHOLD       trim bases from the ends of reads whose quality
                                     is less than the threshold
      --standard-quality             zero quality is `!' (33)
                                     default for FASTQ and SAM files
      --illumina-quality             zero quality is `@' (64)
                                     default for qseq and export files
  -o, --out=FILE                     write the contigs to FILE
  -k, --kmer=KMER_SIZE               K-mer size
  -t, --trim-length=TRIM_LENGTH      maximum length of dangling edges to trim
  -c, --coverage=COVERAGE            remove contigs with mean K-mer coverage
                                     less than this threshold
  -b, --bubbles=N                    pop bubbles shorter than N bp [3*k]
  -b0, --no-bubbles                  do not pop bubbles
  -e, --erode=COVERAGE               erode bases at the ends of blunt contigs with
                                     coverage less than this threshold
  -E, --erode-strand=COVERAGE        erode bases at the ends of blunt contigs with
                                     coverage less than this threshold on either
```

```
                                    strand
      --coverage-hist=FILE          record the
K-mer coverage histogram in FILE
  -g, --graph=FILE                  generate a
graph in dot format
  -s, --snp=FILE                      record
popped bubbles in FILE
  -v, --verbose                     display
verbose output
      --help     display this help and exit
      --version  output version information
and exit

Report bugs to <ABYSS-users@bcgsc.ca>.
```

Figura 27: Descarga de Google-Sparsehash para reducir el uso de memoria en **ABYSS**

Para ensamblar lecturas cortas de un fichero llamado `reads.fa` en *contigs* y generar un fichero llamado *contigs.fa* ejecutamos el siguiente comando:

```
$ABYSS -k25 reads.fa -o contigs.fa
```

donde *k* es la longitud de los *K-mer*.

La única manera de encontrar el valor óptimo de *k* es ejecutar varias veces el programa y observar los resultados. El siguiente *script* ensambla las lecturas para valores de *k* entre 20 y 40.

```
$for k in {20..40}; do
   ABYSS -k$k reads.fa -o contigs-k$k.fa
done
```

El valor máximo de *k* es 64 aunque se puede modificar en tiempo de compilación utilizando `-enable-maxk`. También se puede fijar en un número inferior para disminuir el uso de memoria lo cual es útil en procesos largos paralelizados.

Para ensamblar lecturas cortas emparejadas en dos ficheros llamados *reads1.fa y reads2.fa* en *contigs* en un fichero llamado *ecoli-contigs.fa*, ejecutamos:

```
$ABYSS-pe k=25 n=10 in='reads1.fa reads2.fa' name=ecoli
```

Donde *k* es la longitud de los *K-mer, n* es el número mínimo de pares que se necesitan para considerar que se puede generar un *contig*. El valor óptimo para *n* se tiene que buscar mediante diferentes pruebas.

ABYSS`-pe` es un *script* que se ejecuta como un ensamblaje único pero que llama a varios programas que deben estar en el `PATH`:

- **ABYSS**: el ensamblador
- **AdjList**: busca solapamientos de longitud *k-1* entre *contigs*
- **PopBubbles**: finaliza variaciones en el grafo
- **ParseAligns**: busca pares de lecturas en alineamientos
- **DistanceEst**: estima distancias entre *contigs*
- **Overlap**: busca solapamientos entre *contigs*
- **SimpleGraph**: busca caminos entre pares de *contigs*
- **MergePaths**: une caminos consistentes
- **Consensus**: convierte *contigs* de espacio de color a *contigs* de nucleótidos

La opción *np* de **ABYSS**`-pe` específica el número de procesos que lanzará el trabajo paralelo **ABYSS**`-P`. Si no se configura MPI, permite hacer uso de múltiples *cores* en un sistema de un solo procesador. Para utilizar varios sistemas para ensamblar se tiene que crear un fichero por `mpirun`.

La fase de ensamblaje de lecturas emparejadas es *multithreaded* pero se ejecuta en un sólo servidor. El número de *threads* que se pueden utilizar deben especificarse con el parámetro `j` que por defecto es el mismo que para np. **ABYSS** es *multithreaded* mediante el uso de `pthread` y `OpenNP` pero requiere compilarse con una versión de GCC igual o mayor que la 4.3.

`OpenMPI` se integra correctamente con SGE (*Sun Grid Engine*) por lo que para enviar una lista de trabajos para ensamblar cada valor impar de *k* entre 51

y 63 utilizando 65 procesos por cada trabajo, podemos hacer lo siguiente:

```
$qsub -pe openmpi 64 -t 51-63:2 -N testing ABYSS-pe in=reads.fa n=10
```

6.1.1. PRUEBA DE EJECUCIÓN DE **ABYSS**

Realizamos prueba básica de **Velvet** con 1 fichero de 142.858 lecturas cortas de 35pb.

Incluimos en el PATH tres directorios necesarios para ejecutar los distintos programas de **ABYSS** que nos hacen falta:

- Directorio base de instalación de **ABYSS**: en nuestro caso ~/Ensambladores/**ABYSS**-1.2.6. En este directorio están los siguientes programas que hacen falta cuando se ejecuta el script **ABYSS**-pe:
- AdjList: Encuentra solapamientos de longitud k-1 entre *contigs*
- PopBubbles: Finaliza variaciones
- KAligner: alinea lecturas a *contigs*
- ParseAligns: encuentra parejas de lecturas en alineamientos
- DistanceEst: estima distancias entre *contigs*
- Overlap: busca solapamientos entre *contigs* complejos
- SimpleGraph: busca caminos entre parejas de *contigs*
- MergePaths: une caminos consistentes
- Consensus: convierte espacio *contigs* con espacio de color a *contigs* de nucleótidos.

— Directorio del ejecutable **ABYSS**: en nuestro caso ~/Ensambladores/**ABYSS**-1.2.6/**ABYSS**

— Directorio de los binarios: en nuestro caso: ~/Ensambladores/**ABYSS**-1.2.6/bin

Realizamos la prueba:

```
mmhr@ubuntu:~/Ensambladores/ABYSS-1.2.6$ ~/Ensambladores/ABYSS-1.2.6/ABYSS/ABYSS -k21 ~/Datos/Datasets/EULER-SR/reads.fasta -o ~/Resultados/ABYSS/contigs-datosEULER-prueba.fa

ABYSS 1.2.6

/home/mmhr/Ensambladores/ABYSS-1.2.6/ABYSS/ABYSS -k21 /home/mmhr/Datos/Datasets/EULER-SR/reads.fasta -o /home/mmhr/Resultados/ABYSS/contigs-datosEULER-prueba.fa

Reading `/home/mmhr/Datos/Datasets/EULER-SR/reads.fasta'

Loaded 803044 k-mer

Minimum k-mer coverage is 4

Using a coverage threshold of 4...

The median k-mer coverage is 14

The reconstruction is 98571

The k-mer coverage threshold is 3.74166

Setting parameter e (erode) to 4

Setting parameter E (erodeStrand) to 1

Setting parameter c (coverage) to 3.74166

Generating adjacency

Generated 1590460 edges

Eroding tips

Eroded 693155 tips

Eroded 0 tips

Trimming short branches: 1

Trimming short branches: 2

Trimmed 2 k-mer in 1 branches

Trimming short branches: 4

Trimmed 17 k-mer in 5 branches

Trimming short branches: 8

Trimming short branches: 16

Trimmed 19 k-mer in 2 branches

Trimming short branches: 21

Trimmed 8 branches in 5 rounds

Marked 2545 edges of 1266 ambiguous vertices.

Removing low-coverage contigs (mean k-mer coverage < 3.74166)

Found 109851 k-mer in 1825 contigs before removing low-coverage contigs

Removed 11724 k-mer in 580 low-coverage contigs

Split 1160 ambiguous branches

Eroding tips

Eroded 0 tips

Eroded 0 tips

Trimming short branches: 1

Trimming short branches: 2

Trimming short branches: 4

Trimming short branches: 8

Trimming short branches: 16

Trimming short branches: 21
```

```
Trimmed 0 branches in 5 rounds
Popping bubbles
Removed 8 bubbles
Removed 8 bubbles
Marked 178 edges of 88 ambiguous vertices.
Assembled 97942 k-mer in 114 contigs
Removed 704917 k-mer.
The signal-to-noise ratio (SNR) is -8.56349 dB.
```

El programa genera un fichero (*contigs*-datos**EULER**-prueba.fa) de *contigs*:

```
>0 2478 35896
TCAAAACAGTGTTTTGAGCAACCTGTGACTAGCTTTCTAATCG
ATGCCTTGGTTTTCATTGCCTATAATCAAAAAGAGAAATTTTCTCCT
GAAAAGCATATAGAGTAGCTGGCGTTAAAAGCTCCTGTCTTGCTTTT
TTGACCTATAGTCATATCTATCAAGTATTGTTCTTGCCTAAGCTATC
AATAAAAAGGTGGCATTTTTTAGGCTTGGTGTTAGTGAATTTTGCCT
TATCCTATCTAAGTCATTTCGAGCTTTTTATGGTACAATGGAAACAT
GTTATTCAAATTATCTAAGGAAAAAATAGAGCTAGGCTTATCTCGTT
TATCGCCAGCCCGTCGTATTTTTTTGAGTTTTGCCTTGGTCATTTTA
CTAGGCTCTCTTCTTTTGAGCTTGCCCTTTGTCCAAGTTGAAAGCTC
ACGAGCGACTTATTTTGATCATCTTTTCACTGCTGTCTCTGCAGTCT
GTGTGACAGGTCTCTCAACCCTTCCAGTAGCTCACACCTATAATATC
TGGGGCCAAATAATCTGTTTGCTCTTGATTCAGATCGGTGGTCTAGG
GCTCATGACCTTTATTGGGGTTTTCTATATCCAGAGCAAGCAAAAGC
TTAGTCTTCGTAGCCGTGCAACTATTCAGGATAGTTTTAGTTATGGA
GAAACTCGATCTTTGAGAAAGTTTGTCTATTCTATTTTTCTCACGAC
CTTTTTGGTTGAGAGCTTGGGAGCTATTTTGCTTAGTTTTCGCCTTA
TTCCTCAACTTGGCTGGGGACGTGGTCTTTTTAGTTCCATTTTTCTA
GCGATCTCAGCCTTCTGTAATGCCGGTTTTGATAATTTAGGGAGCAC
CAGTTTATTTGCTTTTCAGACCGATTTACTGGTCAATCTGGTGATTG
CAGGCTTGATTATTACAGGCGGCCTTGGTTTTATGGTCTGGTTTGAT
TTGGCTGGTCATGTAGGAAGAAAGAAAAAAGGACGTCTGCACTTTCA
TACGAAGCTTGTATTATTATTGACTATAGGTTTGTTGTTATTTGGAA
CGGCAACTACTCTCTTTCTTGAGTGGAACAATGCTGGAACGATTGGC
AATCTCCCTGTTGCCGATAAGGTTTTAGTTAGCTTTTTTCAAACAGT
GACGATGCGAACAGCTGGCTTTTCTACGATAGATTATACTCAGGCTC
ATCCTGTGACTCTTTTGATTTATATCTTACAGATGTTTCTAGGTGGG
GCACCTGGAGGAACAGCTGGGGGACTCAAGATTACGACATTTTTTGT
CCTCTTGGTCTTTGCACGAAGTGAGCTTCTAGGCTTGCCTCATGCCA
ATGTTGCGAGACGAACGATCGCGCCGCGAACGGTTCAAAAATCCTTT
AGTGTCTTTATTATCTTTTTGATGAGCTTCTTGATAGGATTGATTCT
GCTAGGGATAACAGCCAAAGGCAATCCTCCCTTTATCCACCTCATAT
TTGAAACCATTTCAGCTCTTAGTACAGTTGGTGTAACGGCAAATCTG
ACTCCTGACCTTGGGAAATTGGCTCTCAGTGTTATCATGCCACTTAT
GTTTATGGGACGAATTGGTCCCTTGACCTTGTTTGTTAGCTTGGCAG
ATTACCATCCAGAAAAGAAAGATATGATTCACTATATGAAAGCAGAT
ATTAGTATTGGTTAAGAAAGGAAAGAGCATGTCAGATCGTACGATTG
GAATTTTGGGCTTGGGAATTTTTGGGAGCAGTGTCCTAGCTGCCCTA
GCCAAGCAGGATATGAATATTATCGCTATTGATGACCACGCAGAGCG
CATCAATCAGTTTGAGCCAGTTTTGGCGCGTGGAGTGATTGGTGACA
TCACAGATGAAGAATTATTGAGATCAGCAGGGATTGATACCTGCGAT
ACCGTTGTAGTCGCGACAGGTGAAAATCTGGAGTCGAGTGTGCTTGC
GGTTATGCACTGTAAGAGTTTGGGGGTACCGACTGTTATTGCTAAGG
TCAAAAGTCAGACCGCTAAGAAAGTGCTAGAAAAGATTGGAGCTGAC
TCGGTTATCTCGCCAGAGTATGAAATGGGGCAGTCTCTAGCACAGAC
CATTCTTTTCCATAATAGTGTTGATGTCTTTCAGTTGGATAAAAATG
TGTCTATCGTGGAGATGAAAATTCCTCAGTCTTGGGCAGGTCAAAGT
CTGAGTAAATTAGACCTCCGTGGCAAATACAATCTGAATATTTTGGG
TTTCCGAGAGCAGGAAAATTCCCCATTGGATGTTGAATTTGGACCAG
ATGACCTCTTGAAAGCAGATACCTATATTTTGGCAGTCATCAACAAC
CAGTATTTGGATACCCTAGTAGCATTGAATTCGTAAAGAGGGATGAC
CCCTCTTTTTTGATGCTTAAGATGGCAAATAGAGACAGAAGCCCCTT
GTCTTCTAGTAAAAGTTCTTCAAAGGCTGGACTTTATGGTAAAATAG
AAGGAAGTGACAAGAGAGAGTAATACTCAATGAAAATC

>1 759 11477
GAGAGCTAAACCACCAAGTAAAAGGAACAAAACCAAGAAAGGT
TGATAGAGATAGACTTAAAAACTATCTTACTGACAATCCAGACGCTT
ATTTGACTGAAATAGCTTCTGAATTTGCCTGTCATCCAACTACCATC
CACTATGCGCTCAAAGCTATGGGCTACACTCGAAAAAAAGAACCACA
CCTACTATGAACAAGACCCAGAAAAAGTAGCCTTATTTCTTAAAAAT
TTTAATAGTTTAAAGCACCTAGCACCTGTTTAGATTGATGAAACAGG
ATTCGATACTTATTTTTATCGAGAATATGGTCGCTCATTAAAAGGTC
AGTTAATAAGAGGTAAAGTATCTGGAAGAAGATATCAGAGGATTTCT
TTGGTTGCAGGTCTAACAAATGGTGAGTTAATCGCTCCAATGACTTA
CGAAGAGATGGTGACGAGCGACTTTTTTGAAGCTTGGTTTCAGAAGT
TTCTCTTACCAACATTAACCACACCATCGGTTATTATTATGGATAAT
GCAAGATTCCATAGAATGGGTAAGTTAGAACTTTTATGCGAGGAGTT
TGGGCATAAACTTTTACCTCTTCCTCCCTACTCACCTGAGTACAATC
CTATTGAGAAAACATGGGCTTATATCAAAAAGAACCTCAAAAAGGTA
TTACCAAGTTGCAATACCTTTTACGAGGCTCTTTTCTCTTGTTCTTG
TTTCAATTGACTATATTAGAGGCGAGACATTTTTCGGTTCTTTGTCA
ACTGTAGTGGG

>2 66 340
ATAAGACTGACGAAGTCAGTTACATATATCTACGGCAAGGCGA
AGCTGACGCGGTTTGAAGAGATT

>3 38 330
CAGTGTTTTGAGCAGCCCGCGGCTAGTTTCCTAGTTTG ….
```

6.2. EVALUACIÓN DE **ALLPATHS**

6.2.1. INTRODUCCIÓN

El programa **ALLPATHS** se puede descargar de ftp://ftp.broadinstitute.org/pub/crd/**ALLPATHS**/Release-LG/ . La versión actual se generó el 24 de Marzo de 2011. **ALLPATHS** es un ensamblador para genomas completos que utiliza lecturas cortas (aprox. 100pb) utilizando un grafo que representa ambigüedades como polimorfismos, errores de lectura no corregidos, repeticiones no resueltas, etc. Por lo que proporciona información útil que se encuentra ausente en ensambladores de una generación previa.

ALLPATHS no está diseñado para ensamblar lecturas Sanger o 454 FLX, ni ninguna mezcla de ambas. Igualmente, requiere una lata cobertura del genoma para compensar la pequeña longitud de las lecturas. Esta cobertura específica depende de la

longitud y la calidad de las lecturas emparejadas pero, normalmente, se sitúa en 100x o superior.

ALLPATHS requiere un mínimo de dos bibliotecas con extremos emparejados (uno corto y uno largo). El tamaño de separación medio de la biblioteca corta debe ser un poco menor que dos veces el tamaño de la lectura, de manera que las lecturas de un par se solapen ligeramente (por ejemplo, para lecturas de 100 bases, el tamaño del inserto debería ser de 180 bases). La distribución de tamaños debe ser el mínimo posible, con una desviación estándar menor del 20%. El tamaño del inserto de la blblioteca larga debe tener aproximadamente 3000 bases de longitud y puede tener una distribución mayor. Se pueden utilizar bibliotecas de insertos mayores para eliminar ambigüedades en grandes estructuras repetidas y necesitar una menor cobertura.

Las bibliotecas deben ser puras, es decir, deben consistir en lecturas que no contengan ninguna porción no genómica de *stuffers* o construcciones similares. Las lecturas que unen bibliotecas pueden ser quiméricas, es decir, pueden cruzar el punto de unión entre los dos extremos del inserto que se produce en bibliotecas que usan el protocolo (*sheared*) de Illumina.

6.2.2. REQUISITOS

Para compilar y ejecutar **ALLPATHS**-LG se necesita un sistema Linux/UNIX con al menos 16GB de RAM aunque se recomienda un mínimo de 32GB para pequeños genomas y 512GB para genomas de tamaño humano. Adicionalmente se necesita el siguiente software:

- El compilador g++, versión 4.3.3 o superior (http://gcc.gnu.org/)
- La librería C++ Boost, versión 1.38 (http://www.boost.org/)
- El comando para grafos `dot` del paquete `graphviz`. Se usa la versión 2.16.1 (http://www.graphviz.org)
- La utilidad `addr2line` del paquete `binutils` (http://www.gnu.org)
- Las utilidades Java en línea de comandos Picard para manipulación de ficheros SAM disponibles en http://picard.sourceforge.net/

6.2.3. INSTALACIÓN

Después de descargar el fichero, se desempaqueta con el comando `tar` y se compila el código fuente con `configure` y `make`. El código fuente se sitúa en el directorio denominado **`ALLPATHS`**-`<revisión>`, con lo que los pasos a seguir son los siguientes:

$ tar xzvf **ALLPATHS**-<revisión>.tar

$ cd **ALLPATHS**lg*

$ ./configure –prefix=/directorio/de/instalación

$ make

En pasos anteriores puede dar problemas la ejecución del comando `configure` y es posible que sea necesario modificar el `PATH` o `LD_LIBRARY_PATH` y también ejecutar dicho comando con opciones como puede ser: `configure –with-boost=/path/a/boost/`. Para un listado de todas las opciones posibles se puede ejecutar lo siguiente:

```
$ configure –help
```

Después de la compilación, los ficheros ejecutables se encontrarán en el subdirectorio bin por lo que será necesario añadirlo a la variable `PATH` al igual que el directorio `addr2line`. Puede ser necesario también modificar la variable `LD_LIBRARY_PATH`.

6.2.4. FLUJO DE TRABAJO

ALLPATHS consiste en una serie de módulos cada uno de los cuales ejecuta una fase en el proceso de ensamblaje. Estos diferentes módulos se pueden ejecutar en un orden variable dependiendo de los parámetros de ensamblaje y existe un único módulo denominado `Runallpathslg` que controla el flujo de trabajo global, decidiendo en cada momento que módulos deben ejecutarse y como. Aunque es posible ejecutar los módulos de manera individual, se recomienda hacerlo a través de `Runallpathslg`.

6.2.4.1. El Módulo `Runallpathslg`

`Runallpathslg` utiliza la utilidad `make` para controlar el flujo de trabajo del ensamblaje de manera que no llama a cada módulo de manera independiente sino que crea un `makefile` especial que se encarga de ello. Dentro de `Runallpathslg` cada modulo se define en términos de sus ficheros origen y destino así como la línea de comandos utilizada para llamarlo. Un módulo se ejecuta únicamente cuando sus ficheros destino no existen, han caducado con respecto a sus ficheros de origen o si el comando utilizado para llamar a dicho módulo se ha modificado. De esta manera `Runallpathslg` se puede ejecutar tantas veces como sea necesario con diferentes parámetros y solamente se llaman aquellos módulos que se necesitan. Esto es una manera eficiente y segura de controlar que todos los ficheros intermedios son siempre correctos independientemente tanto de las veces que se haya ejecutado `Runallpathslg` con un conjunto de datos concreto como de las veces que un módulo ha fallado o ha abortado su ejecución.

6.2.4.2. Estructura de Directorios

El flujo de trabajo de trabajo del ensamblaje utiliza la estructura de directorios (creándolos si no existen)

que se muestra a continuación para almacenar los datos de entrada, intermedios y de salida. Los nombres que se utilizan se pueden cambiar mediante línea de comandos (mediante el argumento PRE de Runallpathslg) pero estos son los utilizados habitualmente:

REFERENCE / DATA / RUN / ASSEMBLIES / SUBDIR

En el directorio especificado por PRE existirán tantos directorios REFERENCE como organismos se estén ensamblando por **ALLPATHS**.

6.2.4.2.1. Directorio REFERENCE (Organismo)

El directorio REFERENCE se utiliza para separar por organismos los proyectos de ensamblaje. Se denomina así porque debe existir un por cada genoma de referencia que se utilice. Igualmente, todos los ficheros intermedios generados que son independientes de un ensamblaje concreto se almacenarán en este directorio.

No es necesario suministrar un genoma de referencia (**ALLPATHS** es un ensamblador para *de novo*), pero incluso en los ensamblajes *de novo*, el flujo de trabajo puede ejecutar diversas evaluaciones en diferentes etapas del ensamblaje por lo que puede ser interesante proporcionar un genoma de referencia. Si no existe tal genoma de referencia, simplemente se crea un único directorio REFERENCE para el organismo que se desee ensamblar.

El directorio REFERENCE puede contener varios directorios DATA, cada uno representa un conjunto particular de datos (lecturas) para ensamblar.

La forma de crear el directorio es la siguiente:

```
$ Runallpathslg argument: REFERENCE_NAME
```

6.2.4.2.2. Directorio DATA (Proyecto)

El directorio DATA contiene los datos (lecturas) originales utilizados en un ensamblaje concreto. Los datos se almacenan internamente en formatos **ALLPATHS**: *fastb, qualb* y *pairs*. También contiene otros ficheros intermedios que se generan en el proceso.

Cada directorio DATA puede contener a su vez varios directorios RUN que representan un intento concreto de ensamblar los datos originales utilizando diferentes parámetros.

La creación de este directorio se realiza de la siguiente manera:

```
$ Runallpathslg argument: DATA_SUBDIR
```

6.2.4.2.3. Directorio RUN (Preprocesamiento del ensamblaje)

El directorio RUN contiene todos los ficheros de ensamblaje no locales, es decir, aquellos ficheros intermedios generados a partir de los datos originales para preparar la etapa de ensamblaje final. Puede contener también ficheros intermedios utilizados en la evaluación y que son dependientes de los parámetros de ensamblaje seleccionados.

La creación de este directorio se realiza de la siguiente manera:

```
$ Runallpathslg argument: RUN
```

6.2.4.2.4. Directorio ASSEMBLIES

El directorio ASSEMBLIES contiene el ensamblaje (o ensamblajes) actual. No tiene argumentos.

6.2.4.2.5. Directorio SUBDIR (ensamblaje)

El directorio SUBDIR es donde se genera el ensamblaje final junto con ficheros intermedios y de evaluación.

Se crea de la siguiente manera:

```
$ Runallpathslg argument: SUBDIR
```

6.2.4.3. Preparación de los Datos

Los argumentos siguientes son obligatorios:

PRE – El directorio raíz en el que se creará el flujo de trabajo de **ALLPATHS** .

REFERENCE_NAME – El nombre del directorio REFERENCE (organismo)

DATA_SUBDIR – El nombre del directorio del proyecto

RUN – El nombre del directorio RUN para el procesamiento previo del ensamblaje.

SUBDIR – El nombre del directorio SUBDIR (ensamblaje).

K – El tamaño utilizado de *K-mer*.

6.2.4.4. Preparación de los Datos

Antes de ejecutar **ALLPATHS** se deben prepara los datos para importarlos en el flujo de trabajo. Esta tarea requiere adaptar los datos a los formatos apropiados y añadir metadatos para describirlos.

6.2.4.4.1. Construcción de las Bibliotecas Soportadas

Cualquier conjunto de datos de entrada debe incluir al menos una biblioteca de fragmentos y otra de saltos. Una biblioteca de fragmentos es una biblioteca con una

corta separación entre insertos, menos que dos veces la longitud de lectura de manera que las lecturas están solapadas (por ejemplo, 100pb de Illumina se toman a partir de 180pb de insertos). Una biblioteca de saltos (*jumping library*) tiene una separación mayor, normalente entre 3kpb y 10kpb y puede incluir bibliotecas recortadas, EcoP15I o cualquier otra biblioteca para construcción. **ALLPATHS** puede manipular lecturas quiméricas en bibliotecas de saltos aunque cada lectura debe ser lo suficientemente larga para asegurar solapamientos.

De manera adicional, **ALLPATHS** también soporta bibliotecas de saltos largos donde una biblioteca de saltos se considera larga si el tamaño del inserto es mayor que 20kpb. Estas bibliotecas son opcionales y se utilizan sólo para mejorar los *supercontigs* en genomas de tamaño similar al humano. Normalmente la cobertura de saltos largos de menos de 1x es suficiente para mejorar los *supercontigs* de manera significativa.

ALLPATHS no soporta actualmente datos de otros métodos de construcción de bibliotecas, incluyendo las lecturas no emparejadas.

6.2.4.4.2. Orientación de las Lecturas

ALLPATHS espera que las lecturas de la biblioteca de fragmentos (y en las de saltos largos) estén orientadas de manera enfrentada (inward):

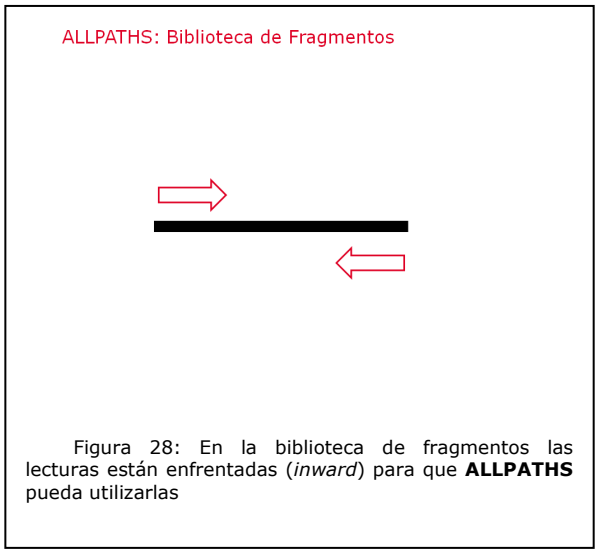


Figura 28: En la biblioteca de fragmentos las lecturas están enfrentadas (*inward*) para que **ALLPATHS** pueda utilizarlas

Sin embargo para las librerías de saltos **ALLPATHS** utiliza las lecturas que están en dirección contraria una de la otra.

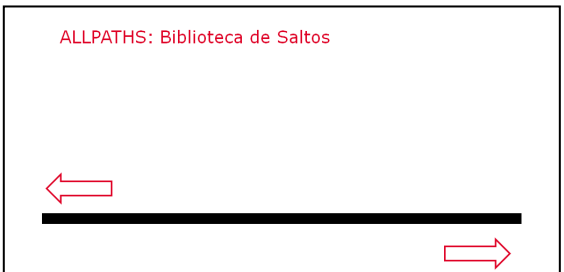


Figura 29: En la biblioteca de saltos las lecturas están en dirección contraria (*outward*) una de la otra debido a la construcción de la propia biblioteca.

6.2.4.5. Ficheros de Entrada

El directorio DATA contiene inicialmente los ficheros de lecturas secuenciadas, su puntuación de calidad e información referente a su emparejamiento. También debe existir un fichero `ploidy`. Todos estos ficheros pueden existir previamente si se trata de la continuación de un ensamblaje o bien pueden ensamblarse conjuntamente utilizando diversas herramientas proporcionadas en el paquete de **ALLPATHS**.

6.2.4.5.1. Ficheros de Bases, Puntuación de Calidad e Información de Emparejamientos

Las bibliotecas de lecturas mencionadas anteriormente son instanciadas para **ALLPATHS** en ficheros que contienen las bases, un fichero que mantiene los indicadores de calidad y un fichero con la información de emparejamiento. Los nombres específicos de los ficheros son los siguientes:

```
<REF>/<DATA>/frag_reads_orig.fastb
<REF>/<DATA>/frag_reads_orig.qualb
<REF>/<DATA>/frag_reads_orig.pairs

<REF>/<DATA>/jump_reads_orig.fastb
<REF>/<DATA>/jump_reads_orig.qualb
<REF>/<DATA>/jump_reads_orig.pairs
```

Los ficheros siguientes son opcionales:

`<REF>/<DATA>/long_jump_reads_orig.fastb`

`<REF>/<DATA>/long_jump_reads_orig.qualb`

`<REF>/<DATA>/long_jump_reads_orig.pairs`

Estos ficheros se pueden generar automáticamente a partir de un conjunto de ficheros BAM, fastq, fasta o fastb.

6.2.4.5.2. El Fichero `ploidy`

El fichero `ploidy` es un fichero de una sola línea que contiene un número. Como su nombre indica, este número indica el «ploide» del genoma con 1 para genomas haploides y dos para genomas diploides. Los genomas poliploides no se soportan actualmente. El nombre del fichero es de la siguiente manera:

`<REF>/<DATA>/ploidy`

6.2.4.5.3. Preparación de los Ficheros de Entrada

La manera más fácil de prepara los datos para **ALLPATHS** dados un conjunto de ficheros de la plataforma Illumina es utilizar el *script* en Perl Prepare**ALLPATHS**Inputs.pl para convertir los ficheros BAM, fasta, fastq o fastb a formatos internos de ALPATHS. También se creará de manera opcional el fichero `ploidy`.

El usuario también debe proporcionar los dos ficheros `.csv` (separados por comas) siguientes:

`in_gropus.csv`

`in_libs.csv`

Que describen, respectivamente, las posiciones y la información de la biblioteca de los diferentes ficheros que se tienen que convertir.

6.2.4.5.4. Formatos de Ficheros Aceptados

Cada fichero de datos debe contener lecturas emparejadas de una única biblioteca pero una biblioteca puede estar dividida en varios ficheros. Normalmente un fichero de datos representa un único carril (*lane*) de Illumina.

Actualmente los ficheros soportados son `.bam`, `.fastq`, `.fasta` y `.fastb`. Las lecturas de los ficheros `.fastq`, `.fasta` y `.fastb` deben estar interpoladas, es decir, a la lectura 1 del par 1 le sigue la lectura 2 del par 1 que a su vez le sigue la lectura 1 del par 2, etc. Igualmente los indicadores de calidad para los ficheros `.fasta` y `.fastb` se corresponden con los ficheros `.quala` y `.qualb` respectivamente.

Para los ficheros `.fastq` es necesario comprobar cómo se han codificado los indicadores de calidad ya que por defecto se asume que dichos indicadores se codifican utilizando códigos ASCII desde el 33 al 126. Si estos indicadores se codifican con ASCII desde 64 a 126 entonces es necesario especificarlo utilizando la opción `PHRED_64=1` cuando se utilice el script de conversión, como se verá a continuación.

6.2.4.5.5. Fichero `in_groups.csv`

Cada línea del fichero `in_groups.csv` proporciona la siguiente información para cada fichero de datos:

`group_name`: un alias único para el conjunto de datos específico.

`library_name`: la biblioteca a la que pertenecen los datos.

`file_name`: el `path` absoluto al fichero de datos.

Ejemplo de `in_gropus.csv`:

```
group_name,      library_name,file_name
302GJ,           Solexa-11541,/seq/Solexa-11541/302GJ.bam
303GJ,           Solexa-11542,/seq/Solexa-11542/303GJ.fastq
```

6.2.4.5.6. Fichero in_libs.csf

Cada línea de este fichero describe una biblioteca. Los campos específicos son:

`library_name`: coincide con el mismo campo del fichero in_groups.csv

`project_name`: una cadena de texto describiendo el proyecto.

`organism_name`: el organismo.

`type:fragment`, `jumping`, `EcoP15`, etc. Este campo es meramente informativo.

`paired`: `0`: lecturas no emparejadas; `1`: lecturas emparejadas.

`frag_size`: número medio de bases de los fragmentos (sólo para bibliotecas de fragmentos).

`frag_stddev`: desviación estándar estimada de los tamaños de las secuencias (sólo para las bibliotecas de fragmentos).

`insert_size`: número medio de bases de los insertos (solo para bibliotecas de saltos).

`insert_stddev`: desviación estándar estimada de los tamaños de los insertos (sólo para bibliotecas de saltos).

`read_orientation`: `inward` o `outward`. Las bibliotecas orientadas en modo *outward* serán invertidas.

genomic_start: índice de la primera base de las lecturas. Si no es cero, todas las bases anteriores a genomic_start serán recortadas.

genomic_end: índice de la última base de las lecturas. Si no es cero, todas las bases posteriores a genomic_end se recortaran.

A continuación se expone un ejemplo de in_libs.csv (todos los campos deben ir en una sola línea por lo como la línea sería muy larga se pone «…»).

```
library_name,    project_name,organism_name,
   type,  paired,       …

Solexa-11541,     Awesome,     E.coli,
   fragment,     1,      …

Solexa-11542,     Awesome,     E.coli,
          jumping,      1,      …


      …      read_orientation,    genomic_start,
genomic_end

      …      inward,
      0,           0

      …      outward,                  0
```

6.2.4.5.7. Ejecución del Script de Conversión

La forma más simple de ejecutar el script Prepare**ALLPATHS**Inputs.pl es de la siguiente manera:

```
Prepare ALLPATHS Inputs.pl
DATA_DIR=<full_path to REFERENCE DIR>/mydata

PICARD_TOOLS_DIR=/opt/picard/bin
```

Donde DATA_DIR es la posición del directorio DATA de **ALLPATHS** donde se irán situando las lecturas convertidas, y PICARD_TOOLS_DIR es el path a las herramientas Picard que se necesitan para la conversión de datos. Hay otras opciones que también se pueden incluir:

IN_GROUPS_CSF – utiliza otro fichero en vez de ./in_groups.csv

IN_LIBS_CSF – utiliza otro fichero en vez de ./in_libs.csv

INCLUDE_NON_PF_READS – 1: (por defecto) incluye lecturas no PF. 0: incluye solo lecturas PF.

PHRED_64 – (solo para ficheros fastq) 0: (por defecto) – los valores de calidad proporcionados están codificados en ASCII entre 33 y 126. 1: ASCII entre 64 a 126.

PLOIDY – genera el fichero ploidy. Valores válidos son 1 o 2.

HOSTS – lista de servidores que se van a utilizar para computación en paralelo. Hay que tener en cuenta que la computación en remoto requiere acceso mediante ssh (se utiliza ssh-agent/ssh-add). Por ejemplo: 2,3.host2,4.host3 genera:

- 2 procesos en la máquina localhost.
- 2 procesos en la máquina host2.
- 4 procesos en la máquina host3.

Las opciones siguientes permiten al usuario seleccionar, de manera aleatoria, una porción del número total de lecturas:

FRAG_FRAC – fracción de los fragmentos de lecturas a incluir. Por ejemplo: 30% o 0.3.

JUMP_FRAC – fracción de lecturas con saltos a incluir. Por ejemplo: 20% o 0.2.

LONG_JUMP_FRAC – fracción de lecturas con saltos largos a incluir. Por ejemplo: 90% o 0.9.

GENOME_SIZE – tamaño estimado del genoma para la estimación de la cobertura.

FRAG_COVERAGE – cobertura deseada en la biblioteca de fragmentos. Por ejemplo: 45. Se necesita utilizar GENOME_SIZE.

JUMP_COVERAGE – cobertura deseada en la biblioteca de saltos. Por ejemplo: 45. Se necesita utilizar GENOME_SIZE.

Hay que tener en cuenta algunas restricciones de las opciones anteriores. Si se utiliza FRAG_FRAC o JUMP_FRAC entonces no se puede utilizar FRAG_COVERAGE o JUMP_COVERAGE. Si se utiliza FRAG_COVERAGE o JUMP_COVERAGE entonces se debe utilizar GENOME_SIZE ya que ambos valores son necesarios para el cálculo de la fracción de lectura.

Después de la ejecución correcta de Prepare**ALLPATHS**Inputs.pl se tienen que tener los ficheros de entrada en sus sitios correctos y preparados para comenzar el ensamblaje.

6.2.4.6. Importación de Referencias

Si se ha planificado realizar evaluaciones entonces se puede importar un genoma de referencia en el directorio de flujo de trabajo a la vez que los datos de las lecturas. El genoma de referencia a importar se especifica utilizando el siguiente argumento:

REFERENCE_DIR=<directorio conteniendo la referencia>

El genoma de referencia se tiene que suministrar en dos ficheros: genome.fasta y genome.fastb. El fichero fastb es una versión binaria del fichero fasta.

Se puede convertir de fasta a fastb utilizando el módulo `Fasta2Fastb` de **ALLPATHS**.

Este argumento se ignora si existe ya un genoma de referencia en el directorio `REFERENCE`. De esta manera no se sobrescribe el genoma de referencia si ya está.

Una vez que se ha importado en el directorio `REFERECNCE`, se puede omitir el argumento `REFERENCE_DIR` en el módulo `Runallpathslg`.

En vez de utilizar el argumento `REFERENCE_DIR`, también se puede crear el directorio `REFERENCE` y copiar los ficheros del genoma de referencia a dicho directorio.

6.2.4.7. Ejecución (resumida) de **ALLPATHS**

Una vez que se han importado los datos se puede pasar a ejecutar el flujo de trabajo de **ALLPATHS** como se desee y cada vez con diferentes parámetros. Cada vez que se ejecute el flujo de trabajo de **ALLPATHS** se determinarán qué módulos son necesarios dependiendo de los parámetros que se hayan elegido y a menos que se desee sobrescribir el ensamblaje previo, en cada ejecución se creará un nuevo directorio `RUN`.

Esta sección describe brevemente los argumentos de `Runallpathslg` más comunes para ejecutar el flujo de trabajo de **ALLPATHS**.

- **Modo evaluación**: dado un genoma de referencia, el flujo de trabajo puede ejecutar evaluaciones de varias fases del proceso de ensamblaje y del ensamblaje en sí mismo. Para utiliza esta opción basta con poner:
  `EVALUATION=STANDARD`
- **Tamaño dek *K-mer***: El tamaño del *K-mer* se restringe por el tamaño del fragmento o secuencia de lectura más pequeña que aparece en los datos de lecturas que hay que ensamblar. El valor de *K* debe ser más pequeño que este tamaño y sólo se admiten ciertos valores. Para secuencias de lecturas de 100pb se sugiere un valor de *K*=96.
- **Targets**: El valor del parámetro `TARGETS` determina las operaciones que se ejecutan en el flujo de trabajo:
  `TARGETS=all` ejecuta el flujo de trabajo completo, incluyendo los módulos de evaluación.
  `TARGETS= standard` ejecuta una versión reducida del flujo de trabajo de manera que obvia varios de los módulos de evaluación
- **Paralelización**: El flujo de trabajo dispone de dos niveles de paralelización. Se pueden ejecutar dos o más módulos de manera concurrente si sus dependencias son independientes y muchos módulos individuales también son susceptibles de paralelizarse via multithreading. Por defecto sólo está activada la paralización multithread y para desactivarla completamente hay que poner el parámetro `PARALLLELIZE=False`.

6.2.4.7.1. Ejemplo

El argumento `TARGETS` de `Runallpathslg` determina si el flujo de trabajo de **ALLPATHS** se ejecuta de manera completa o importa datos y se detiene. Para ejecutar un ensamblaje utilizando datos previamente importantes se hace los siguiente:

```
TARGETS=standard
```

Por ejemplo, para los datos importados mediante `Prepare`**`ALLPATHS`**`Inputs.pl` con `DATA_SUBDIR=<user pre>/staph/mydata` hay que hacer lo siguiente:

```
Runallpathslg PRE=<user pre> DATA_SUBDIR=mydata RUN=myrun REFERENCE_NAME=statph TARGETS=standard K=96
```

Esto creará (si no existe ya) la siguiente estructura de directories:

```
<user pre>/satph/mydata/myrun
```

Donde `staph` es el directorio `REFERENCE`, `mydata` es el directorio `DATA` que contiene los datos importados y `myrun` es el directorio `RUN`.

6.2.4.7.2. Errores en el Flujo de Trabajo

El flujo de trabajo se para cuando encuentra un error que puede ser de dos tipos:

- **Error en la verificación de reglas de consistencia**: antes de que se llame a cualquier módulo, `Runallpathslg` comprueba si es posible generar todos los ficheros de salida para el ensamblaje concreto según los parámetros especificados. Si no es posible entonces el flujo de trabajo se detiene inmediatamente antes de que se ejecute cualquier módulo, generando un fichero con la información necesaria para determinar qué ha ocurrido.
- **Error en la verificación de consistencia en la ejecución**: después de que se ejecute cada módulo, el flujo de trabajo comprueba que los ficheros de salida se han generado correctamente y en caso contrario se detiene informando de los ficheros no generados y el módulo concreto que ha fallado.

Una vez que se han corregido los errores y se vuelve a ejecutar el comando `Runallpathslg`, este continúa por donde falló previamente.

6.2.4.8. Ensamblaje mediante **ALLPATHS**

6.2.4.8.1. Introducción

El script en Perl Prepare**ALLPATHS**Inputs.pl que importa datos para **ALLPATHS** es realmente un conjunto de herramientas que primero crea una caché temporal de ficheros fastb y qualb en `<DATA>/read_cache/` para cada fichero de datos descrito en in_groups.csv. Posteriormente une todos estos ficheros para generar los ficheros de entrada que necesita **ALLPATHS**.

De manera alternativa, se puede crear una caché no temporal de manera separada en diferentes sitios que puede trabajar como un repositorio de datos para diferentes proyectos. La ventaja de disponer de una caché es que separa la fase de consumo de tiempo de la conversión de ficheros de datos (especialmente los ficheros BAM) al formato fastb y qualb de la unión de los ficheros fastb y qualb en los ficheros de entrada para **ALLPATHS**. Esto es útil, por ejemplo, cuando se desea ejecutar diferentes ensamblajes basados en diferentes subconjuntos de los datos originales.

La caché de **ALLPATHS** almacena toda la información de las bibliotecas y grupos en dos ficheros en el directorio caché: `libraries.csv` y `groups.csv`. La construcción de la caché se realiza mediante los siguientes comandos:

```
CacheLibs.pl CACHE_DIR=<CACHE_DIR> IN_LIBS_CSV=in_libs.csv ACTION=Add
```

```
CacheGroups.pl CACHE_DIR=<CACHE_DIR> PICARD_TOOLS_DIR=/opt/picard/bin IN_GROUPS_CSV=in_groups.csv TMP_DIR=/large-tmp HOSTS='2,3.host2,4.host3' ACTION=ADD
```

El commando `CacheLibs.pl` simplemente añade la información de la biblioteca que hay en `in_libs.csv` en la cache `libraries.csv`. El comando `CacheGroups.pl` convierte todos los ficheros de datos descritos en `in_groups.csv` a fastb y qualb en la caché y añade las entradas correspondientes a la caché groups.csv. Las opciones más comunes son:

`CACHE_DIR` – El camino completo al directorio caché. Se puede omitir si está definida la variable de entorno **ALLPATHS**_CACHE_DIR.

`ACTION` – Ejecuta las acciones de `Add`, `List` o `Remove` con respecto a las entradas de la caché.

Las opciones de CacheLibs.pl son:

`IN_LIBS_CSV` – es un fichero alternativo a `./in_libs.csv`

Las opciones de `GroupLibs.pl` son:

`PICARD_TOOLS_DIR` – El camino completo a las herramientas Picard que se necesitan para la conversión de datos. Se puede omitir si está definida la variable de entorno **ALLPATHS**_PICARD_TOOLS_DIR.

`IN_GROUPS_CSV` – fichero alternativo al de por defecto `./in_groups.csv`

`TMP_DIR` – El camino completo del directorio temporal.

`HOSTS` – Lista de máquinas que son paralelizables mediante procesos y bifurcación. Cada bifurcación convierte un solo fichero de datos.

Los contenidos de la caché se pueden enumerar mediante el comando siguiente:

```
CacheGroups.pl CACHE_DIR=<CACHE_DIR> ACTION=List
```

6.2.4.8.2. Uso de la Caché de **ALLPATHS**

Una vez que se ha creado la caché se puede utilizar para generar los ficheros de entrada de **ALLPATHS**:

```
<DATA>/frag_reads_orig.fastb
<DATA>/frag_reads_orig.qualb
<DATA>/frag_reads_orig.pairs

<DATA>/jump_reads_orig.fastb
<DATA>/jump_reads_orig.qualb
<DATA>/jump_reads_orig.pairs
```

El comando para generar estos ficheros es el siguiente:

```
CacheTo**ALLPATHS**Inputs.pl CACHE_DIR=<CACHE_DIR> GROUPS="{12345AAXX.[1,2,3],67890ABXX.{6,7}}" DATA_DIR=<DATA_DIR> FRAG_FRAC=50% JUMP_FRAC=34%
```

Y las opciones son:

`CACHE_DIR` – El path completo al directorio caché. Se puede omitir si está definida la variable de entorno **ALLPATHS**_CACHE_DIR.

`DATA_DIR` – El camino completo al directorio `DATA` donde se situarán los ficheros de entrada.

`GROUPS` – Una lista de los grupos que se van a incluir como entrada.

`IN_GROUPS_CSV` – Fichero que incluye la descripción de los grupos. Alternativa opcional a `GROUPS`.

`FRAG_FRAC` – fracción de los fragmentos de lecturas a incluir. Por ejemplo: `30%` o `0.3`.

JUMP_FRAC – fracción de lecturas con saltos a incluir. Por ejemplo: 20% o 0.2.

LONG_JUMP_FRAC – fracción de lecturas con saltos largos a incluir. Por ejemplo: 90% o 0.9.

FRACTIONS – (se usa sólo con GROUPS) lista de fracciones, una por grupo, por ejemplo: "{0.5,30%,100%}".

GENOME_SIZE – tamaño estimado del genoma para la estimación de la cobertura.

FRAG_COVERAGE – cobertura deseada en la biblioteca de fragmentos. Por ejemplo: 45. Se necesita utilizar GENOME_SIZE.

JUMP_COVERAGE – cobertura deseada en la biblioteca de saltos. Por ejemplo: 45. Se necesita utilizar GENOME_SIZE.

LONG_JUMP_COVERAGE – (necesita el uso de GENOME_SIZE) biblioteca de saltos con la cobertura deseada, por ejemplo: 1.

COVERAGES – (se usa con GROUPS solo y requiere GENOME_SIZE) lista de coberturas, una por grupo, por ejemplo: "{45,50,2}".

Al igual que con Prepare**ALLPATHS**Inputs.pl, hay que tener en cuenta algunas restricciones de las opciones anteriores. Si se utiliza FRAG_FRAC, JUMP_FRAC o FRACTIONS entonces no se puede utilizar FRAG_COVERAGE, JUMP_COVERAGE o COVERAGES. Si se utiliza FRAG_COVERAGE, JUMP_COVERAGE o COVERAGES entonces se debe utilizar GENOME_SIZE ya que ambos valores son necesarios para el cálculo de la fracción de lectura. Si se especifica FRACTIONS y una lista de COVERAGES entonces se debe especificar una lista de GROUPS y añadir una entrada de fracción o cobertura para cada grupo.

Tras la ejecución correcta de CacheToAllPathInputs.pl se obtienen los ficheros de entrada situados en el sitio correcto para comenzar el ensamblaje.

6.2.4.9. Opciones de Compilación

Cuando se crea **ALLPATHS** mediante el comando make, se pueden añadir las siguientes opciones en la línea de comandos:

> -j<n> divide la compilación en n procesadores paralelos. Si se pone n igual al número de CPUs del servidor entonces se espera que se acelere la compilación en nx.

6.2.4.10. Flujo de Trabajo en Detalle

6.2.4.10.1. Principales Características

El flujo de trabajo de **ALLPATHS** incorpora las siguientes características principales:

- Ejecuta sólo aquellos módulos que se necesitan para un conjunto particular de parámetros.
- Asegura que los ficheros intermedios son siempre consistentes.
- Si se modifican los parámetros para un módulo entonces una nueva ejecución sólo modifica la ejecución de ese módulo en concreto y los módulos que dependan de la salida del primero.
- Si el sistema se detiene, cuando vuelva a arrancar contínua por donde se quedó previamente.
- La ejecucións se puede desplazar a cualquier punto.
- Inicialmente se pueden excluir módulos que no se requieren para el proceso de ensamblaje (por ejemplo los módulos de evaluación) y ejecutarlos fácilmente una vez que el ensamblaje esté completo.
- Determina previamente si dispone de todos los ficheros de entrada necesarios y verifica que es posible generar todos los ficheros de salida antes de ejecutar cualquier módulo. Finaliza inmediatamente cuando se detecta un problema.

6.2.4.10.2. Estructrura de Directorios (ALLPATHS_BASE)

Adicionalmente al uso del argumento de la línea de comantos PRE para especificar la localización del directorio del flujo de trabajo, se puede de manera opcional utilizar ALLPATHS_BASE de tal manera que la localización del directorio de flujo de trabajo puede ser cualquiera de los siguientes:

PRE

O bien

PRE/ALLPATHS_BASE

6.2.4.10.3. Targets

El flujo de trabajo determina qué ficheros de salida se necesita generar medianet una lista de objetivos (*targets*). Si se necesita un objetivo particular entonces se ejecutará el módulo correspondiente en el orden correcto. Adicionalmente si todos los ficheros intermedios ya existen y están al día con respecto a los ficheros de los que dependen, entonces se omite la ejecución del módulo correspondiente.

Existen dos maneras de especificar los ficheros objetivos. La más simple es utilizar uno de los pseudo objetivos predefinidos que representan a un conjuno de

ficheros objetivo. La segunda manera es especificar una lista individual de ficheros que sabe como generar el flujo de trabajo. Ambos métodos se pueden utilizar a la vez.

6.2.4.10.4. Pseudo *Targets*

Esta es la mejor manera de controlar qué ficheros creará el flujo de trabajo. El valor del pseudo objetivo se pasa al comando `Runallpathslg` de la siguiente manera:

```
TARGETS=<pseudo target name>
```

Existen cuatro posibles pseudo objetivos:

- `none` – no hay pseudo objetivos. Sólo se generan los ficheros objetivos listados explícitamente.
- `standard` – se crean los ficheros de ensamblaje y de evaluación seleccinados.
- `all` – se crean todos los ficheros objetivo, incluyendo todos los ficheros de evaluación y experimentales.

El valor por defeco es `standard`.

6.2.4.10.5. Ficheros *Targets*

Los ficheros individuales se pueden especificar como objetivos en vez de pseudo objetivos y la lista de los ficheros objetivos de cada subdirectorio del flujo de trabajo se pasan a `Runallpathslg` utilizando los siguientes parámetros:

```
TARGETS_DATA=<target files in the DATA dir>
TARGETS_RUN=<target files in the RUN dir>
TARGETS_SUBDIR=<target files in the SUBDIR dir>
```

Se pueden pasar varios ficheros objetivos de la siguiente manera:

```
TARGETS_RUN="{target1,target2,target3}"
```

6.2.4.10.6. Modo Evaluación

Dado un genoma de referencia, el flujo de trabajo puede ejecutar evaluaciones en varias etapas del proceso de ensamblaje. Ciertas evaluaciones tienen la capacidad de alterar el ensamblaje ya que requieren incorporar datos del genoma de referencia en las estructuras de datos que se utilizan en el proceso de ensamblaje y pueden no considerarse útiles desde el punto de vista del ensamblaje de novo.

Para tener en cuenta esto, el modo de evaluacións s controla mediante:

```
EVALUATION=<evaluation mode>
```

Existen tres modos de evaluación:

- `NONE` – no existe ni evaluación ni referencia disponible.
- `BASIC` – evaluación básica que no requiere de una referencia.
- `STANDARD` – se ejecutan los módulos de evaluación utilizando la referencia suministrada.
- `FULL` – la evaluación se activa en determinados módulos para no perturbar el ensamblaje.
- `CHEAT` – se activan las evaluaciones que potencialmente perturban el ensamblaje (de una forma neutral) pero permite una análisis más detallado.

El modo por defecto es BASIC.

6.2.4.10.7. El Tamaño del *K-mer*

El *K-mer* es el núcleo del ensamblaje mediante **ALLPATHS**. La elección del tamaño del *K-mer* impacta en muchos aspectos del proceso de ensamblaje.

Este tamaño se pasa a `Runallpathslg` utilizando el parámetro siguiente:

```
K=<k-mer size>
```

6.2.4.10.8. Paralelización

Cuando se tiene suficiente memoria es posible paralelizar el flujo de trabajo con el objetivo de reducir tiempo de ejecución. Existen dos maneras de paralelizar que se pueden utilizar a la vez:

- **Paralelización cross-module** – Se utiliza para módulos del flujo de trabajo que no dependan unos de otros y se puedan ejecutar de manera concurrente. Esta funcionalidad se proporciona por la herramienta make de la que hace uso `Runallpathslg` para ejecutar el flujo de trabajo. Es equivalente a la opción `-j<n>` cuando se compila el código fuente de **ALLPATHS**. Hay que tener en cuenta que no se realizan controles para asegurar que existe suficiente memoria para ejecutar múltiples módulos de **ALLPATHS** a la vez. Para poner el máximo número de módulos que se pueden ejecutar simultáneamente hay que usar el siguiente parámetro:

  ```
  MAXPAR=<n>
  ```

- **Paralización de módulos individuales**: Varios de los módulos de **ALLPATHS** se han diseñado para ejecutarse en paralelo mediante threading que es una forma independiente del modelo de paralelización anterior. Cada nivel de paralelización de módulo se puede controlar mediante argumentos en `Runallpathslg` de la siguiente manera:

  ```
  ModuleName_THREADS=<n>
  ```

Para un rendimiento máximo es conveniente poner este número al máximo de procesadores disponibles (teniendo en cuenta el aumento en el consumo de memoria que esto conlleva).

Por defecto, el flujo de trabajo intenta utilizar todos los procesarores disponibles.

Si se especifica `PARALELL=False` entonces esto pondrá a 1 el parámetro `MAXPAR`.

6.2.4.10.9. Ficheros de Log

Adicionalmente a la salida estándar, se generan varios ficheros de log en el subdirectorio `MAKEINFO`, de tal manera que cada fichero generado tiene dos ficheros de logs asociados. Por ejemplo, el fichero `hyper.fasta` tendrá los siguientes ficheros en el directorio `/SUBDIR/makeinfor`:

```
Hyper.fasta.cmd
Hyper.fasta.DumpHyper.out
```

El fichero `.cmd` contiene el comando utilizado para generar hyper.fasta. El fichero `.out` contiene la salida del módulo utilizado para crear `hyper.fasta`. En este caso el módulo se llama `DumpHyper`, tal como se puede ver en el fichero `hyper.fasta.cmd`.

6.3. EVALUACIÓN DE **EULER-SR**

6.3.1. INSTALACIÓN DE **EULER-SR**

La version actual de **EULER-SR** es la 1.1.2 que fue generada el 30 de Marzo de 2009 y se puede descargar desde la página http://**EULER**-assembler.ucsd.edu/portal Tal como vimos en el capítulo dedicado a **EULER-SR**, se trata de un programa de ensamblaje de lecturas *de novo* que utiliza un grafo de de Bruijn para construir dicho ensamblaje. Para **EULER-SR** el ensamblaje de un genoma se corresponde con el recorrido de un camino euleriano en el grafo de de Bruijn. Tanto las lecturas largas como las lecturas emparejadas se utilizan para determinar partes de dicho camino.

Para instalar la versión, desempaquetamos en el directorio deseado el fichero **`EULER-SR`**`.1.1.2.tgz`, de la siguiente manera:

```
$ tar xvfz EULER-SR.1.1.2.tgz
```

6.3.1.1. SOFTWARE BASE

El software necesario para compilar el programa es el siguiente:

- g++
- gmake

EULER-SR solo se ha probado en sistemas Linux x86-64 por lo que aunque debería ejecutarse sin problemas en otros sistemas, no está garantizado.

6.3.1.2. Variables de Entorno

Las variables de entorno que se necesitan son:

- `EUSRC`: cuyo valor debe ser el "path" donde **EULER-SR** fue descomprimido.
- `MACHTYPE`: al valor que corresponda de los siguientes: x86_64, i686 o powerpc.

Una vez tenido en cuenta lo anterior, hacemos lo siguiente:

```
$ cd EULER-SR
$ export EUSRC=`pwd`
$ export MACHTYPE=x86_64
```

6.3.1.3. COMPILACIÓN

Para compilar el programa ejecutamos lo siguiente:

```
$ gmake
```

En la compilación ejecutada en local se produce un error debido a que el fichero `SFF2Fasta.ccp` detecta una falta de librerías por lo que se tiene que editar dicho fichero e incluir la línea siguiente:

```
#include <stdint.h>
```

Una vez modificado est fichero el programa compila perfectamente.

6.3.1.3.1. COMPILACIÓN PARA CORRECCIÓN DE ERRORES EN MULTI-CORE

EULER-SR utiliza OpenMP para paralelizar la corrección de errores, por lo que si tenemos OpenMP (disponible a partir de la versión 4.0 de g++) en nuestro Linux (se puede ver ejecutando $locate omp.h) podemos compilarlo con esta opción de la siguiente manera:

```
$ gmake parallel
```

6.3.1.4. EJECUCIÓN DE LOS PROGRAMAS

Si alguno de los programas se ejecuta sin argumentos muestra una breve ayuda de su uso.

6.3.2. LIMPIEZA DE DATOS

La calidad del ensamblaje es muy dependiente de

la limpieza de los datos. Los valores de calidad se pueden utilizar para recortar/remover lecturas con baja calidad y **EULER-SR** proporciona varias herramientas para realizar esta tarea.

6.3.2.1. TRANSLACIÓN DE DATOS

EULER-SR toma como entrada un único fichero *fasta* con todas las lecturas, un fichero opcional de parejas que específica emparejamientos y un fichero de reglas que describe como asignar las entradas fasta a los tipos de clones.

Se proporcionan herramientas para trasladar de las siguientes fuentes:

- .sff (ficheros binarios de 454)
- .fastq (formato Sanger)
- .fastq (valores de calidad de Illumina)

6.3.2.1.1. FASTQ → Fasta (Sanger)

Para trasladar de *fastq* a *fasta* se utiliza el recorte de calidad a partir de `reads.fastq`.

```
$ ${EUSRC}/assembly/${MACHTYPE}/qualityTrimer -fastq reads.fastq -outFasta reads.fasta
```

Existen opciones adicionales para ajustar el recorte.

6.3.2.1.2. FASTQ → Fasta (Illuminia)

Illumina utiliza una escala diferente a la puntuación de Phred para calcular los valores de calidad y aunque `qualityTrimmer` intenta reconocer el tipo (Sanger vs. Illumina), se puede forzar mediante la línea de comandos de la siguiente manera:

```
$ $EUSRC/assembly/$MACHTYPE/qualityTrimer -fastq reads.fastq -outFasta reads.fasta -type Illumina
```

6.3.2.1.3. Sff → FASTQ

Los ficheros de salida generados por 454 están por defecto en un formato binario que contiene tanto los *fasta* como los valores de calidad y flujo. **EULER-SR** no utiliza estos valores de flujo (**NEWBLER** si). Para pasar de *sff* a *fasta*, se utiliza `sff2fasta` de la siguiente forma:

```
$ $EUSRC/assembly/$MACHTYPE/sff2fasta reads.sff reads.fasta
```

6.3.2.1.4. **ELAND** → FASTQ

Para traducir la salida de Illumina Bustard a un fichero fastq se utiliza el programa **ELAND***ToFastq* que tiene una opción para excluir las lecturas que no pasan el filtro *purity*.

```
$ $EUSRC/assembly/$MACHTYPE/ELANDToFastq reads.txt reads.fastq -printNotPure
```

6.3.2.2. Calidad del Filtrado

Si se tienen valores de calidad, se pueden recortar las secuencias con un nivel bajo de la siguiente manera:

```
$ $EUSRC/assembly/$MACHTYPE/qualityTrimmer -fasta reads.fasta -qual reads.fasta.qual -outFasta reads.trimmed
```

La sensibilidad del recorte se puede ajustar con `-span S -minQual Q`, donde las lecturas se recortarán de tal manera que cada lectura tendrá una calidad mínima de *Q* para cada *S* nucleótidos consecutivos.

6.3.2.3. Filtrado de Lectura Illumina Erróneas

Las lecturas Illumina (particularmente las de la primera generación) tienen unos valores particulares de salida tal como el siguiente ejemplo:

```
>read1
ACGGCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

Estas lecturas se pueden filtrar utilizando el programa `filterIlluminaReads` de la siguiente manera:

```
$ $EUSRC/assembly/$MACHTYPE/filterIlluminaReads reads.fasta reads.filt.fasta
```

6.3.2.4. Preparación de lecturas 454 emparejadas

Las lecturas 454 emparejadas se preparan enlazando los extremos de un clon con un *linker*, cortando los clones, extrayendo las secuencias enlazadas y secuenciado mediante *Lectura1-linker-Lectura2*.

Para dividirlas se necesita conocer la secuencia del *linker* que está en el fichero `linker.fasta` de la siguiente manera:

```
$ $EUSRC/assembly/$MACHTYPE/splitLinkedClones reads.454.fasta linker.fasta reads.454.fasta.split -singletons reads.454.fasta.singletons
```

6.3.2.5. Limpieza del vector y enmascaramiento de secuencias de lecturas Sanger

En Sanger pueden causar problemas el vector y las secuencias enmascaradas debido a errores de corrección y al ensamblaje. Por ello es conveniente

filtrarlas mediante dos herramientas de la siguiente manera:

```
$ cat reads.fasta
$EUSRC/assembly_utils/RemoveMaskedSequence.pl |
$EUSRC/assembly_utils/RemoveVectorSequence.pl >
reads.trimmed
```

6.3.3. EJECUCIÓN DEL ENSAMBLAJE

6.3.3.1. Preparación del fichero de entrada

Si se dispone de varios ficheros de entrada, es necesario concatenarlos previamente en un sólo fichero. Una manera fácil es la siguiente:

```
$ cat fichero1.fasta fichero2.fasta ….
ficheroN.fasta > reads.fasta
```

6.3.3.2. ¿Qué tamaño de *K-mer* escoger?

Una parte fundamental del grafo de de Bruijn es el tamaño de *K* utilizado que está relacionado directamente con la longitud mínima de solapamiento entre lecturas. En la versión actual *K* debe ser menor que 29 y normalmente se escoge tal que $25 \leq k \leq 28$.

6.3.3.3. Ejecución de un ensamblaje por defecto

Este se puede realizar de la siguiente manera:

```
$ $EUSRC/assembly/Assembl.pl reads.fa 25
```

Donde 5 es el tamaño del *K-mer*.

6.3.3.4. Ejecución de un ensamblaje con lecturas emparejadas

Para este caso es necesario generar un fichero de reglas que describa las parejas que se utilizan. El fichero tiene una expresión regular entre comillas seguida de dos palabras claves obligatorias: CloneLength y CloneVar. Los emparejamientos se utilizan en el orden que se han especificado en el fichero de reglas y para un resultado óptimo las reglas asociadas a los clones deberían especificarse en orden incremental al tamaño del clon.

Ej: Un ensamblaje de emparejamientos se puede ejecutar de la siguiente manera teniendo las lecturas en un fichero denominado `reads.fa` y el fichero de reglas en `readtitle.rule`:

```
$ $EUSRC/assembly/Assemble.pl reads.fa 25 -
ruleFile readtitle.rules
```

Un ejemplo de regla es la siguiente:

```
"([^/]*)/([12])"                CloneLength=200
CloneVar=50
```

En general, el formato de un par de lecturas debe ser:

```
CLONE_NAME.DIR1
CLONE_NAME.DIR2
```

Donde CLONE_NAME es el nombre único de cada clon y DIR1 y DIR2 indican de qué lado del clon se están leyendo las lecturas.

Cada expresión regular necesita dos coincidencias, una para el nombre del clon y la otra para la dirección; y no se tienen que tener más de dos lecturas para el mismo nombre de clon.

La palabra clave `CloneLength` es el hueco esperado desde el final de la primera lectura al comienzo de la segunda, o la longitud del fragmento de ADN del que se han secuenciado las dos lecturas (la suma de las longitudes de las dos lecturas).

La palabra clave `CloneVar` es la ventana esperada de variación de las longitudes de los clones. Se calcula el hueco estimado entre dos lecturas y si es manor que `CloneLength + CloneVar` o menor que `CloneLength – CloneVar` entonces el emparejamiento se considera inválido (puede ser un clon quimérico) y, por lo tanto, no se utiliza.

Ejemplos de ficheros de parejas pueden ser:

Para clones 454:

```
>000556 0922 1963.a
ACTGGCGAGAGCCCAGACGT...
>000556_0922_1963.b
GCCCGAGACCGGACTGGGAT...
```

Y la regla para este caso es:

```
"([0-9]+_[0-9]+_[0-9]+)\.([ab])"
CloneLength=2500 CloneVar=500 Type=6
```

Para clones Illumina:

```
>SLXA-EAS1_89:3:1:715:750/1
GTCTTGAAAGCTATGATGTCAAGATTAATTTAATC
>SLXA-EAS1_89:3:1:715:750/2
GTGTATTGCTCAATCTTCGAACGGGGGGAGGATTG
```

Y la regla para este caso es:

```
"([^/]*)/([12])"                CloneLength=200
CloneVar=50  Type=1           #  Emparejamientos
Illumina
```

6.3.3.5. Uso de múltiples *cores*

Si el programa se ha compilado para múltiples cores utilizando `gmake parallel` en vez de `gmake` entonces para usar tres *cores* se puede ejecutar de la

siguiente manera:

```
$ $EUSRC/assembly/Assemble.pl reads.fa 25 -numJobs 3
```

6.3.3.6. Ensamblajes no estándar

6.3.3.6.1. Detección de variaciones

En la detección de variantes no es necesario ejecutar todos los pasos de un ensamblaje. Hay que tener en cuenta que dicha detección está aún en fase de desarrollo por lo que estos pasos se tienen que ejecutar de forma manual aunque ya si existe una herramienta de post-procesado que genera variantes similares o parecidas.

A continuación se muestra un ejemplo de cara a detectar variantes utilizando un fichero `reads.fasta`:

a) Reparación de errores:

```
$ $EUSRC/assembly/FixErrors.pl reads.fasta 25
```

b) Construcción del grafo de de Bruijn:

```
$ $EUSRC/assembly/assemblesec.pl reads.fasta.fixed -vertexSize 25
```

c) Eliminación de errores en el grafo

```
$ mkdir directorio
```

```
$ $EUSRC/assembly/$MACHTYPE/simplifyGraph reads.fasta.fixed directorio/reads.fasta -minEdgeLength 75 -removeLowCoverage 5 3
```

d) Ejecución del post-procesamiento para detectar variantes

```
$ $EUSRC/assembly/${MACHTYPE}/printVariants directorio/reads.fasta reads.variants.fasta
```

Esto producirá un fichero fasta denominado `reads.variants.fasta` que contiene variantes con el siguiente formato:

```
>SHARED_EDGE INDEX (length, read support) VARIANT_INDEX (length, support) [... VARIANT_INDEX2 (length2, support2)] SHARED_EDGE2(length,support )
```

Esto genera una variante y las secuencias que la acompañan de tal manera que la variante puede ser fácilmente alineada contra una referencia. Los números en los títulos FASTA son: EDGE_INDEX (LENGTH, SUPPORT), donde EDGE_INDEX es el índice del enlace en el fichero `.edge` (lo mismo que en los *contigs*, pero los enlaces existen para las dos orientaciones: directa e inversa), la longitud del enlace y el soporte como el número de lecturas utilizadas para construir en enlace en el ensamblaje. Un soporte muy corto indica que la variante es, probablemente, errónea. En el siguiente ejemplo, hay un *indel* de longitud 6 en el ensamblaje:

```
>0(1855, 2957) 21(49, 44) 8(2274, 3692)

AGAAAGGGATTTTAGTTTGTAATATCGCAGCAAGTCGATTGATTTTACCGTCTCCCAATGCATTCAAAGATAGTATTGTA

AAAATCTCAGTTGGTGAAGAATATGATCAACACGCGTTTATCCATCAGTTAAAGGAAAATGGCTATCGAAAAGTTACTCA

AGTACAAACTCAGGGCGAATTTAGTCTTCGAGGAGATATTATTTTTGAAATATCCCAGTTAGAACCTTGTCGAATTGAGT

TTTTTGGTGATGAAATTGATGGTATCAGGTCATTTGAAGTAGAAACACAATTATCGAAAGAAAATAAGACAGAACTCACT

ATCTTTCCAGCTAGTGATATGCTTTTGAGAGAAAAGGATTATCAACGAGGACAGTCAGCTTTAGAAAAACAAATTTCAA

>0(1855, 2957) 22(55, 39) 8(2274, 3692)

AGAAAGGGATTTTAGTTTGTAATATCGCAGCAAGTCGATTGATTTTACCGTCTCCCAATGCATTCAAAGATAGTATTGTA

AAAATCTCAGTTGGTGAAGAATATGATCAACACGCGTTTATCCATCAGTTAAAGGAAAATGGCTATCGAAAAGTTACTCA

AGTACAAACTCAGGGCGAATTTAGTCTTCGAGGAGATATTTTAGATATTTTTGAAATATCCCAGTTAGAACCTTGTCGAA

TTGAGTTTTTTGGTGATGAAATTGATGGTATCAGGTCATTTGAAGTAGAAACACAATTATCGAAAGAAAATAAGACAGAA

CTCACTATCTTTCCAGCTAGTGATATGCTTTTGAGAGAAAAGGATTATCAACGAGGACAGTCAGCTTTAGAAAAACAAAT

TTCAA


Query                                                 1
AGAAAGGGATTTTAGTTTGTAATATCGCAGCAAGTCGATTGATTTTACCGTCTCCCAATG  60

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||
Sbjct                                                 1
AGAAAGGGATTTTAGTTTGTAATATCGCAGCAAGTCGATTGATTTTACCGTCTCCCAATG  60


Query                                                61
CATTCAAAGATAGTATTGTAAAAATCTCAGTTGGTGAAGAATATGATCAACACGCGTTTA  120

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||
Sbjct                                                61
CATTCAAAGATAGTATTGTAAAAATCTCAGTTGGTGAAGAATATGATCAACACGCGTTTA  120


Query                                               121
TCCATCAGTTAAAGGAAAATGGCTATCGAAAAGTTACTCAAGTACAAACTCAGGGCGAAT  180
```

```
||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||

   Sbjct                                          121
TCCATCAGTTAAAGGAAAATGGCTATCGAAAAGTTACTCAAGTACAA
ACTCAGGGCGAAT  180


   Query      181      TTAGTCTTCGAGGAGATAT------
TATTTTTGAAATATCCCAGTTAGAACCTTGTCGAA  234

                ||||||||||||||||||
||||||||||||||||||||||||||||||||||

   Sbjct                                          181
TTAGTCTTCGAGGAGATATTTTAGATATTTTTGAAATATCCCAGTTA
GAACCTTGTCGAA  240


   Query                                          235
TTGAGTTTTTTGGTGATGAAATTGATGGTATCAGGTCATTTGAAGTA
GAAACACAATTAT  294

||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||

   Sbjct                                          241
TTGAGTTTTTTGGTGATGAAATTGATGGTATCAGGTCATTTGAAGTA
GAAACACAATTAT  300


   Query                                          295
CGAAAGAAAATAAGACAGAACTCACTATCTTTCCAGCTAGTGATATG
CTTTTGAGAGAAA  354

||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||

   Sbjct                                          301
CGAAAGAAAATAAGACAGAACTCACTATCTTTCCAGCTAGTGATATG
CTTTTGAGAGAAA  360


   Query                                          355
AGGATTATCAACGAGGACAGTCAGCTTTAGAAAAACAAATTTCAA
399

||||||||||||||||||||||||||||||||||||||||||||||||

   Sbjct                                          361
AGGATTATCAACGAGGACAGTCAGCTTTAGAAAAACAAATTTCAA
405
```

6.3.3.6.2. Puesta en común rápida de lecturas

Si el ratio de error de las lecturas es bajo y todo lo que se necesita es una agrupación de lecturas que tengan *K-mer* compartidos entonces basta con ejecutar `assemblesec`:

```
$ $EUSRC/assembly/assemblesec.pl
reads.fasta.  -vertexSize 25
```

Se puede incrementar la especificidad aumentando el tamaño del vértice aunque la ejecución con tamaño de vértices superiores a 28 es muy lenta.

Es necesario analizar el fichero `reads.fasta.intv` para determinar las lecturas que provocan solapamientos. Este fichero tiene el formato siguiente:

```
EDGE E Length L Multiplicity M
INTV 1 0 50 135
INTV 8 0 50 140
INTV a b c  d
...
EDGE ...
INTV ...
```

El enlace *E* en el ensamblaje se corresponde con cualquier *contig* (directo o inverso). La longitud se refiere a la longitud del enlace y *M* es el número de lecturas de ese enlace *E* incluidas en el ensamblaje.

Cada *INT* corresponde a una porción de una lectura que se asigna a un enlace. En *INTV a b c d*, *a* es el índice de la lectura. Los índices impares numerados son el complemento inverso de una lectura incluida en los datos de tal manera que la lectura 0 es la lectura del fichero de entrada y la lectura 1 es el complemento inverso de 0 y así sucesivamente. *b* es la posición de comienzo de la lectura asignada al enlace, *c* es la longitud del segmento de la lectura asignada al enlace y *d* es la posición a lo largo del enlace donde la lectura está asignada.

Cuando *c* es la longitud de la lectura entonces la lectura completa se asigna al enlace. Cuando es menor, la lectura o bien se asigna a dos enlaces o bien parte de la lectura no se asigna al ensamblaje (normalmente cuando hay errores en la lectura).

6.3.4. EXAMEN DE LA SALIDA Y REFINAMIENTO DE LOS ENSAMBLAJES

Algunos ensamblajes se pueden mejorar iterativamente simplificando el grafo o, en otro caso, determinar que los datos suministrados son insuficientes para producir un ensamblaje válido.

6.3.4.1. DETECCIÓN DE ENSAMBLAJES FRAGMENTADOS

Si se tienen lecturas emparejadas, el ensamblaje se generará en el directorio `matetransformed` y en otro caso en el directorio `transformed`. Los siguientes

ejemplos utilizan el directorio `transformed`:

Ej: Imprime todos los enlaces *muertos*:

```
$ $EUSRC/assembly/$MACHTYPE/printGraphSummary transformed/rea ds.fasta -sources
```

Esto imprimirá los índices de los enlaces seguido por sus longitudes. Si hay un número muy alto de enlaces *muertos* entonces la cobertura será probablemente baja y causará un ensamblaje fragmentado. Algunas plataformas de secuenciación, tal como Illumina, generan bastantes zonas sin cobertura (desiertos) por ejemplo en regiones ricas en GC.

6.3.4.2. Unión de ensamblajes fragmentados

Si existen muchos enlaces *muertos* se puede intentar repararlos y unirlos cuando la cobertura es menor que el tamaño del vértice pero mayor que una longitud mínima, como por ejemplo 10. Para ellos podemos hacerlo de la siguiente manera:

```
$ $EUSR/assembly/${MACHTYPE}/

joinsas transformed/reads.fasta 10 transformed/reads.j

$ $EUSRC/assembly/$MACHTYP}/printContigs transformed/reads.j

$ cp transformed/reads.j.contig
```

El caso anterior buscará uniones ambiguas de enlaces que tengan al menos 10 nucleótidos coincidentes.

6.3.5. PRUEBA DE EJECUCIÓN DE **EULER-SR**

Realizamos prueba básica de **EULER-SR** con 1 fichero (reads.fasta) de 142.858 lecturas cortas de 35pb.

Declaramos las dos variables de entorno:

- `EUSRC`: donde está el directorio de **EULER-SR**. en nuestro caso `EUSRC=~/Ensambladores/`**`EULER-SR`**`.1.1.2`
- `MACHTYPE=x86_64`. Tipo de procesador

Ejecutando en ensamblaje obtenemos lo siguiente:

```
mmhr@ubuntu:~/Ensambladores/EULER-SR.1.1.2$ Assemble.pl reads.fasta 21mkdir -p fixed; /home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/readsToSpectrum reads.fasta 21 fixed/reads.fasta.spect -printCount

/home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/estErrorDist fixed/reads.fasta.spect -binary

/home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/sortIntegralTupleList fixed/reads.fasta.spect -printCount -minMult 0

Using minimum multiplicity: 5

/home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/fixErrors reads.fasta fixed/reads.fasta.spect 21 fixed/reads.fasta -minMult 5 -maxScore 3 -startScore 2 -stepScore 1 -minVotes 2 -edgeLimit 3 -replaceN

/home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/assemblesec.pl fixed/reads.fasta -vertexSize 21

mkdir -p simple; /home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/simplifyGraph fixed/reads.fasta simple/reads.fasta -minEdgeLength 84 -removeBulges 84 -removeLowCoverage 5 3

mkdir -p transformed; /home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/transformGraph simple/reads.fasta transformed/reads.fasta -minPathCount 3 -notStrict -erodeShortPaths 21

/home/mmhr/Ensambladores/EULER-SR.1.1.2/assembly/x86_64/printContigs transformed/reads.fasta; cp transformed/reads.fasta.contig .
```

El resultado genera un directorio denominado `fixed` donde guarda los siguientes ficheros:

```
mmhr@ubuntu:~/Ensambladores/EULER-SR.1.1.2/fixed$ ll

total 34200

-rw-r--r-- 1 mmhr mmhr  8613922 2011-04-05 12:41 reads.fasta

-rw-r--r-- 1 mmhr mmhr   241981 2011-04-05 12:41 reads.fasta.bgraph

-rw-r--r-- 1 mmhr mmhr   532096 2011-04-05 12:41 reads.fasta.dot

-rw-r--r-- 1 mmhr mmhr   406749 2011-04-05 12:41 reads.fasta.edge

-rw-r--r-- 1 mmhr mmhr   213472 2011-04-05 12:41 reads.fasta.graph

-rw-r--r-- 1 mmhr mmhr  5519939 2011-04-05 12:41 reads.fasta.intv

-rw-r--r-- 1 mmhr mmhr  3485476 2011-04-05 12:41 reads.fasta.iovp

-rw-r--r-- 1 mmhr mmhr  3047179 2011-04-05 12:41 reads.fasta.path

-rw-r--r-- 1 mmhr mmhr      727 2011-04-05
```

```
12:41 reads.fasta.report

   -rw-r--r-- 1 mmhr mmhr 12936212 2011-04-05
12:41 reads.fasta.spect
```

6.4. EVALUACIÓN DE ***SOAPDENOVO***

La versión actual de **SOAP*DENOVO*** es la 1.05 que se descarga de http://soap.genomics.org.cn/SOAPDENOVO.html

SOAP*DENOVO* está diseñado especialmente para ensamblar lecturas cortas de Illumina GA de genomas animales, plantas y bacterias. Se ejecuta en sistemas Linux de 64-bits con un mínimo de 5GB de memoria física y para grandes genomas (como el humano) se necesitan alrededor de 150GB de memoria.

El sistema soporta grafos de hasta 127 *K-mer* y lo que hace es proporcionar tres versiones del programa, de acuerdo al número de *K-mer* a utilizar, de tal manera que:

- La versión de 31mer soporta hasta 31.
- La versión de 63mer soporta hasta 63 y dobla en consumo de memoria de la versión anterior, incluso aunque se utilice con un *K-mer* menor o igual a 31.
- La versión de 127mer soporte hasta 127 y dobla el consumo de memoria de la versión de 63mer incluso aunque se utilicen menos de 63.

Hay que tener en cuenta que cuanto mayor sea la longitud del *K-mer* menor es la cantidad de nodos que se generan en el grafo por lo que el consumo de memoria no llega exactamente a doblar a la versión inferior de la que se utilice.

En esta versión se ha añadido un módulo denominado "pregraph" que realiza una reserva previa de memoria para evitar una reasignación posterior. La unidad de este parámetro se mide en GB y sin esta reasignación posterior **SOAP*DENOVO*** se ejecuta más rápido y puede consumir toda la memoria que se le preasigne. También se han introducido instrucciones SIMD para mejorar el rendimiento y se deja de soportar la versión de 32 bits.

En esta versión las bases que se rellenan con huecos se representan en minúsculas en el fichero "scafSeq".

6.4.1. INSTALACIÓN

Una vez descargado, se desempaqueta mediante el comando `tar` y dentro del directorio generado se compila mediante `make`. Como resultado, en el directorio `bin`, se generan tres ejecutables:

```
$$DIRECTORIObin/SOAPDENOVO-127mer

$$DIRECTORIObin/SOAPDENOVO-31mer

$$DIRECTORIObin/SOAPDENOVO-63mer
```

Que se corresponden con las tres versiones del programa.

6.4.1.1. FICHERO DE CONFIGURACIÓN

Los datos de proyectos de grandes genomas con alta secuenciación se organizan normalmente en varios ficheros de secuencias generados desde varias librerías. El fichero de configuración tiene el objetivo de informar al ensamblador donde encontrar estos ficheros y la información relevante que le haga falta.

El fichero de configuración tiene una sección para información general y varias secciones para las librerías. En esta versión se incluye el parámetro `max_rd_len` en la sección general que tiene como función cortar cualquier lectura mayor que el valor del parámetro.

Cada sección de librería comienza con la etiqueta `[LIB]` e incluye los siguientes parámetros:

- `avg_ins`: indica el tamaño medio del inserto de esta librería o la posición del valor pico en la distribución del tamaño del inserto.
- `reverse_seq`: toma el valor 0 o 1 en función de si las lecturas necesitan ser invertidas complementariamente. Esto es debido a que el Illumina GA genera dos tipos de librerías de empajeramientos (*paired-end*):
- `forward-reverse`: generadas de los finales del ADN fragmentado con tamaño del inserto menor de 500 pb. `reverse_seq=0`
- `Forward-forward`: generadas de librerías circulares con un tamaño típico del inserto mayor de 2kb. `reverse_seq=1`
- `asm_flags`: este parámetro indica donde se utilizan las lecturas. 1 para ensamblaje de *contigs*, 2 para ensamblaje de *supercontigs*, 3 para ensamblaje de *contigs* y *supercontigs* y 4 para cierre de huecos.
- `rd_len_cutoff`: el ensamblador cortará las lecturas de la librería a esta longitud.
- `rank`: toma valores enteros y decide en qué orden se utilizan las lecturas para el ensamblaje de *supercontigs*. Las librerías con el mismo `rank` se utilizan a la vez durante en ensamblaje de *supercontigs*.
- `pair_num_cutoff`: este parámetro es el valor de corte del número de pares para una conexión confiable entre dos *contigs*.

- `map_len`: tiene efecto en la fase de asignación (*mapping*) y es la longitud mínima de alineamiento entre un *contig* y una lectura confiable.

El ensamblador acepta ficheros de lecturas en dos formatos: FASTA o FASQ. Las relaciones de emparejamientos (*mate-pair*) se pueden indicar de dos maneras: dos ficheros de secuencias con lecturas en el mismo orden que pertenecen a un par o dos lecturas adyacentes que están emparejadas en un único fichero (sólo en FASTA).

En el fichero de configuración los finales de fichero se indican mediante `f=/path/filename` o `q=/path/filename` para formatos FASTA o FASTQ, respectivamente. Las lecturas emparejadas que están en dos ficheros de secuencias FASTA se indican mediante `f1=` y `f2=`. De igual manera, las lecturas emparejadas en dos ficheros FASTQ se indican por `q1=` y `q2=`. Las lectura emparejadas en un único fichero de secuencias FASTA se indican por `p=`.

Todos los parámetros para la sección de librerías son opcionales y el ensamblador asigna valores por defecto para la mayoría de ellos. Si no se está seguro de un parámetro, se puede eliminar del fichero de configuración.

6.4.2. ARRANQUE

Una vez que está definido el fichero de configuración, una manera típica de ejecutar el ensamblador es:

```
${bin} all -s config_file -K 63 -R -o graph_prefix
```

También se puede ejecutar paso a paso:

```
${bin} pregraph -s config_file -K 63 [-R -d -a] -o graph_prefix
```

```
${bin} contig -g graph prefix [-R -M 1 -D]
```

```
${bin} map -s config_file -g graph prefix [-p]
```

```
${bin} scaff -g graph_prefix [-F -u -G -p]
```

6.4.2.1. OPCIONES

-a ENTERO Inicia la asignación de memoria (en GB) para evitar una reasignación posterior.

-s CADENA fichero de configuraicón.

-o CADENA fichero del grafo de salida

-g CADENA fichero del grafo de entrada

-k ENTERO tamaño del *K-mer* (por defecto 23, mínimo 13, máximo 127)

-p ENTERO "multithreads" (por defecto 8)

-R utiliza lecturas para resolver repeticiones pequeñas (por defecto no).

-d ENTERO elimina los *K-mer* con baja frecuencia por debajo del valor (por defecto 0)

-D ENTERO elimina los enlaces con cobertura no mayor que el valor (por defecto 1)

-M ENTERO capacidad de unir secuencias similares durante la generación de *contigs* (por defecto 1, mínimo 0 y máximo 3).

-F cierre de hueco intra-*supercontigs* (por defecto no).

-u visibilidad de los *contigs* con alta cobertura antes de generar los *supercontigs* (por defecto se ocultan)

-G ENTERO permite diferentes longitudes entre el relleno de huecos.

-L mínima longitud de los *contigs* para generar *supercontigs*.

6.4.2.2. FICHEROS DE SALIDA

Se generan los siguientes ficheros:

`*.contig`: secuencias de *contigs* sin información de emparejamientos.

`*.scafSeg`: secuencias de *supercontigs*.

Para el comando `pregraph` se generan los siguientes ficheros:

`*.kmerFreq`: cada fila muestra el número de *K-mer* con una frecuencia igual que el número de fila.

`*.edge`: cada registro muestra información de un enlace en el pre-grafo: longitud, *K-mer* en ambos extremos, cobertura media por *K-mer*, si es idéntica la inversa complementaria y la secuencia.

`*.markOnEdge` y `*.path`: estos dos ficheros se utilizan para resolver pequeñas repeticiones.

`*.preArc`: conexiones entre enlaces que se han establecido por caminos de lecturas.

`*.vertex`: *K-mer* en los extremos de los enlaces.

`*.preGraphBasic`: información básica acerca del pre-grafo: número de vértices, valor de *K*, longitud máxima de las lecturas, etc.

Los ficheros del comando `contig` son:

`*.contig`: información de los *contigs* referentes al índice de enlaces correspondiente, longitud, cobertura de los *K-mer* y la secuencia. Si se incluye el inverso complementario entonces se genera un fichero "`*.ContigIndex`".

*.Arc: arcos que salen de cada enlace y su cobertura de lecturas correspondiente.

*.update.edge: información básica de cada enlace.

Los ficheros del comando *map* son:

*.peGrads: información para cada librería de clones.

*.readOnContig: localizaciones de las lecturas en los *contigs*.

*.readInGab: este fichero contiene lecturas que se pueden localizar en huecos entre los *contigs*. Esta información se utilizará posteriormente para cerrar los huecos en los *supercontigs*.

Los ficheros del comando "scaff" son:

*.newContigIndex: orden de los *contigs* en función de su longitud previa a la generación de los *supercontigs*.

*.links: enlaces entre *contigs* basados en emparejamientos de lecturas.

*.scaf_gap: *contigs* encontrados en huecos mediante el grafo de *contigs*.

*.scaf: *contigs* para cada *supercontig*.

*.gapSeq: secuencias de huecos entre *contigs*.

El siguiente es un ejemplo del fichero de configuración:

```
#maximal read length
max_rd_len=50
[LIB]
#average insert size
avg_ins=200
#if sequence needs to be reversed
reverse_seq=0
#in which part(s) the reads are used
asm_flags=3
#use only first 50 bps of each read
rd_len_cutoff=50
#in which order the reads are used while scaffolding
rank=1
# cutoff of pair number for a reliable connection (default 3)
pair_num_cutoff=3
#minimum aligned length to contigs for a reliable read location (default 32)
map_len=32
#fastq file for read 1
q1=/path/**LIBNAMEA**/fastq_read_1.fq
#fastq file for read 2 always follows fastq file for read 1
q2=/path/**LIBNAMEA**/fastq_read_2.fq
#fasta file for read 1
f1=/path/**LIBNAMEA**/fasta_read_1.fa
#fastq file for read 2 always follows fastq file for read 1
f2=/path/**LIBNAMEA**/fasta_read_2.fa
#fastq file for single reads
q=/path/**LIBNAMEA**/fastq_read_single.fq
#fasta file for single reads
f=/path/**LIBNAMEA**/fasta_read_single.fa
#a single fasta file for paired reads
p=/path/**LIBNAMEA**/pairs_in_one_file.fa
[LIB]
avg_ins=2000
reverse_seq=1
asm_flags=2
rank=2
# cutoff of pair number for a reliable connection
#(default 5 for large insert size)
pair_num_cutoff=5
#minimum aligned length to contigs for a reliable read location
#(default 35 for large insert size)
map_len=35
q1=/path/**LIBNAMEB**/fastq_read_1.fq
q2=/path/**LIBNAMEB**/fastq_read_2.fq
q=/path/**LIBNAMEB**/fastq_read_single.fq
f=/path/**LIBNAMEB**/fasta_read_single.fa
```

6.5. EVALUACIÓN DE **VELVET**

La version actual es la 1.0.19 y se puede descargar de http://www.ebi.ac.uk/~zerbino/**Velvet**

En esa misma página existe un enlace al manual para la instalación y configuración del programa.

Se recomienda tener al menos 12Gb de memoria y funciona en cualquier plataforma Linux de 64bits

aunque, en teoría, también debe funcionar en plataformas de 32 bits pero puede dar problemas con las limitaciones de memoria.

6.5.1. OPCIONES DE COMPILACIÓN

6.5.1.1. **Velvet** PARA ESPACIO DE COLOR

Para la compilación de los ejecutables realizamos el siguiente paso en el directorio de descarga:

```
$make
```

Si se quiere genera una versión para espacio de color de SOLiD entonces compilamos con la siguiente opación:

```
$make color
```

A partir de aquí todo es igual independientemente de qué versión se utilice y sólo cambia que los ejecutables se llaman `velveth_de` y `velvetg_de`.

Hay que tener en cuenta que las versiones para espacio de color y secuencias son incompatibles por lo que si se desea utilizar las dos, es necesario separar los ejecutables y no se pueden generar *hash* de ficheros de secuencias con la versión para espacio de color y viceversa.

6.5.1.2. CATEGORIAS

Debido al uso de matrices de longitud fija es necesario definir un conjunto de variables en tiempo de compilación. Una de ellas es el número de canales o categorías de lecturas que pueden ser manipuladas de manera independiente. Esto es útil cuando se quieren distinguir lecturas que provienen de diferentes librerías de insertos.

Por defecto, hay sólo dos categorías de lecturas cortas pero esta variable se puede definir con un valor a conveniencia, de la siguiente manera:

```
$ make 'CATEGORIES=57'
```

(Hay que tener en cuenta las comillas simples y la ausencia de espacios)

Cuanto mayor sea el número de categorías, mayor será el uso de memoria.

6.5.1.3. maxkmerlentgh

Esta variable es el valor máximo de la denominada longitud *hash* que es la longitud de los *K-mer* que se indexan en la tabla *hash*. Para definir un buen valor es necesario tener en cuenta las siguientes restricciones:

- Debe ser un valor impar para evitar palíndromos. Si se pone un valor para, Velve lo incrementará de manera automática.
- Debe ser menor o igual que la longitud `MAXKMER`*`HASH`* ya que se almacena en 64 bits.
- Debe ser estrictamente menor que la longitud de lectura ya que, en otro caso, no se producirán solapamientos entre las lecturas.

Por defecto, esta longitud está limitada a 31pb pero se puede poner un límite ajustando el parámetro de la siguiente manera:

```
$make 'MAXKMERLENGTH=57'
```

Hay que tener en cuenta las comillas simples y la ausencia de espacios.

Si se escogen valores grandes, **Velvet** utilizará más memoria.

6.5.2. INSTRUCCIONES DE EJECUCIÓN

6.5.3. EJECUCIÓN DE **VELVET**H

`velveth` tiene la función de preparar el conjunto de datos para el siguiente programa: `velvetg` e indicar al sistema que representa cada fichero de secuencias. La lista de opciones se puede ver ejecutando el programa sin argumentos, de la siguiente manera:

```
$./velveth
```

`velveth` toma un número de ficheros de secuencias, produce una tabla *hash*, genera dos ficheros de salida en un directorio concreto (creándolo si es necesario), las secuencias y las hojas de ruta que son necesarias para `velvetg`. La sintaxis es la siguiente:

```
$./velveth directorio_salida longitud_hash [[-formato_fichero] [-tipo_lectura] nombre_fichero
```

La longitud *hash*, también conocida como la longitud *K-mer*, se corresponde con la longitud, en pares de bases, de las palabras que se van a indexar.

Los formatos de ficheros soportados son:

- fasta (por defecto)
- fastq
- fasta.gz
- fastq.gz
- ELAND
- geradl

Las categorías de lecturas son:

- short (por defecto)
- shortPaired
- short2 (igual que short pero para una librería

separada)

- shortPaired2 (ver arriba)
- long (para Sanger, 454 o cualquiera con secuencias de referencias)
- longPaired

Por concisión, las opciones son estables, es decir, son verdaderas hasta que se contradigan por otro operador. Esto permite escribir tantos nombres de ficheros como se desee sin tener que reescribir descriptores idénticos. Por ejemplo:

```
$./velveth directorio_salida/ 21 -fasta -short solexa1.fa solexa2.fa solexa3.fa -long capilar.fa
```

En este ejemplo, todos los ficheros se considera que están en formato FASTA y solo cambia la categoría de lectura. De todas maneras, como las opciones por defecto son `fasta` y `short`, entonces el ejemplo anterior se puede escribir de la siguiente manera:

```
$./velveth directorio_salida/ 21 solexa*.fa -long capilar.fa
```

Se puede utilizar `velveth` directamente en la salida de otro programa utilizando el carácter "-" para representar la entrada estándar de la línea de comandos. Por ejemplo, si deseamos realizar las siguientes tareas:

```
$miprograma > lecturas_seleccionadas.fa
```

```
$velveth directorio 21 -fastq otras_lecturas.fastq -fasta lecturas_seleccionadas
```

```
$velvetg directorio (… parámetros …)
```

Se puede simplificar de la siguiente manera:

```
$miprograma | velveth directorio 21 -fastq otras_lecturas.fastq -fasta -
```

```
$velvetg directorio (… parámetros …)
```

Si se está utilizando un protocolo de secuenciación de transcriptoma específico, es posible utilizar la siguiente opción para obtener mejores resultados:

```
$./velveth directorio 21 (… ficheros …) -strand_specific
```

Esta opción aplica a todos los datos.

6.5.3.1. Ejecución de `velvetg`

`velvetg` es el núcleo de **Velvet**, donde se construye el grafo de de Bruijn y se manipula posteriormente. Hay que tener en cuenta que aunque `velvetg` salva algunos ficheros durante el proceso para evitar volver a recalcular ciertos datos, los parámetros no se guardan de una ejecución a otra, por lo que:

```
$./velvetg directorio_salida -cov_cutoff 4
```

```
$./velvetg directorio_salida -min_contig_lgth 100
```

Es diferente de:

```
$./velvetg directorio_salida -cov_cutoff 4 -min_contig_lgth 100
```

Esto significa que se puede ejecutar `velvetg` de manera repetida sin rehacer la mayoría de los cálculos:

```
$./velvetg directorio_salia -cov_cutoff 4
```

```
$./velvetg directorio_salia -cov_cutoff 3.8
```

```
$./velvetg directorio_salia -cov_cutoff 7
```

```
$./velvetg directorio_salia -cov_cutoff 10
```

```
$./velvetg directorio_salia -cov_cutoff 2
```

Por otro lado, el orden de los parámetros no es importante dentro de un único comando `velvetg`.

Finalmente, si se tienen dudas en la línea de comandos, basta con ejecutar el programa sin argumentos para obtener un mensaje de ayuda.

6.5.3.1.1. Lecturas Individuales

Inicialmente, si se ejecuta:

```
$./velvetg directorio_salida
```

Producirá un fichero *fasta* de *contigs* y un conjunto de estadísticas. La experiencia demuestra que hay muchos nodos cortos y con baja cobertura, eliminados de la corrección inicial, por lo que para determinar un valor de corte de la cobertura de 5.2x se ejecuta el siguiente comando:

```
$./velvetg directorio_salida -cov_cutoff 5.2
```

Por otro lado, si se desea excluir del ensamblaje los datos con una cobertura muy alta (e.g. secuencias de plásmidos, mitocondrias y cloroplastos) se puede utilizar un corte de cobertura máxima:

```
$./velvetg directorio_salida -max_coverage 300 (… otros parámetros …)
```

Aunque el corte de cobertura se puede ir optimizando en sucesivas iteraciones, también se puede poner de manera automática:

```
$./velvetg directorio_salida -cov_cutoff auto
```

6.5.3.1.2. Adición de Lecturas Largas

Si se tienen una cobertura suficiente de lecturas cortas y una cierta cantidad de lecturas largas, se pueden usan dichas coberturas largas para resolver repeticiones mediante un algoritmo voraz (es necesario haber utilizado `velveth` para lecturas largas).

Para realizar esto, **Velvet** necesita tener una estimación razonable de la cobertura de secuencias únicas en lecturas cortas y la forma más simple de obtener este valor es observar la distribución de la cobertura de los *contigs* y observar valores cercanos con los que los nodos se agrupan (especialmente los nodos grandes del conjunto de datos). Por ejemplo, suponiendo que la cobertura esperada es 19x, entonces se indica de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov 19 (… otros parámetros …)
```

Si se tienen razones para creer que la cobertura es razonablemente uniforme sobre la muestra entonces se le puede indicar a **Velvet** que la estime de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov auto (… otros parámetros …)
```

6.5.3.1.3. Lecturas Emparejadas

Para activar el uso de lecturas emparejadas se deben especificar dos parámetros (además de haber utilizado previamente `velveth` para lecturas emparejadas): la longitud esperada (media) del inserto (o al menos, una estimación) y la cobertura esperada de los *K-mer* de lecturas cortas. La longitud del inserto se entiende como la longitud de los fragmentos secuenciados. De esta manera, por ejemplo, si se estima una longitud del inserto de alrededor de 400pb y una cobertura de alrededor de 21.3x entonces se puede escribir:

```
$./velvetg directorio_salida –ins_length 400 –exp_cov 21.3 (… otros parámetros …)
```

Si se tienen lecturas largas emparejadas y están correctamente ordenadas se puede utilizar **Velvet** para la generación de *supercontigs* indicando la longitud correspondiente del inserto de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov 21 –ins_length_log 40000 (… otros parámetros …)
```

Si la cobertura es razonablemente uniforme se puede instar a **Velvet** para que la calcule de forma automática de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov auto (… otros parámetros …)
```

Velvet puede estimar la longitud de un inserto si no se ha especificado en una librería dada. Lo que hace es ajustar de manera automática los parámetros `ins_length*` y `ins_length*_sd`.

Por otro lado, **Velvet** intenta ensamblar *contigs* aunque no estén correctamente conectados por lo que se generan secuencias de caracteres N en el fichero *`contigs`*`.fa`, que se corresponden a la distancia estimada entre dos *contigs* vecinos. Si no se desea este ensamblaje es posible modificarlo de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov 21 –ins_length_long 200 –scaffolding no (… otros parámetros …)
```

Con respecto a las desviaciones estándar, **Velvet** no utiliza valores absolutos para las longitudes de los insertos, sino sus valores relativos por lo que no es necesario realizar las estimaciones de las desviaciones estándar para que sean consistentes. Es posible introducir los valores que tenemos de la siguiente manera:

```
$./velvetg directorio_salida –exp_cov 21 –ins_length 200 –ins_length_sd 20 –ins_length2 20000 –ins_length2_sd 5000 –ins_length_long 40000 –ins_length_long_sd 1000 (… otros parámetros …)
```

6.5.3.1.4. Salida de **Velvet**

6.5.3.1.4.1 Selección de los *Contigs*

Por defecto, **Velvet** imprime todos los *contigs* que sea posible. Esto tiene el inconveniente de saturar la salida con *contigs* excesivamente cortos que pueden ser inutilizables. Es posible indicar que la longitud de los *contigs* generados sea mayor que una determinada longitud, de la siguiente manera:

```
$./velvetg –min_contig_lgth 1000 (… otros parámetros …)
```

6.5.3.1.4.2 Seguimiento de Lecturas

Aunque el seguimiento de las lecturas consume más memoria y tiempo de computación, tiene la ventaja de que produce una descripción más detallada del ensamblaje. Se hace de la siguiente manera:

```
$./velvetg directorio_salida –read_trkg yes (… otros parámetros …)
```

6.5.3.1.4.3 Generación de un fichero .afg

Para generar ficheros **AMOS** que muestran la información del ensamblaje en una estructura de datos se hace lo siguiente:

```
$./velvetg directorio –AMOS_file yes (… otros parámetros …)
```

6.5.3.1.4.4 Uso de Múltiples Categorías

Es posible mantener separados varios tipos de conjuntos de lecturas cortas (que previamente se han organizado mediante las opciones apropiadas de `velveth`) de la siguiente manera:

```
$./velvetg directorio_salida –ins_length 400 –ins_length2 10000 (… otros parámetros …)
```

El ejemplo anterior pone los insertos más cortos en la primera categoría, supone que los primeros

conjuntos de lecturas tienen una longitud aproximada de 400pb y el segundo de 10.000pb.

6.5.3.1.4.5 Obtención de las Lecturas no Utilizadas en el Ensamblaje

Se genera un fichero FASTA denominado `UnusedReads.fa` de la siguiente manera:

```
$./velvetg directorio_salida –unused_reads yes (… otros parámetros …)
```

6.5.3.2. PARÁMETROS AVANZADOS

- Longitud Máxima de Rama: Por defecto está puesta a 100pb, pero se puede modificar de la siguiente manera:

```
$./velvetg directorio_salida –max_branch_length 200
```

- Ratio de Divergencia Máximo: por defecto **Velvet** no simplifica dos secuencias que se han alineado pero que son divergentes en más de un 20%. Si se desea modificar este límite, se puede hacer de la siguiente manera:

```
$./velvetg directorio_salida –max_divergence 0.33
```

- Número Máximo de Huecos: después de alinear dos secuencias con un alineamiento dinámico estándar, **Velvet** compara el número de pares de nucleótidos alineados con la longitud de la secuencia más larga. Por defecto, **Velvet** no simplifica las secuencias si más de 3pb de la secuencia más larga, no se han alineado. Para modificar esto, se hace de la siguiente manera:

```
$./velvetg directorio_salida –max_gap_count 5
```

6.5.3.3. PARÁMETROS AVANZADOS: rock band

6.5.3.3.1. Corte Mínimo para la Conexión de Lecturas Largas

Cuando se utilizan lecturas largas para conectar y completar *contigs*, **Velvet** aplica un corte multiplicativo simple (por defecto 2) para reducir la cantidad de ruido. Dicho de otra manera, se necesitan al menos dos lecturas largas para validar una conexión. No obstante, puede ser necesario bajar este corte a 0 en los casos en los que se tengan secuencias no solapadas pero confiables. Y por el contrario, a veces puede ser necesario incrementar este corte si la cobertura de lecturas largas es muy alta.

6.5.3.4. PARÁMETROS AVANZADOS: pebble

6.5.3.4.1. Validación Mínima del Par de Lectura

Velvet asume por defecto que las lecturas emparejadas están perfectamente situadas y requiere que una conexión entre dos *contigs* sea corroborada por, al menos, 10 lecturas emparejadas. Si se desea cambiar este valor, se hace de la siguiente manera:

```
$./velvetg directorio_salida –min_pair_count 20
```

6.5.4. FORMATOS DE FICHEROS

6.5.4.1. ficheros de secuencias de entrada

Velvet trabaja principalmente con formato *fasta* y *fastq*. Para lecturas emparejadas, se asume que cada lectura es seguida por su emparejada. En otras palabras, si las lecturas están indexadas desde 0, entonces las lecturas 0 y 1 están emparejadas.

Si por alguna razón se tienen lecturas directas e inversas en dos ficheros FASTA diferentes pero con su orden correspondiente, existe un *script* en Perl (`shuffleSecuencies_fasta.pl`) que une los dos ficheros en uno solo, de la siguiente manera:

```
$./shuffleSequences_fasta.pl forward_reads.fa reverse_reads.fa output.fa
```

Existe el correspondiente script `shuffleSequences_fastq.pl` para los ficheros FastQ.

6.5.4.2. Ficheros de salida

El ejecutable `velvetg` genera los siguientes ficheros:

- *`Contigs`*`.fa`: contiene las secuencias de los *contigs* mayores de 2k, donde *k* es la longitud de palabra usada en `velveth`. Si se ha utilizado un umbral `min_contig_lgth` entonces se omiten los *contigs* que sean más cortos que este valor.

  Hay que tener en cuenta que la información de longitud y cobertura proporcionada en la cabecera de cada contig debe ser interpretada en *K-mer* y cobertura de *K-mer*, respectivamente

  Las secuencias de *N* corresponden a huecos entre *supercontigs.* El número de *N* corresponde a la longitud estimada del hueco. Por razones de compatibilidad con los ficheros ningún hueco más corto de 10pb se representa por una secuencia de 10 N.

- `Stats.txt`: Es un fichero delimitado por tabuladores que describe los nodos. La longitud de los nodos se dan en *K-mer*. Para obtener la longitud en nucleótidos de cada nodo se necesita simplemente sumar *k-1*, donde *k* es la longitud de palabra utilizada en `velveth`.

Las columnas in y out se corresponden con el número de arcos de los extremos 5' y 3' de los *contigs*.

Las coberturas de las columnas `short1_cov`, `short1_0cov`, `short2_cov` y `short2_0cov` se dan en cobertura de *K-mer*.

- **`Velvet`**`_asm.afg`: Este fichero está diseñado para que el paquete de ensamblaje **AMOS** lo pueda leer. El fichero describe todos los *contigs* contenidos en el fichero *`contigs`*`.fa`.
- `LastGraph`: Este fichero describe el grafo generado por **Velvet** de una manera particular, sin formato estándar, de la siguiente manera:
  - Una línea de cabecera para el grafo:

    $NUMER_OF_NODOS $NUMBER_OF_SEQUENCES $*HASH*_LENGTH

  - Un bloque por cada nodo:

    NODE $NOD_ID $COV_SHORT1 $0_COV_SHORT1 $COV_SHORT2 $0_COV_SHORT2 $ENDS_OF_KMERS_OF_NODE $ENDS_OF_KMERS_OF_TWIN_NODE

  - Una línea por cada arco:

    ARC $START_NODE $END_NODE $MULTIPLICITY

  - Un bloque conteniendo el camino de cada secuencia larga

    NR $NODE_ID $NUMBER_OF_SHORT_READS $READ_ID $OFFSET_FROM_START_OF_NODE $START_COORD $READ_ID2 etc.

6.5.5. PRUEBA DE EJECUCIÓN DE **VELVET**

Realizamos una prueba básica de **Velvet** con 1 fichero (`reads.fasta`) de 142.858 lecturas cortas de 35pb.

```
mmhr@ubuntu:~/Ensambladores/Velvet_1.0.19$ ./velveth ../../Datos/Resultados/Velvet/prueba-corta 21 -short ../../Datos/Datasets/EULER-SR/reads.fasta

[0.000001] Reading FastA file ../../Datos/Datasets/EULER-SR/reads.fasta;

[0.256637] 142858 sequences found

[0.256653] Done

[0.256719] Reading read set file ../../Datos/Resultados/Velvet/prueba-corta/Sequences;

[0.283461] 142858 sequences found

[0.542089] Done

[0.542113] 142858 sequences in total.

[0.542196] Writing into roadmap file ../../Datos/Resultados/Velvet/prueba-corta/Roadmaps...

[0.578163] Inputting sequences...

[0.578188] Inputting sequence 0 / 142858

[1.372468] Inputting sequence 100000 / 142858

[1.679134] Done inputting sequences

[1.679151] Destroying splay table

[1.699918] Splay table destroyed

mmhr@ubuntu:~/Ensambladores/Velvet_1.0.19$
```

Se generan tres ficheros que son los que trata `velvetg`

```
mmhr@ubuntu:~/Datos/Resultados/Velvet/prueba-corta$ ll

total 15260

-rw-r--r-- 1 mmhr mmhr 131 2011-04-05 00:51 Log

-rw-r--r-- 1 mmhr mmhr 5828855 2011-04-05 00:51 Roadmaps

-rw-r--r-- 1 mmhr mmhr 9788539 2011-04-05 00:51 Sequences

mmhr@ubuntu:~/Datos/Resultados/Velvet/prueba-corta$
```

Ejecutamos `velvetg`

```
mmhr@ubuntu:~/Ensambladores/Velvet_1.0.19$ ./velvetg ../../Datos/Resultados/Velvet/prueba-corta/ -cov_cutoff auto

[0.000000] Reading roadmap file ../../Datos/Resultados/Velvet/prueba-corta//Roadmaps

[0.248801] 142858 roadmaps reads

[0.248872] Creating insertion markers

[0.277613] Ordering insertion markers

[0.432207] Counting preNodes

[0.456942] 124592 preNodes counted, creating them now
```

[0.649361] Sequence 100000 / 142858

[0.709600] Adjusting marker info...

[0.735518] Connecting preNodes

[0.819664] Connecting 100000 / 142858

[0.860556] Cleaning up memory

[0.861169] Done creating preGraph

[0.861178] Concatenation...

[0.934792] Renumbering preNodes

[0.934824] Initial preNode count 124592

[0.944552] Destroyed 70414 preNodes

[0.944581] Concatenation over!

[0.944584] Clipping short tips off preGraph

[0.970041] Concatenation...

[0.988350] Renumbering preNodes

[0.988376] Initial preNode count 54178

[0.988948] Destroyed 52893 preNodes

[0.988955] Concatenation over!

[0.988958] 28938 tips cut off

[0.988961] 1285 nodes left

[0.989050] Writing into pregraph file ../../Datos/Resultados/**Velvet**/prueba-corta//PreGraph...

[1.009551] Reading read set file ../../Datos/Resultados/**Velvet**/prueba-corta//Sequences;

[1.036258] 142858 sequences found

[1.280579] Done

[1.324558] Reading pre-graph file ../../Datos/Resultados/**Velvet**/prueba-corta//PreGraph

[1.324651] Graph has 1285 nodes and 142858 sequences

[1.330916] Scanning pre-graph file ../../Datos/Resultados/**Velvet**/prueba-corta//PreGraph for k-mers

[1.332701] 105905 kmers found

[1.340270] Sorting kmer occurence table ...

[1.381584] Sorting done.

[1.436873] Threading through reads 0 / 142858

[1.897598] Threading through reads 100000 / 142858

[2.108675] Correcting graph with cutoff 0.200000

[2.108853] Determining eligible starting points

[2.110223] Done listing starting nodes

[2.110228] Initializing todo lists

[2.110361] Done with initilization

[2.110365] Activating arc lookup table

[2.110501] Done activating arc lookup table

[2.123376] 1000 nodes visited

[2.126531] Concatenation...

[2.126576] Renumbering nodes

[2.126579] Initial node count 1285

[2.126595] Removed 889 null nodes

[2.126599] Concatenation over!

[2.126601] Clipping short tips off graph, drastic

[2.126662] Concatenation...

[2.127996] Renumbering nodes

[2.128007] Initial node count 396

[2.128014] Removed 204 null nodes

[2.128017] Concatenation over!

[2.128020] 192 nodes left

[2.128112] Writing into graph file ../../Datos/Resultados/**Velvet**/prueba-corta//Graph...

[2.145426] Measuring median coverage depth...

[2.145511] Median coverage depth = 14.504422

[2.145547] Removing *contigs* with coverage < 7.252211...

[2.145575] Concatenation...

[2.146178] Renumbering nodes

[2.146183] Initial node count 192

[2.146188] Removed 117 null nodes

[2.146191] Concatenation over!

[2.146197] Concatenation...

[2.146202] Renumbering nodes

[2.146205] Initial node count 75

[2.146208] Removed 0 null nodes

[2.146210] Concatenation over!

[2.146214] Clipping short tips off graph, drastic

[2.146220] Concatenation...

[2.146225] Renumbering nodes

[2.146227] Initial node count 75

[2.146231] Removed 1 null nodes

[2.146233] Concatenation over!

[2.146236] 74 nodes left

[2.146238] WARNING: NO EXPECTED COVERAGE PROVIDED

[2.146241] **Velvet** will be unable to resolve any repeats

[2.146244] See manual for instructions on how to set the expected coverage parameter

[2.146247] Concatenation...

[2.146252] Renumbering nodes

[2.146255] Initial node count 74

[2.146258] Removed 0 null nodes

[2.146260] Concatenation over!

[2.146263] Removing reference *contigs* with coverage < 7.252211...

[2.146271] Concatenation...

[2.146276] Renumbering nodes

[2.146279] Initial node count 74

[2.146282] Removed 0 null nodes

[2.146284] Concatenation over!

[2.146330] Writing *contigs* into ../../Datos/Resultados/**Velvet**/prueba-corta//*contigs*.fa...

[2.158418] Writing into stats file ../../Datos/Resultados/**Velvet**/prueba-corta//stats.txt...

[2.158814] Writing into graph file ../../Datos/Resultados/**Velvet**/prueba-corta//LastGraph...

[2.175324] Estimated Coverage cutoff = 7.252211

Final graph has 74 nodes and n50 of 14327, max 30503, total 97802, using 0/142858 reads

7. ANÁLISIS BÁSICO DE CÓDIGO

Hemos realizado un análisis básico del código de los cinco programas evaluados. Tres de ellos (**ABYSS**, **ALLPATHS** y **EULER-SR**) están desarrollados en C++ y los otros dos (**SOAP*DENOVO*** y **Velvet**) lo están en el lenguaje de programación C. Todo ellos, a su vez, tienen diversos *scripts* de ejecución tanto en Perl como en Shell de Unix.

Para evaluar ciertas métricas hemos utilizado dos herramientas concretas:

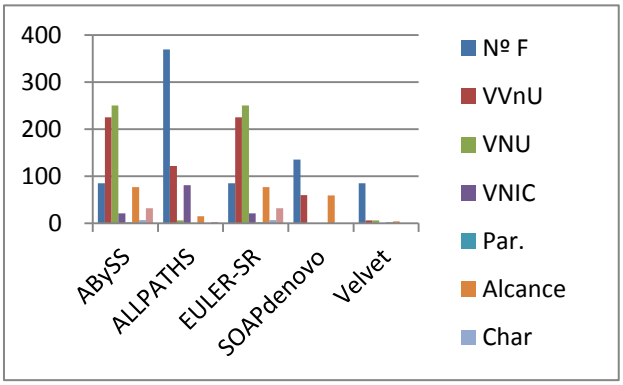
- **Cppcheck** [24] que es una herramienta para análisis estático de cógido en C/C++.
- **Sonar** [134] que es una herramienta más completa pero que no dispone de un plugin estable para C++ con lo que sólo podemos evaluar los dos programas desarrollados en C.

7.1. CPPCHECK

Cppcheck es un programa que se ejecuta en línea de comandos.

Tabla 3 Análisis estático de código de los cinco programas mediante cppcheck para determinar métricas básicas: (Nº F) Número de Ficheros, (VVnU) Variables con valores asignados que nunca se utilizan, (VNU) Variables no utilizadas, (VNIC) Variables no Inicializadas en su Constructor, (Par.) Parámetros en funciones pasados por valor cuando se debería pasar por referencia, (Alcance) Variables cuyo alcance puede ser reducido, (Char) Variables `char` definidias como `arrays` y (Funciones) Funciones que pueden devolver una contante.

cppcheck	Nº F	VVnU	VNU	VNIC	Par.	Alcance	Char	Funciones
ABySS	85	225	250	21	2	77	7	32
ALLPATHS	369	122	6	81	1	15	0	3
EULER-SR	85	225	250	21	2	77	7	32
SOAPdenovo	135	60	0	0	0	59	0	0
Velvet	85	6	6	0	2	4	0	0



El comando ejecutado para generar los datos de `cppcheck` es el siguiente:

```
$cppcheck -j 4 directorio_ensamblador -enable=all > resultados.out 2> errores.out
```

7.2. SONAR

Una vez realizado el análisis con cppcheck realizamos un análisis con sonar que posee un servidor web integrado que permite una visualización gráfica de los resultados. Para ejecutarlo hemos instalado **maven** [6] y generado un fichero `POM.xml` básico en el directorio de cada programa que nos permite activar dicho programa en sonar mediante el comando `mvn sonar:sonar`.

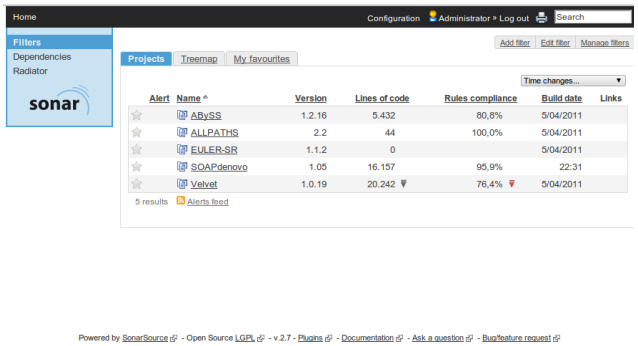


Figura 30: Carga de los cinco programas en Sonar. Debido a que sólo existe un plugin estable para Lenguaje C, solo podemos realizar un análisis básico de **SOAP*DENOVO*** y **Velvet**.

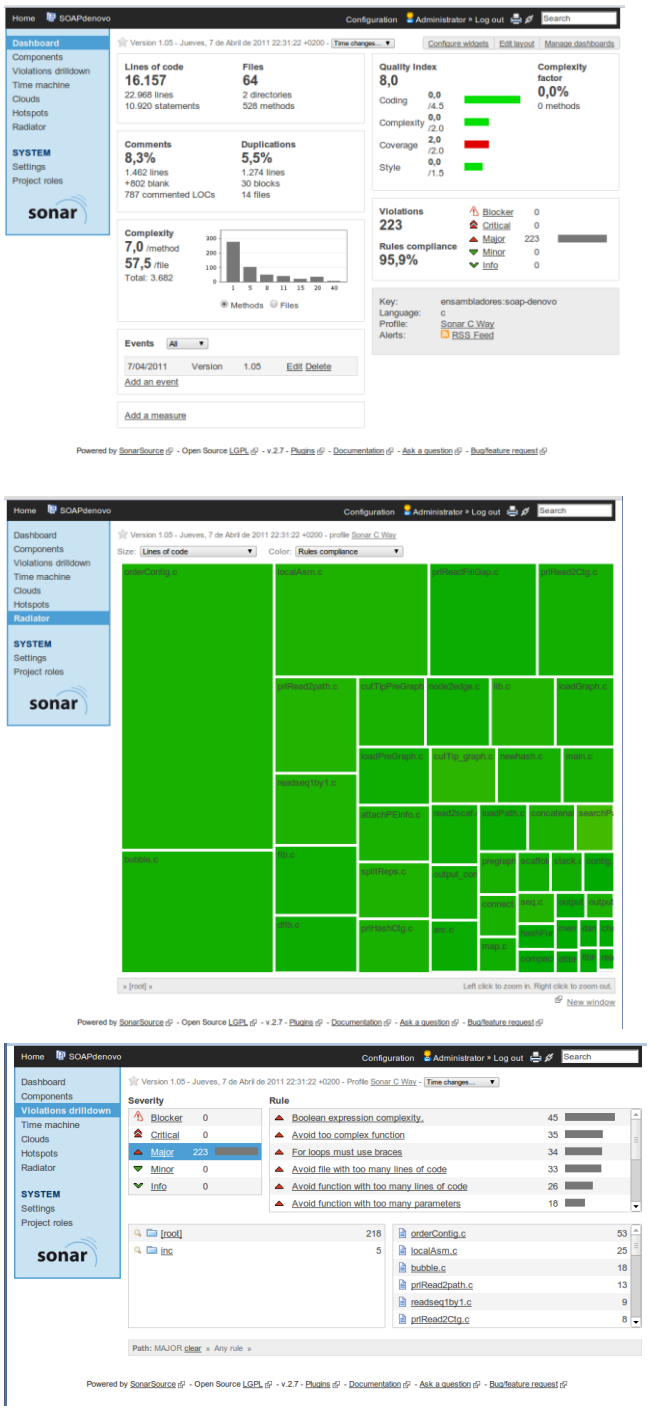


Figura 31: Análisis básico mediante Sonar del código de **SOAP*DENOVO*** mediante el plugin de C. Los datos muestran que existe un 95% de cumplimiento de reglas pero existen violaciones importantes en el código que sería conveniente corregir, como pueden ser las expresiones complejas booleanas y el uso de funciones complejas. El programa más importante es `ordenContig.c`.

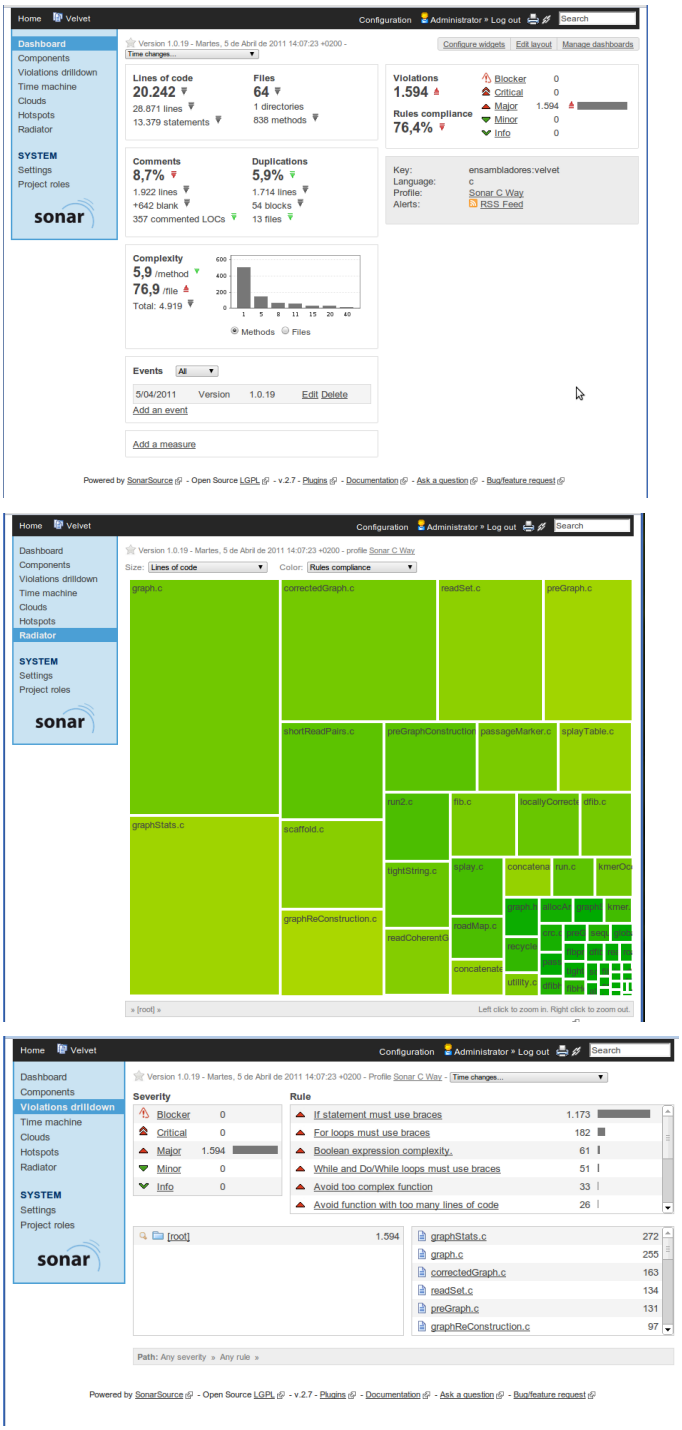


Figura 32: Análisis básico mediante Sonar del código de **Velvet** mediante el plugin de C. Los datos muestran que tiene el mismo número aproximado de ficheros pero más líneas de código. Existe mayor complejidad por fichero pero menor por método. Tiene un número muy importante de violaciones de codificación de sentencias if que pueden ser debidos a hábitos de programación. Existen dos programas importantes: `graph.c y grapStats.c`

7.3. CONCLUSIONES

Los ensambladores evaluados se encuentran en versiones tempranas de su desarrollo ya que el más avanzado es **ALLPATHS** que está en la versión 2.2 y el reston está en versiones 1.X, aunque llevan años desde que comenzaron a desarrollarse. Se trata, pues, de desarrollos software complejos habitualmente ejecutados por un equipo de desarrollo pequeño que dificulta un versionado y avance más rápido de los productos. Adicionalmente, en algunos casos, se presentan problemas de compilación derivados de las versiones tempranas del código que se ven arrastradas en su evolución.

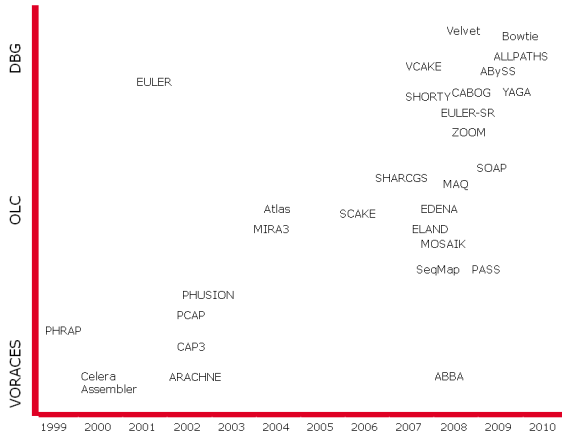


Figura 33: Evolución metodológica de los programas de ensamblaje de secuencias. Los últimos ensambladores tienen en común una orientación hacía la construcción y recorrido de un grafo como reconocimiento del ensamblaje y concretamente el uso extensivo de grafos de Bruijn para su construcción. Todos ellos hacen un uso intensivo de memoria por lo que se hace prácticamente inviable la secuenciación completa.

Podemos considerar **Velvet** como un ensamblador bien construido, con un código bien desarrollado, sin limitación a priori del la longitud de los *K-mer* y con un uso extensivo de los grafos de Bruijn.

Todos ellos están preparados para funcionar en procesamiento paralelo y utilizan MPI. No obsante, todos tienen en común una orientación hacía el uso intensivo de memoria en el modelado de los grafos para intentar proporcionar una alta calidad en los resultados teniendo en cuenta una gran cobertura para compensar la pequeña longitud de las lecturas.

En los últimos años, la mayoría de los desarrollos se han orientado hacía modelos de grafos con *K-mer* de longitud fija pequeña que están siendo evolucionados pero no se observan nuevos modelos disruptivos o nuevas aproximaciones matemáticas teniendo en cuenta tanto el auge de la computación distribuida como la aparición de nuevas tecnologías de secuenciación más allá de NGS que sean capaces de generar mayores longitudes de lecturas.

Estos factores deben influir en los próximos años para el desarrollo de una nueva generación de ensambladores que se orienten a una computación distribuida en la nube con una orientación al modo servicio teniendo en cuenta la posible cooperación entre los distintos nodos de supercomputación y que estén mejor preparados para transcriptómica y para nuevas longitudes de lectura cada vez mayores.

8. REFERENCIAS

1. Aleksey V. Zimin, Douglas R. Smith, Granger Sutton and James A. Yorke. Assembly reconciliation. *December. 2007*. http://bioinformatics.oxfordjournals.org/content/24/1/42.full
2. Alex Bateman and John Quackenbush. Bioinformatics for next generation sequencing. *Bioinformatics 2009;25:429*. http://bioinformatics.oxfordjournals.org/content/25/4/429.full.pdf+html
3. Alexei Papanicolaou, Remo Stierli, Richard H ffrench-Constant and David G Heckel. Next generation transcriptomas for next generation genomes using *est2assembly*. *BMC Bioinformatics. 2009*. http://www.biomedcentral.com/1471-2105/10/447
4. Altshul S. F., Gish W., Myers E. W., Lipman D. J. Basic local alignment search tool. *J Mol Bio. 1990 Oct 5;215(3):403-10*. http://www.ncbi.nlm.nih.gov/pubmed/2231712 http://www.nature.com/scitable/topicpage/basic-local-alignment-search-tool-blast-29096
5. Andrew D. Smith, Zhenyu Xuan and Michael Q Zhang. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics. 2008*. http://www.biomedcentral.com/1471-2105/9/128
6. Apache Maven: http://maven.apache.org/
7. Ariella Sasson and Todd P. Michael. Filtering error from SOLid Output. *Bioinformatics. Vol. 26 no. 6 2010, pages 849–850. doi:10.1093/bioinformatics/btq045.* http://bioinformatics.oxfordjournals.org/content/26/6/849.full.pdf
8. Bastien Bastien Chevreux, Thomas Pfisterer, Bernd Drescher, Albert J. Driesel, Werner E.G. Müller, Thomas Wetter and Sándor Suhai. Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs. http://genome.cshlp.org/content/14/6/1147.full
9. Bastien Chevreux, Thomas Pfisterer, Bernd Drescher, Albert J. Driesel, Werner E.G. Müller, Thomas Wetter and Sándor Suhai. Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs. *Cold Spring Harbor Laboratory Press. December 16, 2008*. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC419793/
10. Bastien Chevreux, Thomas Pfisterer, Bernd Drescher, Albert J. Driesel, Werner E.G. Müller, Thomas Wetter and Sándor Suhai. Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs. *Cold Spring Harbor Laboratory Press. December 16, 2008*
11. Batzoglou, S. (2005) in *Encyclopedia of genomics, proteomics and bioinformatics, Algorithmic challenges in mammalian genome sequence assembly*, ed Dunn, M., et al. (John Wiley and Sons, New York) Part 4.
12. Batzoglou, S., Jaffe, D.B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J.P., Lander, E.S.(2002) **ARACHNE**: A whole genome shotgun assembler. *Genome Res. 12:177-189*. http://genome.cshlp.org/content/12/1/177.full
13. Batzolgou S., Jaffe DB., Stanley K., Butler J., Gnerre S., Mauceli E., Berger B., Mesirov JP., Lander ES., Laboratory for Computer Science, Massachusetss Institue of Technology. Cambridge, Massachusetts 02139, USA. **ARACHNE**: a whole-genome shotgun assembler. 2002. http://genome.cshlp.org/content/12/1/177.long
14. Ben Langmead, Cole Trapnell, Mihai Pop and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology 2009, 10:R25. Doi:10.1186/gb-2009-10-3-r25.* http://genomebiology.com/2009/10/3/R25
15. Benjamin G. Jackson, Patrick S. Schnable and Srinivas Aluru. Parallel short sequence assembly of transcriptomes. *The Seventh Asia Pacific Bioinformatics Conference (APBC 2009) 13-16 January, 2009. Beijing, China*. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2648799/?tool=pubmed
16. Bentley, D.R. (2006) Whole-genome re-sequencing. Curr. Opin. Genet. Dev. 16:545-552. http://www.ncbi.nlm.nih.gov/pubmed/17055251?dopt=Abstract
17. Brent Ewing, LaDeana Hillier, Michael C. Wendl, Phil Green. Base-Calling of Automated Sequencer Traces Using *Phred*. Accuracy Assessment. http://genome.cshlp.org/content/8/3/175.abstract?ijkey=744c33c481d6c258af1b6b289c9ee8ffffca3a3ee&keytype2=tf_ipsecsha
18. Bryant DW, Jr, Wong WK, Mockler TC. QSRA: a quality-value guided de novo*de novo* short read assembler. *BMC Bioinformatics* 2009;10:69. http://www.biomedcentral.com/1471-2105/10/69
19. Burrows, M and Wheeler, D (1994). A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment

Corporation. http://www.cs.ucdavis.edu/~gusfield/spring06readings/BWTOriginal.pdf
20. Carl Kingsford, Michael C. Schatz, Miahi Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinformatics. 2010*. http://www.biomedcentral.com/1471-2105/11/21
21. Código fuente de **BOWTIE**: http://**BOWTIE**-bio.sourceforge.net/index.shtml
22. Código fuente del **CELERA ASSEMBLER**: http://sourceforge.net/apps/mediawiki/wgs-assembler/index.php?title=Main_Page.
23. Cole Trapnel & Steven L. Salzberg. How to map billions of short reads onto genomes. *Nature Biotechnol 2009;27:455-7* http://www.cbcb.umd.edu/publications/files/ShortReadMappingPrimer-reprint.pdf
24. Cppcheck: http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main_Page
25. D. Fasulo, A. Halpem, I. Dew, C. Mobarry. Efficiently detecting polymorphisms during the fragment assembly process. Bioinformatics 18 (Suppl 1) (2002) 5294-5302. http://www.ncbi.nlm.nih.gov/pubmed/12169559
26. D.R. Zerbino, G.K. McEwen, E.H. Margulies, E. Birney. Pebble and rock band: heuristic resolution of repeats and scaffolding in the **Velvet** short-read de novo*de novo* assembler. PLoS One 4 (2009) e8407. http://www.plosone.org/article/info:doi/10.1371/journal.pone.0008407
27. Daniel Branton, David W. Deamer, Andre Marciali et al. The potential and challenges of nanopore sequencing. *Nat Biothecmol 2008:26:1146-53*. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2683588/?tool=pubmed
28. Daniel J. Turner, Thomas M. Keane, Ian Sudbery and David J. Adams. Next-generation sequencing of vertebrate experimental organisms. Mamm Genome. 2009. http://en.scientificcommons.org/48353445
29. Daniel R. Zerbino and Ewan Birney. **Velvet**: Algorithms for de novo*de novo* short read assembly usion de Bruijn graphs. http://www.ncbi.nlm.nih.gov/pubmed/18349386
30. David Camapgna, Alesandro Albiero, Alessandra Bilardi, Elisa Caniato, Claudio Forcato, Svetlin Manavski, Nicola Vitulo and Giorgio Valle. **PASS**: A program to Align Short Sequences. http://bioinformatics.oxfordjournals.org/content/25/7/967.long
31. David Hernandez, Patrice Francois, Laurent Farinelli, Mange Osteràs and Jacques Schrenzel. De novo*De novo* bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Res 2008. 10.2008. doi:10.1101/gr.072033.107*. http://genome.cshlp.org/content/early/2008/04/03/gr.072033.107
32. David R. Bentley et al. Accurate Whole Human Genome Sequencing using Reversible Terminator Chemistry. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2581791/
33. David Stephen Horner, Giulio Pavesi, Tiziana Castrignano, Paolo D'Onorio, Sabino Liuni, Michael Sammeth, Ernesto Picardi and Graziano Pésole. Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Briefings in Bioinformatics. Vol. II, no. 2, 181-197. October 2009*. http://bib.oxfordjournals.org/content/11/2/181.full.pdf
34. De Bona F, Ossowski S, Schneeberger K, et al. Optimal spliced alignments of short sequence reads. Bioinformatics 2008;24:i174-80. http://bioinformatics.oxfordjournals.org/content/24/16/i174.full.pdf+html
35. Documentación de Mapreads v2.4.1. http://solidsoftwaretools.com/download/docmanfileversion/Mapreads_Documentation_v2.4.1.pdf
36. Dohm JC, Lottaz C, Borodina T, Himmelbauer H. **SHARCGS**, a fast and highly accurate short-read assembly algorithm for de novo*de novo* genomic sequencing. Genome Res. 2007 Nov; 17(11):1697-706. Epub 2007 Oct. 1. http://www.ncbi.nlm.nih.gov/pubmed/17908823
37. E.W. Myers, G.G. Sutton, A.L. Delcher, I.M. Dew, D.P. Fasulo, M.J. Flanigan, S.A. Kravitz, C.M. Mobarry, K.H. Reinert, K.A. Remington, E.L. Anson, R.A. Bolanos, H.H. Chou, C.M. Jordan, A.L.Halpern, S. Lonardi, E.M. Beasley, R.C. Brandon, L. Chen, P.J. Dunn, Z. Lai, Y. Liang, D.R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G.M. Rubin, M.D. Adams, J.C. Venter, A whole-genome assembly of Drosophila, Science 287 (2000) 2196–2204. http://www.ncbi.nlm.nih.gov/pubmed/10731133
38. E.W. Myers. Toward simpliflying and accurately formulating fragment assembly. J. Comput. Biol. 2 (1995) 275-290. http://www.ncbi.nlm.nih.gov/pubmed/7497129
39. Eaves HL., Gao Y. MOM: maximum oligonucleotide mapping. Bioinformatics. 2009.

http://www.ncbi.nlm.nih.gov/pubmed/19228804

40. Enlace a **AMOS**: http://www.cbcb.umd.edu/research/assembly.shtml
41. Geo Pertea, Xiaoqiu Huang, Feng Liang, Valentin Antonescu, Razvan Sultana, Svetlana Karamycheva, Yuandan Lee, Joseph White, Foo Cheung, Babak Parvizi, Jennifer Tsai and John Quackenbush. **TIGR** Genes Indices clustering tools (TGICL): a software system for fast clustering of large EST datasets. *Bioinformatics. Vol 19. no. 5 2003, pages 651-652* http://bioinformatics.oxfordjournals.org/content/19/5/651.full.pdf+html
42. Goldberg SM, Johnson J, Busam D, Feldblyum T, Ferriera S, Friedman R, Halpern A, Khouri H, Kravitz SA, Lauro FM, Li K, Rogers YH, Strausberg R, Sutton G, Tallon L, Thomas T, Venter E, Frazier M, Venter JC. A Sanger/pyrosequencing hybrid approach for the generation of high-quality draft assemblies of marine microbial genomes. Proc. Natl. Acad. Sci. USA. 2006 Jul 25;103(30):11240-5. Epub 2006 Jul 13. http://www.ncbi.nlm.nih.gov/pubmed/16840556
43. Gupta PK. Single-molecule DNA sequencing technologies for future genomics research. *Trends Biotechmol 2008:26:602-11*. http://www.ncbi.nlm.nih.gov/pubmed/18722683
44. Haeyoung Jeong and Jihyun F. Kim. An optimized strategy for geneme assembly of Sanger/Pyrosequencing hybrid data using available software. *Genomics & Informatics. Vol. 6(2) 87-90, June 2008*. http://www.genominfo.org/html/UploadFile/article7_200806.pdf
45. Havlak, P., Chen, R., Durbin, J., Egan., Ren, Y., Song, X.-Z., Weinstock, G.M., Gibbs, R.A. (2004) The **ATLAS** genome assembly system. *Genome Res. 14:721-732*. http://genome.cshlp.org/content/14/4/721.full
46. Heng Li, Richard Durbin. Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform. The Wellcome Trust Sanger Institute. Bioinformatics Advance Access published May 18, 2009. http://bioinformatics.oxfordjournals.org/content/early/2009/05/18/bioinformatics.btp324.full.pdf
47. http:// sourceforge.net/apps/mediawiki/mira-assembler/index.php?title=Main_Page
48. http://454.com/products-solutions/analysis-tools/gs-de-novo-assembler.asp
49. http://bioinformatics.bc.edu/marthlab/**MOSAIK**
50. http://code.google.com/p/**MOSAIK**-aligner/
51. http://**PASS**.cribi.unipd.it/cgi-bin/**PASS**.pl
52. http://soap.genomics.org.cn/**SOAP*DENOVO***.html
53. http://www.encuentros.uma.es/encuentros128/comofunciona128.pdf
54. http://www.ncbi.nlm.nih.gov/pubmed/20659677?dopt=Abstract
55. http://www.**PHRAP**.org/phred**PHRAP**consed.html
56. Hui Jiang, Wing Hung Wong. **SEQMAP**: mapping massive amount of oligonucleotides to the genome. Stanford University. California 94305. Usa. July, 2008. http://bioinformatics.oxfordjournals.org/content/24/20/2395.full.pdf+html
57. Iain MacChallum, Dariusz Przybylski, Sante Gnerre, Joshua Burton, Ilya Shlyakhter, Andreas Gnirke, Joel Malek, Kevin McKernan, Swati Ranade, Terrance P. Shea, Louise Willians, Sarah Young, Chad Nusbaum and David B. Jaffe. **ALLPATHS2**: small genomes assembled accurately and with high continuity short paired reads. *Open Access. 2009*. http://genomebiology.com/content/10/10/R103
58. Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. J. Comput Biol. 1995 Summer;2(2):291-306. http://www.ncbi.nlm.nih.gov/pubmed/7497130
59. Información sobe GenomeMapper: http://1001genomes.org/downloads/genomemapper.html
60. Información sobe **SLIDER**: http://www.bcgsc.ca/platform/bioinfo/software/slider
61. Información sobre **AMOS**: http://www.cbcb.umd.edu/research/assembly.sht
62. Información sobre **BOWTIE**: http://**BOWTIE**-bio.sourceforge.net/index.shtml
63. Información sobre **BWA**: http://www.bioperl.org/wiki/HOWTO:Short-read_assemblies_with_**BWA**
64. Información sobre **EDENA** se puede encontrar en: http://www.genomic.ch/**EDENA**.php
65. Información sobre **gnumap**: http://dna.cs.byu.edu/gnumap/
66. Información sobre **LaserGene**: http://www.dnastar.com/t-products-lasergene.aspx
67. Información sobre **MAQ**: http://**MAQ**.sourceforge.net/
68. Información sobre **MIRA**: http://www.chevreux.org/projects_mira.html
69. Información sobre MOM: http://mom.csbc.vcu.edu/
70. Información sobre **NOVOCRAFT**: http://www.novocraft.com/main/index.php

71. Información sobre **RMAP**: http://rulai.cshl.edu/rmap/
72. Información sobre **SHORTY**: http://www.cs.sunysb.edu/~skiena/**SHORTY**/
73. Información sobre **SHRIMP**: http://compbio.cs.toronto.edu/SHRIMP/
74. Información sobre **SOAP*DENOVO***: http://soap.genomics.org.cn/**SOAP*DENOVO***.html
75. Información y software de CloudBurst: http://sourceforge.net/apps/mediawiki/CloudBurst-bio/index.php?title=CloudBurst
76. J. Butler, I. MacCallum, M. Kleber, I.A. Ahlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, D.B. Jaffe. **ALLPATHS**: de novo*de novo* assembly of whole-genome shotgun microreads. Genome Res. 18 (2008) 810-820. http://genome.cshlp.org/content/18/5/810.long
77. J. Craig Venter et al. Environmental Genome Shotgun Sequencing of the Sargasso Sea. *Science 2 April 2004. Vol. 304 no. 566 pp. 66-74*. http://www.sciencemag.org/content/304/5667/66.abstract?ijkey=150e9d820352ca291ccc8e198b5304b26ce92d89&keytype2=tf_ipsecsha
78. J.C. Dohm, C. Lortaz, T. Borodina. H. Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. Nucleic Acids Res. 36 (2008) e105. http://nar.oxfordjournals.org/content/36/16/e105.full
79. J.C. Venter, M.D. Adams, E.W. Myers, P.W. Li, R.J. Mural, G.G. Sutton, H.O. Smith, M. Yandell, C.A. Evans, R.A. Holt, J.D. Gocayne, P. Amanatides, R.M. Ballew, D.H. et al. The sequence of the human genome. Science 291 (2001) 1404-1351
80. Jason R. Miller, Arthur L. Delcher, Sergey Koren, Eli Venter, Brian P. Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry and Granger Sutton. Aggresive asembly of pyrosequencing reads with mates. *October 2008*. http://bioinformatics.oxfordjournals.org/content/24/24/2818.abstract
81. Jason R. Miller, Sergey Koren, Granger Sutton. Assembly algorithms for next-generation sequencing data. J. Craig Venter Institute. Genomics 95 (2010) 315-327
82. Jason R. Miller, Sergey Koren, Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics 95 (2010) 315-327*. http://www.cromatina.icb.ufmg.br/seminar/2.assembly.pdf
83. Jonathan M. Rothberg and John H. Leamon. The development and impact of 454 sequencing. *Nature Biotechnology. 2008*. http://www.dkfz.de/gpcf/fileadmin/454/454sequencing.pdf
84. Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina and Heinz Himmelbauer. **SHARCGS**, a fast and highly accurate short-read assembly algorithm for de novo*de novo* genomic sequencing. http://genome.cshlp.org/content/17/11/1697.abstract
85. Keving Judd McKernan, Heather E. Peckham, et al. Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. Genome Res. 2009 September, 19(9): 1527-1541. Doi: 10.1101/gr.091868.109. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2752135/?tool=pmcentrez
86. Lance E. Palmer, Mathaeus Dejori, Randall Bolanos, Daniel Fasulo. Improving de novo*de novo* sequence assembly using machine learning and comparative genomics for overlap correction. http://www.biomedcentral.com/1471-2105/11/33
87. Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwing J, Devon K, Dewar K, Doyle M, FitzHugh W, et al. (2001) Initial sequencing and analysis of the human genome. *Nature 409:860-921*. http://www.nature.com/nature/journal/v409/n6822/full/409860a0.html
88. Lander ES, Waterman MS. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics, 1988 apr;2(3):231-9.* http://www.ncbi.nlm.nih.gov/pubmed/3294162?dopt=AbstractPlus&holding=f1000,f1000m,isrctn
89. Lin H., Zhang Z., Zhan MQ, Ma B., Li M. **YAGA**! Zillions of Oligos Mapped. Institute for Computing Technology. Chinese Academy of Sciences. Beijing. China. http://www.ncbi.nlm.nih.gov/pubmed/18684737
90. M. Chaisson, P.A. Pevzner. H. Tang. Fragment assembly with short reads. Bioinformatics 20 (2004). 2067-2074. http://bioinformatics.oxfordjournals.org/content/20/13/2067.short
91. Margulles, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S., Bemben, L.A., Berka, J-, Braverman, M.S., Chen, Y.-J., Chen, Z., et al. (2005) Genome sequencing in microfabricated high- density picolitre reactors. *Nature 437:376-380*. http://www.ncbi.nlm.nih.gov/pubmed/16056220?dopt=Abstract

92. Mark Chaisson, Pavel Pevzner, Haixu Tang. Fragment assembly with shorts reads. *March 2004*. http://bioinformatics.oxfordjournals.org/content/early/2004/04/01/bioinformatics.bth205.full.pdf
93. Mark J. Chaisson and Pavel A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res. 2007 18:000. Doi:10.1101/gr.7088808*. http://genome.cshlp.org/content/early/2007/12/01/gr.7088808.full.pdf+html
94. Mark JP Chaisson, Dumitru Brinza and Pavel A Pevzner. De novo*De novo* fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res. 2008. Gr.079053.108.* http://genome.cshlp.org/content/early/2008/12/03/gr.079053.108.abstract
95. Metzker, M.L. (2005) Emerging technologies in DNA sequencing. *Genome Res. 15:1767-1776*. http://genome.cshlp.org/content/15/12/1767.full
96. Michael Schatz. Center for Bioinformatics and Computational Biology. University of Maryland. CloudBurst. https://ngslib.i-med.ac.at/node/79
97. Mihai Pop and Steven L. Salzberg. Bioinformatics challenges of new sequencing technology. *Cell Press. TIGS-622. 2007*. http://www.cell.com/trends/genetics/abstract/S0168-9525(08)00022-X
98. Mihai Pop, SL. Salzberg. Bioinformatics challenges of new sequencing technology. Trends Genet. 2008 Mar;24(3):142-9. Epub 2008 Feb 11. http://www.ncbi.nlm.nih.gov/pubmed/18262676
99. Mihai Pop. Genome assembly reborn: recent computational challenges. Brief Bioinform (2009) 10 (4):354-366.doi: 10.1093/bib/bbp026. http://bib.oxfordjournals.org/content/10/4/354.abstract
100. **MIRA**: An Automated Genome and EST Assembler. http://www.chevreux.org/uploads/media/chevreux_thesis_**MIRA**.pdf
101. Mohammad Sajjad Hossaing, Navid Azimi, Steven Skiena. Crystallizing short-read assemblies around seeds. BMC Bioinformatics 2009, 10 (Suppl 1):S16 doi:10.1186/1471-2105-10-S1-S16. http://www.biomedcentral.com/1471-2105/10/S1/S16
102. Morrissy AS, Morin RD, Delaney A, et al. Next-Generation tag sequencing for cancer gene expression profiling. *Genome Res 2009;19:1825-35*.
103. Mullikin JC, Ning Z. The **PHUSION** assembler. http://www.ncbi.nlm.nih.gov/pubmed/12529309
104. Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., Flanigan, M.J., Kravitz, S.A., Mobarry, C.M., Reinert, K.H.J., Remington, K.A., et al. (2000) A whole-genome assembly of *Drossophila Science 287:2196-224*. http://www.sciencemag.org/content/287/5461/2196.full
105. N. Nagarajan, M. Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. J. Comput. Biol. 16 (2009) 897-908. http://www.ncbi.nlm.nih.gov/pubmed/19580519
106. Nava Whiteford, Niall Haslam, Gerald Weber, Adam Prügel-Bennett, Jonathan W. Essex, Peter L. Roach, Mark Bradley and Cameron Neylon. An analysis of the feasibility of short read sequencing. http://nar.oxfordjournals.org/content/33/19/e171.full
107. P. Ferragina, G. Manzini, V. Makinen and G. Navarro. An alphabet-friendly FM-Index. In Proc SPIRE'04, pp. 150-160, 2004. LNCS 3246. http://captura.uchile.cl/jspui/bitstream/2250/1901/1/Ferragina%20P.pdf
108. P.A. Pevzner, H. Tang, G. Tesler. De novo*De novo* repeat classification and fragment assembly. Genome Res. 14 (2004) 1786-1796. http://genome.cshlp.org/content/14/9/1786.long
109. P.A. Pevzner, H. Tang. Fragment assembly with double-barreled data. Bioinformatics 17 (Suppl 1) (2001) S225-S233. http://bioinformatics.oxfordjournals.org/content/17/suppl_1/S225.short
110. P.A. Pevzner. 1-Tuple DNA Sequencing: computer analysis. J. Biomol. Struct. Dyn. 7 (1989) 63-73. http://www.ncbi.nlm.nih.gov/pubmed/2684223
111. Página de **SSAKE** en el Canada's Michael Smith Genome Sciences Centre. http://www.bcgsc.ca/platform/bioinfo/software/**SSAKE**
112. Paolo Ferragina and Giovanni Manzini. Opportunistic Data Structures with Applications. 2000. P.390. Proceedings of the 41st Annual Symposium on Foundations of Computer Science. http://dimacs.rutgers.edu/Workshops/BWT/ferragina.pdf
113. Paul Medvedev, Konstantinos Georgiou, Gene Myers and Michael Brudno. Computability

of Models for Sequence Assembly. http://www.cs.utoronto.ca/~brudno/medvedev_etal_wabi07.pdf

114. Pavel A. Pevzner, Haixu Tang and Michael S. Waterman. An **EULER**ian path approach to DNA fragment assembly. http://bioinformatics.oxfordjournals.org/content/23/4/500.full

115. Pavel A. Pevzner, Haixu Tang and Michael S. Waterman. An **EULER**ian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. USA. 98 (2001) 9748-9753. http://nbcr.sdsc.edu/**EULER**/9748.pdf

116. Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research, 2010. Vol 38, No. 6 1767-1771. Doi:10.1093|nar|gkp1137*. http://nar.oxfordjournals.org/content/38/6/1767.full.pdf+html

117. Petterson E., Lundeberg J., Ahmadian A. Generations of sequencing technologies. *Genomics 2009;93:105-11*. http://www.ncbi.nlm.nih.gov/pubmed/18992322

118. Phil Green. 2x genomes-Does depth matter? Genome Res. November 2007 17:1675-1689;doi:10.1101/gr.6380007. http://genome.cshlp.org/content/17/11/1547.long

119. Pop M. Genome assembly reborn: rectn computational challenges. *Brief Bioinform 2009;10:354-66*.

120. Publicación sobre **ALLPATHS-LG**: http://www.pnas.org/content/early/2010/12/20/1017351108.full.pdf+html

121. R. L. Warren, R. A. Holt. **SSAKE** 3.0: Improved speed, accuracy and contiguity, Pacific Symposium on Biocomputing, 2008.

122. R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, G. Shan, K. Krsitiansen, H. Yang, J. Wang. De novo*De novo* assembly of human genomes with massively parallel short read sequencing. Genome Res. 20 (2009) 265-272. http://genome.cshlp.org/content/20/2/265.long

123. R. Li, W. Fan, G. Tian, H. Zhu, L. He, J. Cai et al. The sequence and de novo*de novo* assembly of the giant panda genome. Nature 463 (2009) 311-317. http://www.nature.com/nature/journal/v463/n7279/full/nature08696.html

124. R. Li, Y. Li, H. Zheng, R. Luo, H. Zhu, Q. Li, W. Quian, Y. Ren, G. Tian, J. Li, G. Zhou, X. Zhu, H. Wu, J. Qin, X. Jin, D. Li, H. Cao, X. Hu, H. Blanche, H. Cann, X, Zhang, S. Li, L. Bolund, K. Kristinansen, H. Yang, J. Wang, Building the sequence map of the human pan-genome. Nat. Biothecnol. 28 (2009) 57-63. http://www.nature.com/nbt/journal/v28/n1/full/nbt.1596.html

125. Reinhardt JA, Baltrus DA, Nishimura MT, Jeck WR, Jones CD, Dangl JL. De novo*De novo* assembly using low-coverage short read sequence data from the rice pathogen Pseudomonas syringae pv. Oryzae. Genome Res. 2009 Feb;19(2):294-305. Epub 2008 Nov 17. http://www.ncbi.nlm.nih.gov/pubmed/19015323

126. René L. Warren, Granger G. Sutton, Steven J. M. Jones and Robert A. Holt. Assembling millions of short DNA sequences using **SSAKE**. *Bioinformatics. October 2006*. http://bioinformatics.oxfordjournals.org/content/23/4/500.full

127. Robert M. Gray. Entropy and Information Theory. Springer-Verlag. 1990. http://ee.stanford.edu/~gray/it.pdf

128. Samuel Levy, Granger Sutton, Pauline C. Ng, Lars Feuk, Aaron L. Halpern, Brian P. Walenz, Nelson Axelrod, Jiaqi Huang, Ewen F. Kirkness, Gennady Denisov, Yuan Lin, Jeffrey R. MacDonald, Andy Wing, Chun Pang, Mary Shago, Timoty B. Stockwell, Alexia Tsiamouri, Vineet Bafna, Vikas Bansal, Saul A. Kravitz, Dana A. Busam, Karen Y. Beeson, Tina C. McIntosh, Karin A. Remington, Josp F. Abril, John Gill, Jon Borman, Yu-Hui Rogers, Marvin E. Frazier, Stephen W. Scherer, Robert L. Strausberg, J. Craig Venter. The Diploid Genome Sequence of an Individual Human. PLoS Biology. October 2007. http://www.plosbiology.org/article/info:doi/10.1371/journal.pbio.0050254

129. Shah, M.K., Lee, H., Rogers, S.A., Touchman, J.W.(2004) Computational Systems Bioinformatics Conference, An exhaustive genome assembly algorithm using *k-merK-meros* to indirectly perform n-squared comparisons in O(n) *(IEEE, New York), pp 740-741*. http://gatekeeper.dec.com/pub/toomany/158_Shah_M_Assembler.pdf

130. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I (2009) **ABYSS**: A parallel assembler for short read sequence data. *Genome Res 19:1117-1123*. http://genome.cshlp.org/content/19/6/1117.full

131. Slater, G.S.C., Birney, E.(2005) Automated generation of heuristics for biological sequence comparison. *BMC*

Bioinformatics 6:31, doi:10.1186/1471-2105-6-31. http://www.biomedcentral.com/1471-2105/6/31

132. Smith TF, Waterman MS (1981). Identification of common molecular subsequences. J. Mol. Biol. 147 (1): pp. 195-7. http://www.ncbi.nlm.nih.gov/pubmed/7265238?dopt=Abstract
133. Software de **ABYSS**: http://www.bcgsc.ca/platform/bioinfo/software/**ABYSS**
134. SONAR: http://www.sonarsource.com/plugins/plugin-c/overview/
135. Steven L. Salzberg, Daniela D. Sommer, Daniela Puiu, Vincent T. Lee. Gene-Boosted Assembly of a Novel Bacterial Genome from Very Short Reads. *PLOS*. http://www.ploscompbiol.org/article/info:doi/10.1371/journal.pcbi.1000186
136. Sundquist, A., Ronaghi, M., Tang, H., Pevzner, P., Batzolglou, S.(2007)W Whole-genome sequencing and assembly with high throughput, short-read technologies. *PLoS ONE 2:e484, doi:10.1371/journal.pone.0000484.* http://www.plosone.org/article/info:doi/10.1371/journal.pone.0000484
137. Sz. Grabowski, V. Makinen, G. Navarro and A. Salinger. A simple alphabetindependent FM-Index. In Proc. PSC'05, pp. 230-244, 2005. http://www.dcc.uchile.cl/~gnavarro/ps/psc05.3.pdf
138. Trapnell C, Pachter L, Salzberg SL. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics 2009;25:1105-11*.
139. Valen E, Pascarella G, Chalk A, et al. Genome-wide detecction and analysis of hippocampus core promoters using DeepCAGE. *Genome Res 2009;19:255-65*.
140. W. James Kent. BLAT – The BLAST-Like Alignment Tool. *Genome Res. 2002;12:656-64* http://genome.cshlp.org/content/12/4/656.long
141. Wang L, Jiang T. On the complexity of multiple sequence alignment. J Comput. Biol. 1994 Winter;1(4):337-48. http://www.ncbi.nlm.nih.gov/pubmed/8790475?dopt=Abstract
142. Wang Z, Gerstein M, Snyder M. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet 2009;10:57-63*.
143. Willian R. Jeck, Josephine A. Reinhardt, David A. Baltrus, Matthew T. Hickenbotham, Vincent Magrini, Elaine R. Mardis, Jeffery L. Dangl and Cobin D. Jones. Extending assembly of short DNA sequences to handle error. *Bioinformatics. July 23, 2007*. http://bioinformatics.oxfordjournals.org/content/23/21/2942.full
144. Xiaoqiu Huang and Anup Madan. **CAP3**: A DNA Sequence Assembly Program. 1999. http://genome.cshlp.org/content/9/9/868.long
145. Xiaoqiu Huang, Jianming Wang, Srinivas Aluru, Shiaw-Pyng Yang and LaDeana Hillier. **CAP**: A Whole- Genome Assembly Program. http://genome.cshlp.org/content/13/9/2164.full
146. Y. Lee, J. Tsai, S. Sunkara, S. Karamucheva, G. Pertea, R. Sultana, V. Antonescu, A. Chan, F. Cheung and J. Quackenbush. The **TIGR** Gene Indices: clustering and assembling EST and Known genes and integration with eukaryotic genomes. *Nucleic Acids Research, 2005. Vol. 33, Database issue D71-D74*. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC540018/
147. Yue Chen, John Carlis, Elizabeth Shoop and John Riedl. A High-throughoput System to Resolve Inconsistent Reading Frame Predictions for Expressed Sequence Tags. http://people.scs.carleton.ca/~bertossi/papersijcai01/Chen.pdf
148. **YAGA**: Versión gratuita para uso no comercial. http://www.bioinfor.com/**ZOOM**