Plataforma de supercomputación para bioinformática

Tesis Doctoral

Diego Darío Guerrero Fernández

2015

Director: Dr. M. Gonzalo Claros

AUTOR: Diego Darío Guerrero Fernández
http://orcid.org/0000-0001-6749-0962

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

TESIS DOCTORAL

Plataforma de supercomputación para bioinformática

Diego Darío Guerrero Fernández

Universidad de Málaga
Departamento de Biología Molecular y Bioquímica
Facultad de Ciencias
Plataforma Andaluza de Bioinformática
Edificio de Bioinnovación

Málaga. Mayo de 2015

Plataforma de supercomputación para bioinformática

Memoria presentada por:

Diego Darío Guerrero Fernández

Para optar al grado de Doctor por la Universidad de Málaga

Tesis realizada bajo la dirección del Dr. M. Gonzalo Claros Díaz en la Plataforma Andaluza de Bioinformática y el Departamento de Biología Molecular y Bioquímica de la Universidad de Málaga

Fdo. Diego Darío Guerrero Fernández

$V^o.B^o$. DIRECTOR DE LA TESIS DOCTORAL:

Fdo. M. Gonzalo Claros Díaz

Málaga, mayo de 2015

D. M. Gonzalo Claros Díaz, Investigador de la Plataforma Andaluza de Bioinformática y el Departamento de Biología Molecular y Bioquímica de la Universidad de Málaga.

CERTIFICA:

Que Don Diego Darío Guerrero Fernández, Ingeniero en Informática, ha realizado bajo mi dirección en el Departamento de Biología Molecular y Bioquímica y la Plataforma Andaluza de Bioinformática de la Universidad de Málaga, el trabajo de investigación recogido en la presente memoria de Tesis Doctoral que lleva por título: "Plataforma de supercomputación para bioinformática".

Tras la revisión de la presente Memoria se ha estimado oportuna su presentación ante la Comisión de Evaluación correspondiente, por lo que autorizo su exposición y defensa para optar al grado de Doctor.

Y para que así conste, en cumplimiento de las disposiciones legales vigentes, firmo el presente certificado.

Málaga, mayo de 2015

El Director de la Tesis,

Dr. D. M. Gonzalo Claros Díaz

D. M. Gonzalo Claros Díaz, Investigador del Departamento de Biología Molecular y Bioquímica de la Universidad de Málaga.

INFORMA:

Que Don Diego Darío Guerrero Fernández, Ingeniero en Informática, ha realizado bajo mi dirección en el Departamento de Biología Molecular y Bioquímica y la Plataforma Andaluza de Bioinformática de la Universidad de Málaga, el trabajo de investigación recogido en la presente memoria de Tesis Doctoral que lleva por título: "Plataforma de supercomputación para bioinformática", y que la misma cumple los requisitos de idoneidad necesarios para ser presentada por compendio de publicaciones. La memoria está avalada por los siguientes artículos:

- AlignMiner: a Web-based tool for detection of divergent regions in multiple sequence alignments of conserved sequences. - Guerrero, D., Bautista, R., Villalobos, D. P., Cantón, F. R., and Claros, M. G. - Algorithms for Molecular Biology.

- SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+ - Darío Guerrero-Fernández, Juan Falgueras, and M. Gonzalo Claros - Computational Biology Journal

- GENote v.β: A Web Tool Prototype for Annotation of Unfinished Sequences in Non-model Eukaryotes - Bioinformatics for Personalized Medicine - Noé Fernández-Pozo, Darío Guerrero-Fernández, Rocío Bautista, Josefa Gómez-Maldonado, Concepción Avila, Francisco M. Cánovas, M. Gonzalo Claros - Lecture Notes in Computer Science

Y para que así conste, en cumplimiento de las disposiciones legales vigentes, firmo el presente informe.

Málaga, mayo de 2015

El Director de la Tesis,

Dr. D. M. Gonzalo Claros Díaz

Proyectos de investigación

- Incremento de la eficiencia en el uso del agua en *Vitis vinífera* L.: bases genéticas y fisiológicas para una mejor adaptación al cambio climático (2014-2017, RTA2013-00068-C03-02, MINECO-INIA). IP: M.G. Claros
- Implementación de TECnologías INNOvadoras de mejora genética en lenguado senegalés (*Solea senegalensis*) y dorada (*Sparus aurata*) para la optimización de su producción industrial (INNOTECSS) (2014-2017, RTA2013-00023-C02-01, MINECO-INIA). IP: Manuel Manchado Campaña
- Desarrollo de herramientas bioinformáticas para los estudios genómicos y transcriptómicos a partir de datos de secuenciación de lecturas cortas de alto rendimiento para las especies que no tienen un organismo modelo de referencia (NEOGEN) (1-4-11 a 30-4-2016; Proyecto de Excelencia de la Junta de Andalucía, P10-CVI-6075). IP: M. G. Claros
- Arquitecturas, compiladores y aplicaciones en multiprocesadores - TIN2010-16144 [2011-2013] Ip: E. López Zapata y Oscar Plata
- Genomic tools in maritime PINE for enhanced biomass production and SUSTAINable forest Management (SUSTAINPINE). (2010-2013; MICINN and FP7-PLANT-KBBE Scientific Advisory Board PLE2009-0016). IP: Francisco Cánovas

Artículos y capítulos de libros

- Canales, J., Bautista, R., Label, P., Gómez-Maldonado, J., Lesur, I., Fernández-Pozo, N., ... Cánovas, F. M. (2014). De novo assembly of maritime pine transcriptome: Implications for forest breeding and biotechnology. Plant Biotechnology Journal, 12(3), 286–299. http://doi.org/10.1111/pbi.12136
- Benzekri, H., Armesto, P., Cousin, X., Rovira, M., Crespo, D., Merlo, M., ... Manchado, M. (2014). De novo assembly, characterization and functional annotation of Senegalese sole (Solea senegalensis) and common sole (Solea solea) transcriptomes: integration in a database and design of a microarray. BMC Genomics, 15(1), 952. http://doi.org/10.1186/1471-2164-15-952
- Darío Guerrero-Fernández, Juan Falgueras, M. Gonzalo Claros (2013). SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+. Computational Biology Journal: 10/2013; 2013. DOI:10.1155/2013/707540
- Darío Guerrero-Fernández, Rafael Larrosa, and M. Gonzalo Claros (2013). FQbin a compatible and optimized format for storing and managing sequence data. IWBBIO 2013, page 337-344.

- Noé Fernández-Pozo, Darío Guerrero-Fernández, Rocío Bautista, Josefa Gómez-Maldonado, Concepción Avila, Francisco M. Cánovas, M. Gonzalo Claros (2012). GENote β.1: A Web Tool Prototype for Annotation of Unfinished Sequences in Non-model Eukaryotes. Bioinformatics for Personalized Medicine - 10.1007/978-3-642-28062-7_7

- Claros, M. G., Bautista, R., Guerrero-Fernández, D., Benzerki, H., Seoane, P., and Fernández-Pozo, N. (2012). Why Assembling Plant Genome Sequences Is So Challenging. Biology. http://doi.org/10.3390/biology1020439

- Fernández-Pozo, N., Canales, J., Guerrero-Fernández, D., Villalobos, D. P., Díaz-Moreno, S. M., Bautista, R., ... and Claros, M. G. (2011). EuroPineDB: a high-coverage web database for maritime pine transcriptome. BMC genomics, 12(1), 366.

- Guerrero, D., Bautista, R., Villalobos, D. P., Cantón, F. R., and Claros, M. G. (2010). AlignMiner: a Web-based tool for detection of divergent regions in multiple sequence alignments of conserved sequences. Algorithms for Molecular Biology : AMB, 5, 24. http://doi.org/10.1186/1748-7188-5-24

Artículos en preparación o revisión

- Rosario Carmona, A. Zafra , Pedro Seoane, A. Castro, Darío Guerrero-Fernández,Trinidad Castillo, Ana Medina-García, Francisco M. Cánovas, José F. Aldana-Montes, Ismael Navas-Delgado, Juan D. Alché , M. Gonzalo Claros - ReprOlive: a Database with Linked-Data for the Olive Tree (Olea europaea L.) Reproductive Transcriptome - Frontiers in Journal (2015).

- Guerrero-Fernández, D., Bocinos, A., Bautista, R.Fernández-Pozo, Juan Falgueras and Claros, M. G. SeqTrimNext: pre-processing sequence reads for next-generation sequencing projects.

- Guerrero-Fernández and Claros, M. G. InGeBIOL: A web interface generator for command line tools.

- Fernández-Pozo, N., Guerrero-Fernández, D., Bautista, R. and Claros, M. G. FULLLENGTHERNEXT: A tool for fine-tuning de novo assembled transcriptomes of non-model organisms.

Comunicaciones orales en congresos

- Darío Guerrero-Fernández, Noé Fernández-Pozo, Almudena Bocinos, Rocío Bautista and M. Gonzalo Claros. “Highly efficient pre-processing of NGS reads and identification of full-length genes” - JBI2012, Barcelona. Enero 2012.

- D. Guerrero, A. Bocinos, R. Bautista, J. Falgueras, M.G. Claros. "SeqTrimNext: preprocessing for NGS". RES Scientific Seminar of Supercomputing and Next Generation Sequencing, Málaga. Marzo 2011.

- D. Guerrero, "Infraestructuras del DataCenter" - IIR Datacenter design. Madrid. Noviembre 2011.

- D. Guerrero, "Introducción al uso de las herramientas bioinformáticas de la PAB" - I Curso PAB de análisis de micromatrices - Málaga. 2010.

Otras colaboraciones y comunicaciones a congresos

- P. Seoane, R. Carmona, R. Bautista, D. Guerrero-Fernández, M.G. Claros. AutoFlow: an easy way to build workflows. International Work-Conference on Bioinformatics and Biomedical Engineering IWBBIO14. Granada, abril 2014.

- Pedro Seoane, Rosario Carmona, Rocío Bautista, Darío Guerrero-Fernández y M.G. Claros. «Using Autoflow, a workframe to resolve workflows, to build a de novo plant transcriptome». Plant Genomics Congress, Londres (UK), 12-13 de mayo de 2014

- Hicham Benzekri, Darío Guerrero-Fernández, Rocío Bautista, and M.G. Claros. "Detecting and correcting mis-assembled reads in contigs". International Work-Conference on Bioinformatics and Biomedical Engineering IWBBIO13. Granada, marzo 2013

- Hicham Benzekri, Rocío Bautista, Darío Guerrero-Fernández, Noé Fernández-Pozo, M. G Claros. «A reliable pipeline for a transcriptome reference in non-model species». International Conference The next NGS Challenge: data processing and integration. Valencia, mayo 2013

- H. Benzekri, N. Fernández-Pozo, D. Guerrero-Fernández, R. Bautista, M.G. Claros "Aproximación bioinformática al transcriptoma de *Solea* y su disponibilidad en SoleaDB". Biotecnología y recursos genómicos aplicados a la acuicultura. Avances logrados en AQUAGENET. Puerto Real, mayo 2012.

- N. Fernández-Pozo, D. Guerrero-Fernández, R. Bautista, J. Gómez-Maldonado, C. Avila, F.M. Canovas, M.G. Claros. «GeNOTE: a web tool for annotation of non-model eukaryotic, unfinished sequences». Workshop on Bioinformatics for Personalized Medicine (X Jornadas de Bioinformática). Málaga, 27-29/05/10 INT

- D. Guerrero-Fernández, R. Bautista, D.P. Villalobos, F.R. Cantón, M.G. Claros. "Detection of divergent regions in aligned conserved sequences with AlignMiner". Workshop on Bioinformatics for Personalized Medicine (X Jornadas de Bioinformática). Málaga, 27-29/05/10 INT

- M.G. Claros, R Bautista, N Fdez-Pozo, D Guerrero, J Falgueras "Free bioinformatics tools for forestry genomics at the Andalusian Bioinformatics Platform". Sustainable forest Management: genomic and biotechnological resources, Baeza, setiembre de 2009

Agradecimientos

Como no podría ser de otra forma, aquí podéis ver un script de agradecimientos escrito en Ruby para todas las personas que han influido en este trabajo.

```ruby
#!/usr/bin/env ruby
# encoding: utf-8

def agradecimientos(personas)
  personas.each do |persona, agradecimiento|
    puts "#{agradecimiento}\n\n"
  end

  puts "Gracias a todos,\n\n      Darío";
end

lista_de_personas={
  gonzalo_claros: "Gracias Gonzalo, por dirigir esta tesis junto con otros tantos
       proyectos que hemos compartido, y por explicarme tantas cosas importantes
      como la diferencia entre la mantequilla y la margarina ;).",
  rocio_bautista: "Rocío, tantos años que llevamos juntos me han dado tiempo para
       aprender muchas cosas de tí, y a contagiarte algunas manías de informático
      . Gracias por todo.",
  rafael_larrosa: "Rafa, mi maestro en el mundo de la supercomputación y
      compañero de penurias como administrador de sistemas, nadie podría
      imaginarse la cantidad de fuegos que hemos apagado en los últimos 8 años y
      espero que podamos seguir apagando en el futuro.",
  compañeros_scbi: "Pepi, Carlos, Diego, Noé, Hicham, Isabel, Pedro, David,
      Rosario y creo que no me dejo a ninguno: también vosotros habéis
      participado de un modo u otro en los trabajos de esta tesis, ya sea siendo
      mis conejillos de indias, ayudándome a entrenar mis niveles de paciencia u
      obligándome a aprender cosas nuevas.",
  juan_carlos_y_manolo:"Juan Falgueras, Carlos Rossi y Manuel Enciso: habéis sido
       un referente para mí desde mis comienzos en el mundo de la informática, sé
       que siempre tendré un apoyo en vosotros y unos amigos con los que discutir
       sobre nuestras pasiones tecnológicas.",
  mis_padres_y_hermano: "Diego y Paqui, mis padres, y a tí Javier, mi hermano,
      gracias. Siempre me habéis incentivado para que progrese en la vida, y
      habéis estado ahí para que pudiese hacerlo. Este trabajo también es vuestro
      .",
  mi_mujer:"Almudena, desde que nos conocimos hace 18 años has sido mi ilusión y
      todavía me parece que fué ayer cuando empezamos nuestra vida juntos.
      Gracias por compartirlo todo conmigo. Te quiero."
}

agradecimientos(lista_de_personas)
```

Resumen y contextualización

En el año 2007 la Universidad de Málaga amplió y trasladó sus recursos de cálculo a un nuevo centro dedicado exclusivamente a la investigación: el edificio de Supercomputación y Bioinnovación sito en el Parque Tecnológico de Andalucía. Este edificio albergaría también la Plataforma Andaluza de Bioinformática junto con otras unidades y laboratorios con instrumentación muy especializada. Desde aquel momento he trabajado como administrador de los recursos de supercomputación del centro y como parte del equipo bioinformático para proporcionar soporte a un gran número de investigadores en sus tareas diarias. Teniendo una visión de ambas partes, fue fácil detectar las carencias existentes en la bioinformática que podían ser cubiertas con una aplicación adecuada de los recursos de cálculo disponibles, y ahí es donde surgió la semilla que nos llevó a comenzar los primeros trabajos que componen este estudio.

Al haberse realizado en un entorno tan orientado a la resolución de problemas como el que hemos descrito, esta tesis tendrá un carácter eminentemente práctico, donde cada aportación realizada lleva un importante estudio teórico detrás, pero que culmina en un resultado práctico concreto que puede aplicarse a problemas cotidianos de la bioinformática o incluso de otras áreas de la investigación. Así, con el objetivo de facilitar el acceso a los recursos de supercomputación para los bioinformáticos, hemos creado un generador automático de interfaces web para programas que se ejecutan en línea de comandos, que permite ejecutar los trabajos utilizando recursos de supercomputación de forma transparente para el usuario. Además aportamos un sistema de escritorios virtuales que permiten el acceso remoto a un conjunto de programas ya instalados que proporcionan interfaces visuales para analizar pequeños conjuntos de datos o visualizar los resultados más complejos que hayan sido generados con recursos de supercomputación.

Para optimizar el uso de los recursos de supercomputación hemos diseñado un nuevo algoritmo para la ejecución distribuida de tareas, que puede utilizarse tanto en el diseño de nuevas herramientas como para optimizar la ejecución de programas ya existentes. Por otra parte, preocupados por el incremento en la cantidad de datos producidos por las técnicas de ultrasecuenciación, aportamos un nuevo formato de compresión de secuencias, que además de reducir el espacio de almacenamiento utilizado, permite buscar y extraer rápidamente cualquier secuencia almacenada sin necesidad de descomprimir el archivo completo.

En el desarrollo de nuevos algoritmos para resolver problemas biológicos concretos, proporcionamos cuatro herramientas nuevas que abarcan la búsqueda de regiones divergentes en alineamientos, el preprocesamiento y limpieza de lecturas obtenidas mediante técnicas de ultrasecuenciación, el análisis de transcriptomas de especies no modelo obtenidos mediante ensamblajes *de novo* y un prototipo para anotar secuencias genómicas incompletas.

Como solución para la difusión y el almacenamiento a largo plazo de resultados obtenidos en diversas investigaciones, se ha desarrollado un sistema genérico de máquinas virtuales para bases de datos de transcriptómica que ya está siendo utilizado en varios proyectos. Además, con el ánimo de difundir los resultados de nuestro trabajo, todos los algoritmos y herramientas productos de esta tesis se han publicado como código abierto en `https://github.com/dariogf`.

Índice general

Acrónimos y Abreviaturas

Aa aminoácido

ADN ácido desoxirribonucleico

AFP Apple File Protocol

API interfaz de programación de aplicaciones (del inglés *Application Programming Interface*)

ARN ácido ribonucleico

BAM Binary Alignment Map

CIFS Common Internet File System

CPD Centro de Procesamiento de Datos

CPU unidad central de procesamiento

DNS Domain Name Server

EBI European Bioinformatics Institute

EC Enzyme Commission

EMBL Laboratorio Europeo de Biología Molecular

EST etiquetas de secuencias expresadas (del inglés *Expressed Sequence Tag*)

FDR Fourteen Data Rate

FIFO First In First Out

FTP protocolo de transferencia de archivos (del inglés *File Transfer Protocol*)

GO Gene Ontology

GPFS General Parallel File System

GPU unidad de procesamiento gráfico

HPC High Performance Computing

IP Internet Protocol

KEGG Kyoto Encyclopedia of Genes and Genomes

KVM Kernel-based Virtual Machine

LDAP Lightweight Directory Access Protocol

MID identificadores multiplexados

MPI Message Passing Interface

NCBI National Center for Biotechnology Information

NFS Network File System

NGS secuenciación de nueva generación (del inglés *next generation sequencing*)

NIS Network Information Service

nt nucleótido

OLC Overlap/Layout/Consensus

PAB Plataforma Andaluza de Bioinformática

PCR reacción en cadena de la polimerasa

QDR Quadruple Data Rate

RAM memoria de acceso aleatorio

RDP Remote Desktop Protocol

REST Representational State Transfer

RoR Ruby-on-Rails

SAM Sequence Alignment Map

SBS Sequencing by Synthesis

SCBI Centro de Supercomputación y Bioinformática de la UMA

SMB Server Message Block

SMFS Single Molecule Fluorescent Sequencing

SMRT Single Molecule Real-Time sequencing

SMS Single Molecule Sequencing

SNP polimorfismos mononucleotídicos

SRA sequence Read Archive

SSH Secure SHell

SSR repeticiones de secuencias simples

TB terabyte, unidad de almacenamiento equivalente a 1000 o 1024 gygabytes

UMA Universidad de Málaga

URL Uniform Resource Locator

VNC Virtual Network Computing

WGA secuenciación de todo el genoma (del inglés *whole genome sequencing*)

X11 Interfaz gráfica para sistemas Linux/UNIX

Parte I

INTRODUCCIÓN

Capítulo 1

Bioinformática

1.1. Visión general

La bioinformática se puede considerar una de las áreas científicas que más rápido está avanzando en los últimos años. Continuamente se producen nuevos avances tecnológicos que permiten realizar experimentos más complejos y con un coste más reducido. Esto ha propiciado que la cantidad de resultados obtenidos esté aumentando considerablemente [143].

Para que el enorme volumen de información que se está acumulando sea lo más útil posible, hay que desarrollar repositorios complejos para estos conocimientos que es necesario mantener vivos y accesibles para futuros proyectos. Un ejemplo claro de esta tendencia podemos encontrarlo en la gráfica de la figura 1.1, donde se observa la evolución de la base de datos sequence Read Archive (SRA) [102], que almacena datos en bruto de proyectos de secuenciación masiva. Es fácil intuir que esta cantidad de datos será cada vez mayor, y esto ya está provocando problemas de procesamiento y almacenamiento considerables, no sólo desde el punto de vista de que la cantidad de datos generada sobrepase el límite manipulable por una persona por medios manuales, sino que también sobrepasa el límite de los datos que una persona puede analizar con un sistema informático medio y es necesario utilizar recursos de supercomputación. Recientemente se ha

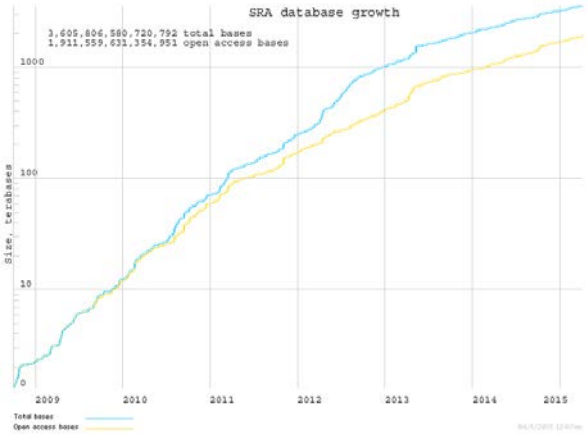


Figura 1.1: *Evolución de la cantidad de secuencias brutas almacenadas en la base de datos SRA. Fuente: NCBI-*`http://www.ncbi.nlm.nih.gov/Traces/sra/`

acuñado el término BigData [8] para referirse a los problemas relativos a la manipulación de ingentes cantidades de datos, que no son únicos de la bioinformática sino que se encuentran en áreas tan diversas como la mercadotecnia, las redes sociales, y el análisis de mercados o tendencias [135]. Hoy en día se proponen técnicas y entornos de trabajo como el MapReduce o Hadoop en grandes granjas de servidores para procesar tal cantidad de datos en tiempos razonables [107, 159], para lo que se necesita de manera indiscutible la bioinformática.

La informática además puede aplicarse a la biología para resolver problemas, sobre todo

cuando la cantidad de datos disponibles hace inviable su manipulación manual o tradicional, lo que abrió el campo de la bioinformática. Algunas aplicaciones de la bioinformática son:

- Análisis estadísticos e interpretación automática de resultados.
- Estudio de redes funcionales mediante la interacción entre genes.
- Análisis de la estructura, función e interacciones de las proteínas. Este fue uno de los primeros problemas que se abordó con la bioinformática y sigue siendo un problema que exige grandes recursos informáticos.
- Ensamblaje de secuencias y anotación.
- Análisis de expresión.
- Minería de datos y de textos.
- Genómica comparativa, filogenia, estudios evolutivos y clasificación de especies.
- Determinación de mutaciones, reconocimiento de marcadores y polimorfismos mononucleotídicos (SNP).

1.2. Algunas limitaciones

De un modo u otro, todos los campos de la bioinformática presentan una serie de problemas que limitan su máximo aprovechamiento. Entre ellos nos encontramos con:

- Enorme variedad de pequeños programas que realizan funciones individuales.
- Conexión insatisfactoria entre dichos programa porque se necesita la traducción de formatos de archivos entre un programa y otro, así como la manipulación de los datos para adaptar su estructura.
- Mucho trabajo repetitivo que se puede simplificar con el desarrollo de flujos de trabajo automáticos para poder aplicar un mismo procedimiento a diferentes conjuntos de datos de forma eficiente.
- Gran cantidad de datos. En sus dos vertientes más problemáticas: muchos archivos pequeños y archivos extremadamente grandes.
- Almacenamiento ordenado y difusión de grandes cantidades de datos a los usuarios finales de forma satisfactoria.
- Tiempos de ejecución demasiado elevados.
- Poca paralelización de las aplicaciones o de los algoritmos.
- Altos requisitos de memoria RAM. Los algoritmos se pensaron para pequeñas cantidades de datos, al ampliar los datos de entrada los requisitos de memoria se multiplican.
- Complejidad en la instalación y manejo de los programas. Casi todos los programas se utilizan en un terminal mediante línea de comandos lo que provoca que no sean cómodos para la mayoría de investigadores, que aunque disponen de los conocimientos necesarios para su área, no son expertos en informática.

1.3. Ordenadores virtuales

Dada la complejidad de instalación de algunas herramientas bioinformáticas y la enorme cantidad de herramientras disponibles, una de las posibilidades que han tenido los usuarios menos expertos en bioinformática es usar distribuciones de sistemas operativos con este tipo de programas ya instalados [60], ya sea en una

máquina real o en forma de máquina virtual. De esta forma, la comunidad científica no tiene que dedicar un esfuerzo adicional a preparar un entorno de trabajo con las herramientas bioinformáticas que necesitan. Las primeras distribuciones las encontramos de mano de los laboratorios del CERN y Fermilab, cada uno con sus propias distribuciones Linux que usaban internamente (CERN Linux y Fermi Linux respectivamente). A principios del 2004 decidieron unir esfuerzos y lanzar Scientific Linux [52] que es una de las distribuciones más utilizadas en la actualidad. También existen otras distribuciones que siguen la misma filosofía, como Poseidon Linux [59]. Las alternativas más usadas [177] son BioLinux [60], BioPuppy [96], DNALinux [20] o LXtoo [239] que se centran en el software útil para distintos aspectos de la bioinformática. Cubren la instalación unificada y simple de paquetes de herramientas bioinformáticas en los ordenadores de los usuarios y también la instalación de máquinas virtuales que proporcionan el entorno totalmente configurado. Esta es una buena idea para facilitar el acceso de los investigadores a las herramientas bioinformáticas necesarias, pero no ofrece el acceso a los recursos de supercomputación, por lo que no se pueden usar para tareas computacionales intensivas. Sería ideal disponer de algún tipo de acceso de escritorio remoto que sea compatible con el uso de recursos de supercomputación para las situaciones en las que los análisis no se pueden realizar en una máquina personal.

1.4. Infraestructuras centralizadas

Por motivos de aprovechar mejor la financiación, la organización interna, el montaje, soporte de los programas, la formación y mantenimiento del sistema, en todas partes del mundo se están desarrollando infraestructuras centrales para apoyar los análisis bioinformáticos de los grupos de investigación [114] Aunque existen muchas variantes en función de su evolución histórica y el contexto científico en el que se generaron, una de las grandes ventajas es que las personas que la componen son capaces de aprender nuevas metodologías y expandirse hacia nuevas áreas de conocimiento de la bioinformática con más facilidad que los componentes de los grupos de investigación de laboratorio, pero no deja de ser importante contratar personal que tenga conocimientos afianzados en alguna de las áreas en las que dará servicio el centro, ya que eso proporcionará una garantía de su capacidad de adaptarse a otras, o al menos aportar un punto de vista alternativo a los problemas. [115].

Aún así, siguen siendo necesarios más bioinformáticos para procesar todos los datos que se generan hoy en día [41], por lo que no parece que sean las infraestructuras el paso limitante, sino el personal capaz de usarlas. Quizá uno de los problemas sea que la carrera profesional de bioinformático no está bien definida, y se suele basar en colaboraciones multidisciplinarias que no obtienen una recompensa personal o profesional a la medida del esfuerzo invertido, pues se les considera más un apoyo técnico que un colaborador indispensable en la investigación, según comenta el codirector del Bioinformatics Service Center de la Universidad de Texas (EE. UU.) [41].

De hecho, uno de los principales problemas que aparecen al ofrecer servicios de bioinformática de forma centralizada es que en muy pocos proyectos de investigación se pueden aplicar análisis automatizados (por ejemplo, solo el 20 % en la Universidad de Texas), mientras que la mayoría requieren una adaptación de los análisis al problema biológico concreto, o sea, necesitan investigación bio-

informática. Por tanto, hay que tender hacia unos análisis bioinformáticos en los que, aunque haya una infraestructura centralizada que facilite la parte informática y bioinformáticos que conozcan muy bien el problema biológico que hay que tratar, lea los artículos científicos relevantes del área, y esté completamente inmerso en la investigación, también se impliquen de lleno los científicos promotores de la investigación y colaboren en el análisis de los datos y en la toma de decisiones, pues ellos son realmente los que tienen mayor experiencia en el área [80].

La mayoría de estos servicios centrales para bioinformática necesitan el apoyo de recursos de supercomputación. Algunos centros optan por un modelo híbrido [116] en el que alojan sus propios servidores en centros de cálculo que les proporcionan un valor añadido en cuanto a mantenimiento y experiencia en administración de equipos del personal, mientras que otros delegan toda la carga computacional en servicios de supercomputación externos, y se limitan exclusivamente a usar los programas allí instalados, liberándose totalmente de la gestión de los servidores y la infraestructura. En el caso de supercomputación, una infraestructura centralizada implica menos problemas que en el caso de la bioinformática, pues los ordenadores desde el punto de vista bioinformático son sólo una herramienta y no una variable de la investigación como puede ser las técnicas de secuenciación o las aplicaciones utilizadas para los análisis.

Capítulo 2

Supercomputación para principiantes

2.1. Computación de altas prestaciones

Cuando los problemas bioinformáticos que estamos abarcando no pueden solucionarse con los equipos informáticos que puede tener disponibles un laboratorio normal, llega la hora de echar mano de la supercomputación.

La supercomputación consiste básicamente en coordinar la utilización de grandes cantidades de recursos de cómputo para resolver problemas específicos, como puede ser el ensamblaje o análisis de datos biológicos [170]. Dichos recursos pueden estar formados por grandes servidores de memoria compartida o por una agrupación de gran cantidad de equipos más pequeños llamado *cluster*. Cada tipo de sistema presenta unas ventajas respecto al otro, y cada uno de ellos es apropiado para resolver un tipo de problemas, por lo cual un centro de supercomputación que tenga la intención de ser versátil, debe disponer de todos ellos para adaptarse a problemas de todo tipo.

Aparte de los sistemas de cálculo (unidad central de procesamiento (CPU) y memoria), un supercomputador necesita una serie de elementos no menos importantes, como pueden ser un alojamiento apropiado, almacenamiento y copias de seguridad, redes de alto rendimiento y programas que faciliten el uso y gestión de los recursos. El conjunto de elementos físicos necesarios suele llamarse CPD [17]. En los siguientes apartados se explicará con más detalles cada uno de sus elementos.

2.2. Infraestructura del CPD

Un supercomputador no puede alojarse en cualquier habitación con un aire acondicionado doméstico, es necesario contar con una infraestructura correctamente dimensionada [18] para el equipamiento que va a alojar. En la figura 2.1 podemos ver el CPD de la Universidad de Málaga (Universidad de Málaga (UMA)), cuyo diseño hemos esquematizado en la figura 2.2 y que nos servirá de apoyo para identificar todas las partes que componen un CPD.

La sala destinada a un CPD debe proporcionar un espacio amplio, con el menor número posible de elementos que entorpezcan la distribución de los sistemas y además tener una altura suficiente para poder alojar los armarios (*racks*). A esa altura habrá que descontar la necesaria para la posible instalación de un suelo flotante capaz de soportar el peso de los equipos que van a instalarse encima (un único ar-

Figura 2.1: *Foto de la sala del supercomputador Picasso de la Universidad de Málaga*

mario puede pesar más de 1000 kg y tiene una superficie de algo menos de 1 m^2). El suelo flotante es especialmente práctico para distribuir con libertad los cableados de energía o redes de comunicación y al mismo tiempo se puede utilizar para hacer llegar el aire acondicionado a cualquier punto de la sala. En algunos CPD también se llevan las instalaciones con bandejas guía que van colgadas del techo. Igual que el suelo flotante debe estar pensado para soportar un peso considerable, la estructura del edificio debe estar igualmente preparada. En el caso del CPD de la UMA, la sala tiene el doble de altura que uno convencional. Con tal volumen de aire no tiene sentido enfriar la sala al completo, por lo que cada uno de los 11 armarios dispone de una mampara de metacrilato transparente que dirige el aire justo a la zona de aspiración de los servidores. El suelo flotante se encuentra reforzado y es utilizado para distribuir tanto el aire acondicionado como el cableado de alimentación y las redes de comunicaciones.

En un CPD orientado a supercomputación debe tenerse en cuenta que el consumo eléctrico va a ser realmente importante, a pesar de la bajada en el consumo de los servidores (figura 2.3), los niveles de integración han aumentado, de modo que el consumo por armario se mantiene más o menos estable (un único armario llega a consumir entre 15 y 20 kW), es necesario contar con líneas de suministro adecuadas, y un aire acondicionado especializado capaz de contrarrestar el calor generado por

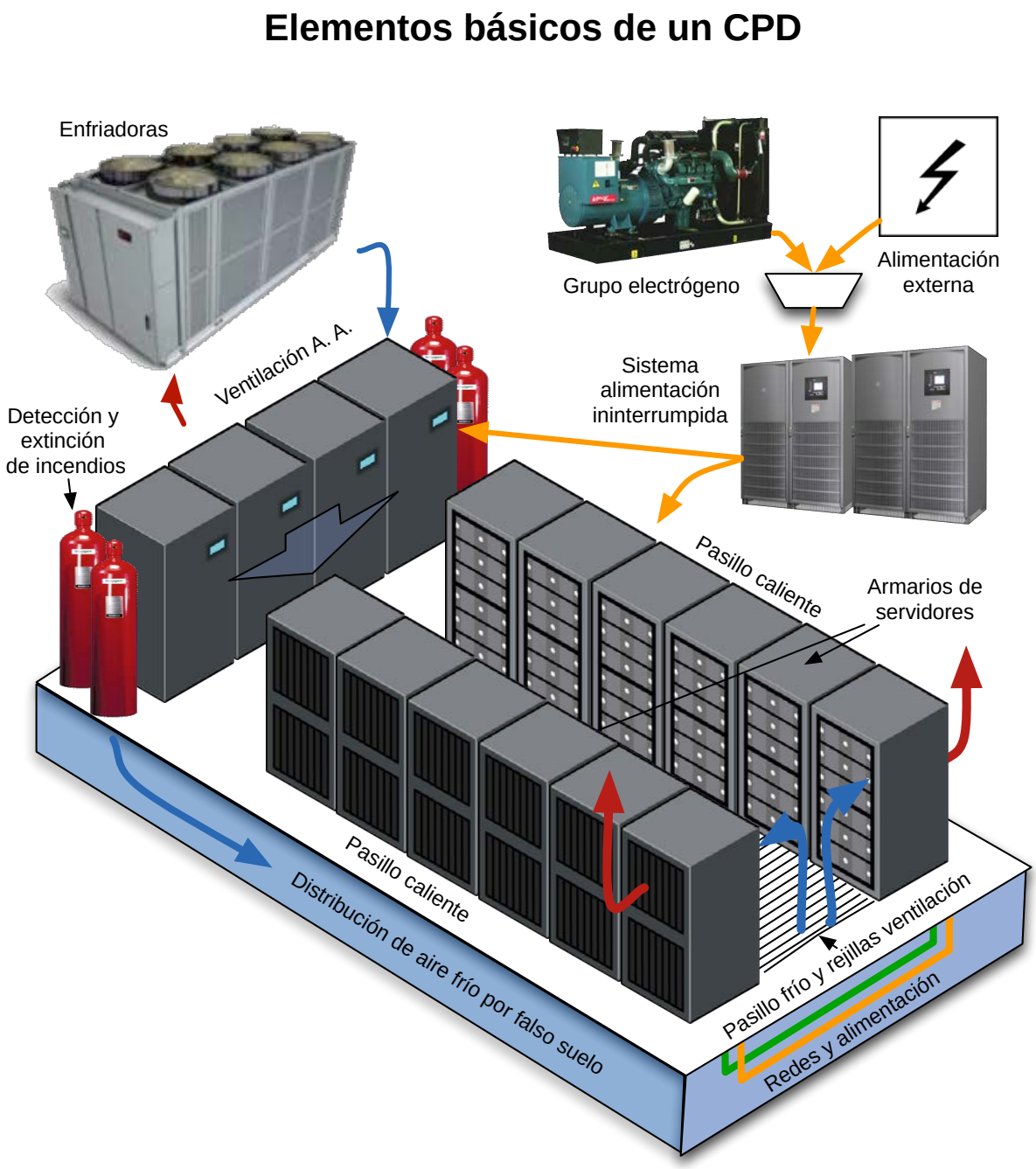


Figura 2.2: *Esquema de los componentes básicos que podemos encontrar en un CPD*

los equipos [7].De igual forma, es interesante disponer de dispositivos de alimentación ininterrumpida y generadores eléctricos auxiliares para evitar interrupciones en el servicio [178]. El CPD de la Universidad de Málaga cuenta con la última generación de ordenadores HP que consumen de media 11 kW por armario (equivalente a unos 40 W por cada núcleo), 4 compresores de climatización industrial capaces de mantener estables la humedad y la temperatura, además de un sistema de alimentación ininterrumpida de 160 kW respaldado a su vez con un grupo electrógeno de 420 kW que alimenta todo el edificio en caso de un corte en el suministro eléctrico externo. En los CPD de grandes dimensiones es recomendable estudiar la situación física del edificio en base al precio del suministro eléctrico, (pues este será el principal gasto fijo de la instalación), opciones para realizar *free-cooling* (enfriar por métodos naturales), disponibilidad de líneas de comunicaciones y probabilidad de catástrofes naturales como inundaciones o terremotos.

La seguridad física es igualmente indispensable, puesto que ningún equipo informático es seguro si puedes acceder físicamente a él.

Los sistemas de extinción de incendios para salas de supercomputación deben cumplir unos requisitos especiales, no podemos apagar un fuego eléctrico con agua, a no ser que sea agua nebulizada, ni se puede rociar el CPD con ningún compuesto sólido que afecten a los circuitos electrónicos. En la mayoría de los casos se opta por un sistema de desplazamiento del oxígeno, que consiste en un conjunto de extintores de gran capacidad que son capaces de reemplazar en segundos todo el oxígeno de la sala por un gas no inflamable, con lo que se impide la combustión sin afectar a los componentes electrónicos. Nuestro CPD de referencia cuenta con un sistema de extinción por desplazamiento de oxígeno además de dispositivos de alerta temprana VESDA [82], que son capaces de detectar una presencia ínfima de humo debidas a sobrecalentamiento o explosión de cables, transistores o condensadores que pueden ser los primeros indicios de un incendio.

2.3. Componentes de un supercomputador

Además de la inevitable infraestructura de soporte, todo supercomputador necesita ordenadores, almacenamiento y redes. La utilización de armarios es lo más eficiente, puesto que permite mantener una organización adecuada de los servidores, cableado y activos de red a la vez que facilita las labores de mantenimiento (los supercomputadores también se estropean). Los elementos imprescindibles son:

Sistema de almacenamiento compartido: Cuando disponemos de varios ordenadores que van a colaborar para solucionar un problema, es interesante que todos ellos puedan acceder al mismo espacio de almacenamiento. Aunque no sea imprescindible, evita tener que copiar explícitamente los datos de un equipo a otro. Dichos sistemas suelen estar basados en equipos redundantes, de modo que un fallo en un componente no implique una parada del servicio. Sobre estos equipos redundantes se instala un sistema de archivos compartido determinado, como puede ser Lustre [197], General Parallel File System (GPFS) [195], o Network File System (NFS) [202]. Existen muchos otros sistemas alternativos, aunque en supercomputación estos son los más utilizados. El CPD de la UMA utiliza un sistema de almacenamiento Lustre soportado por una cabina redundante SFA10000 de DataDirect Networks que proporciona 740 TB de almacenamiento bruto.

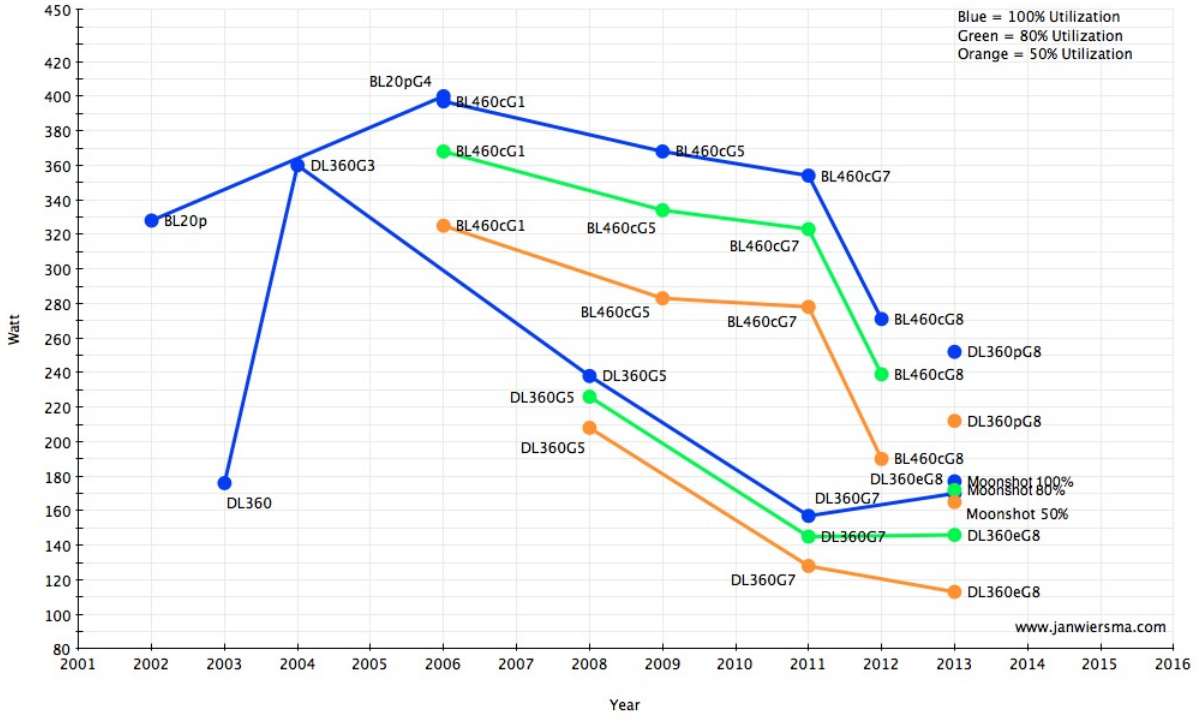


Figura 2.3: *Evolución del consumo de los servidores. Imagen obtenida de* `http://datacenterpulse.org/blogs/jan.wiersma/where_rack_density_trend_going`

Redes de comunicación: Existen multitud de redes de comunicación [3], que según su tecnología y velocidad podremos utilizar para diferentes propósitos. Las redes de comunicación más extendidas para gestión de los servidores y acceso a los servicios básicos suele ser Ethernet a 1 Gbps, mientras que para las comunicaciones de sincronización entre los servidores de cálculo y acceso al almacenamiento compartido se utilizan redes más rápidas y con la latencia más baja posible, como son Ethernet a 10 Gbps, Myrinet, Infiniband Quadruple Data Rate (QDR)/Fourteen Data Rate (FDR) a 40/56 Gbps o FiberChannel. En la configuración de la Universidad de Málaga se utiliza red Ethernet Gigabit para las redes de gestión y red Infiniband 56 Gbps o FiberChannel para el almacenamiento.

Servidores de cálculo: Como indica su nombre, serán los que realicen las ejecuciones de los programas. Lo ideal es disponer de equipos que cubran todas las situaciones que vayamos a encontrarnos, desde equipos con mucha memoria que permitan ejecutar programas sobre conjuntos de datos muy grandes hasta equipos normales que añaden capacidad de cómputo al sistema. Los grandes fabricantes de ordenadores disponen de líneas de servidores específicamente diseñados para crear supercomputadores. Aun así, también existe la posibilidad de crear *clusters* con componentes convencionales (llamados *clusters* Beowulf [210]) que pueden ofrecer un rendimiento aceptable a cambio de aumentar la proporción de errores de *hardware*. En ambos casos, los servidores de cálculo suelen disponer de un espacio local para datos temporales llamado *scratch*. A ve-

ces este espacio también puede estar compartido a través de una red de alto rendimiento. La UMA ha optado por una arquitectura heterogénea para los servidores de cálculo, que está formada por bloques de características muy definidas que hacen que se adapten perfectamente a trabajos de todo tipo. Las máquinas que contiene son:

- 32 máquinas HP SL230S-G8, cada una de 16 núcleos y 64 Gb de memoria de acceso aleatorio (RAM) con red Infiniband FDR 56 Gbps, para trabajos distribuidos que pueden beneficiarse de una red de alto rendimiento.
- 16 máquinas HP SL250S-G8 idénticas a las SL230 anteriores actualizadas con 2 tarjetas Tesla M2075 para proporcionar capacidad de cálculo adicional a programas capaces de utilizar las GPU.
- 7 máquinas HP DL980-G7 con 80 núcleos y 2.000 Gb de RAM cada una, con red Inifiniband QDR a 40 Gbps, que proporcionan la capacidad necesaria para realizar trabajos que exigen mucha memoria en una sola máquina, como son los ensamblajes de genomas.
- 41 máquinas HP DL165-G7 con 24 núcleos y 96 Gb de RAM cada una con red Gigabit Ethernet, que se utiliza para trabajos de paralelismo compulsivo que no necesitan mucho acceso a disco ni comunicación entre servidores.

Esta arquitectura tan diversificada no suele ser habitual en los grandes CPD, puesto que añadir máquinas tan diferentes implica mayores problemas con la gestión del mismo. Otro aspecto que produce reticencias para adoptar este tipo de arquitecturas donde se mezclan máquinas con mucha memoria con máquinas normales es que muchos CPD orientados a supercomputación priman exclusivamente la capacidad de cálculo (número de núcleos o tarjetas GPU) antes que la capacidad de memoria por servidor, cuyo coste sube exponencialmente.

La mayoría de servidores de cálculo usados para formar un supercomputador presentan características comunes y están alojados en CPD, con el objetivo de obtener un rendimiento homogéneo y previsible, facilitar su gestión y poder controlar la seguridad física necesaria. Sin embargo, existen otras alternativas para obtener una capacidad de cálculo elevada como es el caso de los *grids* de computación [24]. Un *grid* consiste en agrupar equipos que pueden ser muy heterogéneos, con procesadores de arquitecturas diferentes, redes de comunicación de distintas tecnologías, velocidades y latencias, almacenamientos diferentes e incluso sistemas operativos distintos. Un caso muy habitual consiste en agrupar los equipos de sobremesa presentes en las aulas de enseñanza o laboratorios [29] para poder aprovechar su capacidad de cálculo mientras los usuarios no los están utilizando, aunque también existen grandes projectos como SETI@home [103], que han utilizado este esquema de computación distribuida a gran escala, pero haciendo uso de los ordenadores personales de las personas que voluntariamente se unían al proyecto. El sistema de colas utilizado en un *grid* debe ser capaz de lidiar con los problemas de fiabilidad provocados por una arquitectura tan diversa, como por ejemplo detectar errores en los equipos del *grid* para poder enviar de nuevo las ejecuciones fallidas. HTCondor [218, 219] es uno de los sistemas más utilizados para computación en *grid*.

Las máquinas de entrada: Entre los servidores de cálculo suele haber uno con carac-

terísticas especiales, que suele ser la parte visible desde el exterior y que es el equipo donde se conectará todo usuario. Desde él se interacciona con el superordenador para enviar trabajos al sistema de colas (véase el apartado 2.5.3) y ver resultados. El sistema de colas determinará el conjunto de máquinas que el usuario necesita y se encargará de reservarlas y realizar la ejecución en las mismas de forma transparente.

Servidores de soporte: Para el funcionamiento y monitorización del CPD es necesario disponer de una serie de servicios básicos: bases de datos de usuarios (Lightweight Directory Access Protocol (LDAP)), servidores de nombres (Domain Name Server (DNS)), servidores de sistemas de colas. Dichos sistemas pueden alojarse sobre máquinas físicas o puede optarse por usar un sistema de virtualización de servidores (que es la opción elegida para el CPD de la UMA) que además pueden facilitar la gestión y mejorar los tiempos de respuesta ante fallos.

Sistemas para copias de seguridad: Aunque en supercomputación no se suele ofrecer una copia de seguridad de los datos de usuario (debido a la gran cantidad de datos que se manejan sería muy costoso), sí que es imprescindible tenerlas de los propios sistemas y servicios para poderlos recuperar ante cualquier imprevisto. Se suelen emplear cintas, discos, sistemas de almacenamiento compartido con replicación automática, etc. En el caso del CPD de la UMA, se proporciona a cada usuario su almacenamiento dividido en varios bloques, un almacenamiento de espacio reducido del que se realizan copias de seguridad (su directorio $HOME), y un almacenamiento mucho más amplio para realizar los cálculos y del que no se realizan copias de seguridad (su directorio $SCRATCH).

2.4. Herramientas y programas en un supercomputador

Para que un supercomputador resulte útil a los usuarios y se pueda gestionar sin problemas, los equipos que lo componen necesitan tener instaladas determinadas herramientas:

Sistema operativo y herramientas de replicación: Los servidores que forman un supercomputador también necesitan un sistema operativo instalado, aunque no necesitan su interfaz gráfica. Lo normal es que todos los equipos contengan el mismo sistema operativo instalado con el objetivo de facilitar la gestión y evitar problemas por incompatibilidades. A pesar de los intentos de Microsoft® para introducirse en el mercado de la supercomputación [235], la gran mayoría de supercomputadores del mundo utilizan alguna distribución de Linux o UNIX, ya sea alguna versión de pago como Suse Linux Enterprise Server (SLES) [226] o RedHat Enterprise Linux (RHEL) [64] que ofrecen planes de soporte extendidos, o versiones gratuitas como OpenSuse, Fedora, Centos, Ubuntu, Debian o ScientificLinux [52]. También existen distribuciones de Linux específicamente diseñadas para montar *clusters* de forma rápida como Rocks Cluster Distribution [188]. Sea cual sea la versión de sistema operativo elegida, instalar un sistema operativo en cientos de servidores, y configurarlos manualmente no es una tarea agradable. Para evitarlo existen soluciones como HP Cluster Management Utility (herramienta de uso interno de HP) o xCAT [63] que permiten configurar un servidor, y replicar su sistema operativo en el resto de servidores que forman el cluster [221].

Herramientras de monitorización: Estas herramientas sirven para poder vigilar en todo momento lo que está pasando [9] tanto en los equipos como en el propio CPD, (por ejemplo, fallo del aire acondicionado, averías de algún componente de los equipos, caídas en el rendimiento, problemas en el cableado de la red de comunicaciones, etc.). Nagios [19] y Ganglia [136] son dos de las herramientas más utilizadas, aunque existen otras variantes como ICINGA [66], que es la utilizada en el CPD de la UMA.

Aplicaciones específicas para afrontar las investigaciones que quieren realizar los usuarios finales. Las hay de muchos tipos y a menudo requiere conocimientos avanzados para poder instalarlos correctamente en un entorno distribuido como el que estamos tratando. En el CPD de la UMA se utiliza Modules [65], un sistema de carga por módulos que ayuda a mantener los programas organizados en diferentes categorías. Cada usuario puede así decidir qué programa y versión desea utilizar en función de sus necesidades.

Sistema de colas: Se encarga del funcionamiento correcto de un *cluster* de computación al orquestar las ejecuciones de los diferentes programas que lanzan los usuarios y garantizar el uso correcto de todos los recursos del mismo. Entre los sistemas de colas más utilizados se encuentran Slurm [92], PBS, SGE [68], Torque/Maui [90], Moab [47] o Condor/HTCondor [218]. A pesar de tanta variedad, todos funcionan de un modo muy parecido. En el CPD de la UMA se utiliza Slurm como sistema de colas, de manera que el usuario accede a la máquina de entrada, crea un fichero (*script*) donde define los recursos que necesita (memoria, duración del trabajo, número de CPU o unidad de procesamiento gráfico (GPU), etc.), e indica el programa que desea ejecutar junto con sus argumentos de entrada. Luego envía el *script* al sistema de colas con el comando correspondiente y éste organiza la ejecución en base a los recursos solicitados, prioridades asignadas al usuario, otros trabajos en ejecución, y otros parámetros (más adelante, en en la figura 2.5 y en el apartado 2.5.3, se explicará esto con ficheros reales). Cuando en el *cluster* quedan libres suficientes recursos para comenzar la ejecución, el sistema de colas ejecuta el trabajo en los servidores de cálculo apropiados sin intervención del usuario. Durante el proceso, además de los resultados del programa que ha decidido lanzar el usuario, se suelen obtener otros ficheros con los resultados que el usuario hubiese visto en la pantalla si hubiese estado delante durante la ejecución. Visto de forma simplificada, el sistema de colas actúa como un delegado al que encargamos la ejecución del programa y luego nos cuenta qué ha pasado.

2.5. Supercomputación en la práctica

El principal obstáculo a la hora de aprovechar los recursos de un supercomputador es que se quieran ejecutar principalmente programa que solo sean capaces de aprovechar un núcleo de una CPU, con lo que no se hará un uso óptimo de los recursos. Peor aún es que haya usuarios que crean que sí se puede, con lo que reservan varias CPU, de las que solo se usará una y las demás estarán inutilizadas mientras dure la ejecución, con lo que se están desaprovechando de forma grave los recursos. Los usuarios, que muchas veces tienen que actuar como programadores, tendrán que saber o aprender las técnicas de paralelización y creación de flujos de trabajos que exponemos a continuación, además de utilizar correctamente el sistema de colas, si desean obtener los resul-

tados necesarios para sus investigaciones en el menor tiempo posible.

2.5.1. Paralelización

Un supercomputador no tiene porqué ser más rápido que un ordenador convencional si no se es capaz de resolver el problema en paralelo. Muchos usuarios entran en el sistema y ejecutan en vivo algún programa para ver lo rápido que es, llevándose la sorpresa de que es sólo un poco más rápido que su ordenador personal. La verdadera potencia de un supercomputador se debe a que se pueden ejecutar problemas en paralelo para reducir proporcionalmente el tiempo usado respecto a una ejecución de forma secuencial. Conseguir esto implica un esfuerzo importante de programación, aunque existen otros paradigmas más complejos como las redes de actores [76] que se utilizan para obtener paralelismo, así como lenguajes con aproximaciones puramente paralelos, como Chappel [192], Scala [158] o Swift [233] que pueden simplificar en gran medida la paralelización. Existen muchas formas de clasificar el paralelismo, pero vamos a centrarnos en las clasificaciones basadas en (1) qué parte del problema se paraleliza, (2) dónde se ejecutan los trabajos y (3) cómo se comunican entre sí los procesos paralelos.

Respecto a la la parte del algoritmo que se paraliza, cuando la ejecución paralela se obtiene porque un mismo programa, algoritmo, o conjunto de instrucciones se ejecuta de forma simultánea sobre diferentes conjuntos de datos de entrada, hablamos de paralelismo orientado a datos. Cuando el paralelismo se obtiene al aplicar diferentes programas, algoritmos o tareas de forma simultánea a un mismo conjunto de datos, se dice que está orientado a tareas [15].

La clasificación basada en el lugar que se ejecutan los trabajos [164] los divide en: (a) programas paralelos, cuando su ejecución no sale de un sólo ordenador y utiliza sólo los recursos disponibles en una máquina, ya sea mediante la creación de hilos de ejecución o de procesos adicionales, y (b) programas distribuidos, cuando un programa es capaz de utilizar varios ordenadores conectados entre sí.

Si atendemos a la forma de comunicación entre los procesos que se ejecutan en paralelo, los paradigmas más frecuentes son:

Paralelismo compulsivo: Se trata del tipo de procesamiento paralelo más fácil de conseguir y más eficiente, aunque eso no significa que sea el más habitual. Se da cuando el problema que estás resolviendo permite partir los datos de entrada en conjuntos más pequeños y ejecutarlos por separado sin influir en el resultado final. O bien cuando diferentes programas se pueden aplicar a la vez de forma independiente al mismo conjunto de datos. De este modo, un programa de ejecución lineal que cumpla estos requisitos puede ejecutarse en paralelo con el número de equipos que deseemos. Los procesos (ordenadores) que trabajan en este tipo de ejecución lo hacen de forma totalmente independiente, ni siquiera tienen la información de que existe otro ordenador trabajando en lo mismo que él. La mayoría de las veces, este tipo de problemas se puede abordar sin necesidad de modificar los algoritmos implicados, desde una perspectiva de alto nivel que utiliza los propios sistemas de colas para lanzar bloques de trabajos con características comunes. Todos los sistemas de colas los facilitan, ya sea de forma nativa o mediante extensiones el concepto de *arrayjob*, que consiste en añadir de forma automática un número n de trabajos idénticos al sistema de colas en los que únicamente varía el valor de una variable de entorno. En función de esa variable de entorno, cada ejecución del pro-

grama podrá decidir el paquete de datos sobre el que desea realizar la ejecución. Estos trabajos, al ejecutarse de forma independiente y posiblemente simultánea en diferentes servidores de cálculo del supercomputador, proporcionan el paralelismo deseado.

MapReduce: Se utiliza mucho en el análisis de *BigData*, y comparte similitudes con el paralelismo compulsivo. La idea principal es básicamente la misma: una fase de mapeo (para preparar en paquetes los datos de entrada), la ejecución parcial de cada uno de los paquetes en diferentes CPU, y finalmente una fase de reducción (para reorganizar o combinar los resultados parciales obtenidos de cada paquete de entrada para obtener el resultado final). Este tipo de paralelismo requiere una fase de sincronización entre los ordenadores que están trabajando en conjunto antes de comenzar la etapa de reducción que en muchas ocasiones provoca retrasos de ejecución y hace disminuir el rendimiento de la paralelización. Al tratarse de un algoritmo básico, puede implementarse en cualquier lenguaje, aunque además existen librerías (*frameworks*) que facilitan su aplicación, como son el propio MapReduce que utiliza Google [107], o Hadoop [25, 6], que es una implementación de código abierto desarrollada en Java que inició Yahoo y que hoy día es parte del proyecto Apache. Implementaciones como Pydoop [111] proporcionan acceso a Hadoop desde Python, y comienzan a implantarse nuevas evoluciones como YARN [227] que persiguen un mayor rendimiento bajo grandes cargas realizando una gestión distribuida de los trabajos frente a la gestión centralizada que se realizaba anteriormente. MapReduce y Hadoop son excelentes candidatos para muchos análisis bioinformáticos [217].

Granjas de tareas (*Task-farm*): Puede considerarse una variante dinámica de MapReduce en la que un proceso principal coordina un conjunto de tareas independientes, asigna su ejecución a procesadores distintos (granja de procesadores) y realiza una recolección de los resultados a medida que los diferentes procesadores van finalizando su ejecución [50]. Las granjas de tareas suelen usar un reparto de tareas dinámico, mediante el cual se van asignando nuevas tareas a los procesadores que van quedando libres, un comportamiento que las hace especialmente útiles para entornos con equipos de rendimiento heterogéneos, y evita que se queden procesadores sin utilizar porque hayan procesado su conjunto de datos más rápidamente que otro.

Paso de mensajes con Message Passing Interface (MPI) [220]: Hay casos en los que ejecutar un programa en paralelo/distribuido no suele ser tan sencillo como trocear los datos de entrada y aplicar un paralelismo compulsivo o MapReduce. Los problemas más complejos requieren una programación más minuciosa donde se alternan fases de ejecución en paralelo con otras fases de sincronización o ejecución lineal. En esos casos se debe utilizar algún sistema de paso de mensajes entre los equipos involucrados para organizar los diferentes trabajos que están realizando cada uno [43]. Este tipo de ejecución se beneficia del uso de una red de baja latencia, como puede ser InfiniBand o Myrinet, para disminuir al mínimo el tiempo en que el proceso inicial está parado a la espera de una respuesta de otro servidor. Cuanto más rápida sea la comunicación, más eficiente será este tipo de ejecución y se malgastará menos capacidad de cálculo. Existen múltiples implementaciones de MPI, entre las que OpenMPI[71], MPICH[34] y MVAPICH[83] son las más utilizadas en supercomputación.

Paralelización automática: Consiste en procesar automáticamente el código de un programa de ejecución lineal y obtener cierto grado de paralelismo. Existen compiladores comerciales, como los de Intel o The Portland Group, que proporcionan la opción de paralelización automática, aunque lo más extendido es utilizar herramientas como OpenMP [42], que buscan zonas en el código fuente del programa que se puedan ejecutar en paralelo (como pueden ser bucles o condicionales múltiples que no tengan dependencias internas) y produce una versión modificada del código fuente que puede ser compilada con los compiladores tradicionales para obtener una versión paralelizada. El programador puede ayudar a la herramienta dándole pistas en forma de comentarios dentro del programa de qué partes él considera que son aptas para paralelizar. A veces da resultados espectaculares, y en otros casos no tanto [120], pero para bases de código que ya están escritas y no merece la pena reescribir desde cero de forma paralela constituye la mejor forma de obtener más rendimiento. Otro uso habitual es combinar OpenMP con MPI para diseñar un programa paralelo distribuido [175]. OpenMP se centra en programas escritos en C [98] y Fortran [13], pero la misma idea se puede encontrar en otros lenguages como R [89] de la mano de pR [119], Java [31] o Python [212] con Pydron [146].

2.5.2. Flujos de trabajo y automatización

Para realizar un trabajo de investigación, lo normal es que haya que utilizar varios algoritmos para obtener el resultado buscado. Es más, los distintos algoritmos suelen estar en paquetes de programas diferentes que habrá que hacer compatibles entre sí. Para ello, lo más recomendable es crear un flujo de trabajo *(workflow)* que automatice dicha combinación y compatibilización, que de otra forma sería muy tediosa. Un flujo de trabajo permitirá aplicar los mismos pasos a diferentes conjuntos de datos de entrada, sin tener que ejecutar manualmente cada uno de los programas [183]. De hecho, un flujo de trabajo no es más que un programa simplificado y resumido que hace llamadas a otros programas, ya sean instalados en la propia máquina o remotamente vía *web services*, lo que permite aprovechar todas las opciones de control de flujo que se utilizan en programación: puede ser totalmente lineal, incluir bifurcaciones basándose en los resultados previos (figura 2.4), realizar bucles de un determinado programa, realizar ejecuciones en paralelo, etc.

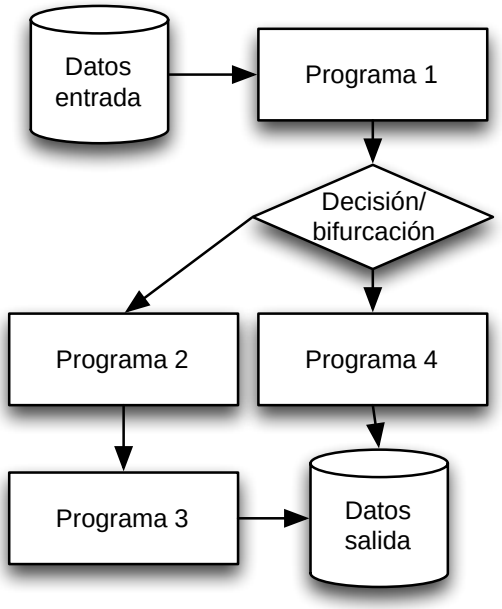


Figura 2.4: *Un flujo de trabajo consiste en encadenar ejecuciones de trabajos.*

Hay muchas opciones para crear flujos de trabajo, como por ejemplo:

- que el usuario haga un programa compi-

lado que llame a otros programas instalados en la máquina en el orden deseado. Es una práctica habitual de muchos programas que realizan gran parte de sus funciones realizando llamadas a otros programas externos, como es el caso de TopHat [99] —realiza llamadas a Bowtie [108], a scripts externos de Python [171] y a Samtools [117]— para realizar un flujo de trabajo complejo con el objetivo de mapear lecturas de RNA-Seq. La ganancia de rendimiento obtenida por usar un lenguaje compilado en la creación de flujos de trabajos no es significativa, puesto que el código necesario para lanzar y organizar la ejecución de programas externos sólo representa un pequeño porcentaje del tiempo total de ejecución del flujo de trabajo.

- usar un lenguaje de *scripting* como Ruby, Python, Bash, Perl, etc., para crear un programa que llame a otras aplicaciones [39]. Esta opción es mucho más flexible que usar un programa compilado, puesto que permite realizar modificaciones más rápidamente y es la opción utilizada en muchos programas como SeqTrim [58] o GATK y Dintel [142], así como en proyectos como SoleaDB [23] donde se utilizan *pipelines* para realizar ensamblajes e importaciones automáticas, o en MspJI-seq [84] para estudiar la metilación de genomas;

- usar herramientas como Conveyor [124] o AutoFlow [198] que facilitan construir flujos de trabajo en línea de comando;

- usar alguna herramienta que proporcione colecciones de servicios web preparados para ser enlazados, como son el MowServ [176] o BioExtract [132];

- utilizar herramientas visuales, algunas también basadas en repositorios de servicios web, como Taverna [223], Galaxy [70], Tavaxy [1], Yabi [88], Pipeline Pilot [191], Ergatis [162], Kepler [128], Triana [216], Dyscovery Net [184] o YAWL [225]; a pesar de la sencillez de diseño y ejecución que proporcionan gracias a sus elaboradas interfaces, se encuentran limitadas en cuanto a flexibilidad, o número de herramientas que pueden utilizar, y algunas de ellas exigen que su ejecución se realice en la misma máquina donde se ha diseñado el flujo y de una forma visual, por lo que no se integran bien con los sistemas de colas utilizados en supercomputación.

Para grandes cantidades de datos y uso en supercomputación es mucho más eficiente echar mano de *frameworks* que de herramientas visuales porque permitan el uso de la línea de comandos. Las últimas versiones de Taverna y Galaxy están dando pasos para facilitar su ejecución en sistemas de supercomputación y en nuestro grupo de investigación hemos desarrollado AutoFlow [198] basado en Ruby y Bash y sin interfaz gráfica específicamente adaptado para crear, ejecutar y compartir flujos de trabajo con paralelización.

Independientemente del sistema utilizado para crear el flujo de trabajo, éste consistirá en ejecutar un programa con los datos de entrada originales, comprobar los resultados para ver si son correctos y dichos resultados pasarlos a otro programa después de haber convertido su formato si es necesario. Esta cadena de programas puede hacerse el número de veces que se necesite hasta obtener el resultado final.

2.5.3. Utilidad del sistema de colas

En un sistema de supercomputación, el modo de trabajar difiere mucho de cómo se trabaja en un ordenador de sobremesa. Cuando se

utiliza un ordenador de escritorio, todo lo que ocurre aparece de una manera u otra en la pantalla. En supercomputación esto no es posible puesto que el usuario final no tiene acceso al ordenador donde se está ejecutando su trabajo, un equipo alojado en un CPD que puede estar muy lejos del usuario y que además ¡no tiene pantalla!.

En supercomputación se trabaja con un modelo desconectado como el resumido en la figura 2.5: el usuario accede con su nombre y contraseña a un servidor concreto (máquina de entrada) con un protocolo de comunicación remoto que suele ser Secure SHell (SSH) por motivos de seguridad. En esa máquina el usuario preparará su trabajo (o un flujo de trabajo), subirá al servidor los ficheros de entrada que quiere analizar desde su ordenador para almacenarlos en su espacio privado, y cuando lo tiene todo preparado, lo ejecuta. No tiene sentido que el trabajo se ejecute en directo porque se malgastarían los recursos de supercomputación, sino que el sistema de colas decidirá cuándo y dónde ejecutarlo. El usuario podrá desconectarse e incluso apagar su ordenador (que funciona solo como terminal) si quiere, porque el sistema de colas ya se encargará de gestionar su trabajo.

Como se puede intuir, todos los programas no son aptos para ejecutarlos en un supercomputador. Solo tendrán sentido los que tengan un modo de línea de comandos que permita indicarle al programa qué hacer sin necesidad de utilizar ventanas gráficas, y también debe ser capaz de proporcionar los resultados sin necesidad de mostrarlos en una pantalla, es decir, debe escribirlos en ficheros de salida, ya sea texto, imágenes, vídeo o cualquier otro formato que el usuario pueda inspeccionar posteriormente.

El sistema de colas no deja de ser un intermediario entre el usuario y los recursos de cómputo, y se encarga de gestionar que el uso de dichos recursos sea óptimo. Para ello requieren una puesta a punto continua por parte de los administradores de sistemas para adaptarse a las necesidades de los usuarios sin llegar a perjudicarlos ni malgastar los recursos.

2.5.4. Acceso al supercomputador de la UMA

Al supercomputador de la UMA se accede mediante una conexión SSH al servidor picasso.scbi.uma.es con el comando

```
ssh usuario@picasso.scbi.uma.es
```

desde un terminal de OS X o Linux, o con algún cliente de SSH de terceros como puede ser Putty [161] o WinSSH [27] si se utiliza Windows® (figura 2.6).

Al acceder a su cuenta, el usuario se encuentra en el directorio de inicio alojado en el almacenamiento compartido. Este espacio dispone de una cuota de disco limitada, y podrá usarse para almacenar los archivos que se consideren más importantes. Para trabajar con ficheros de datos que necesitan una cuota más amplia y con un acceso a disco más rápido, deberá cambiarse a su directorio de *SCRATCH* simplemente con

```
cd \$SCRATCH
```

Las cuotas de disco asignadas a los usuarios son variables y se establecen según las necesidades particulares. Para intercambiar ficheros con Picasso, se debe usar un programa que soporte el protocolo SFTP, ya sea por línea de comandos o alguno con interfaz gráfica como FileZilla [61] o CyberDuck [49] (figura 2.7).

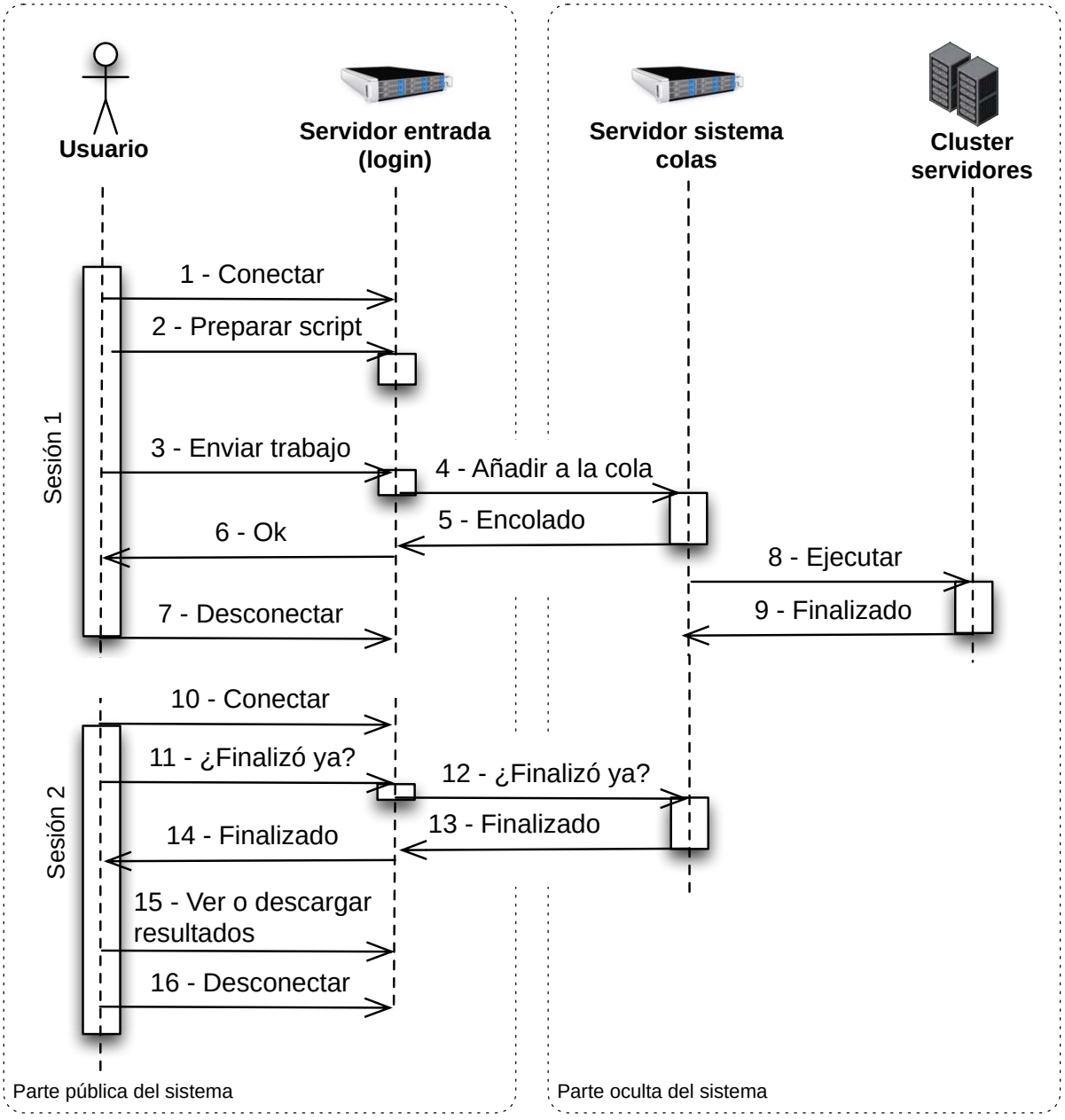


Figura 2.5: *Modelo de trabajo desconectado de un sistema de colas*

La preparación del trabajo consiste en crear un archivo de texto con un formato concreto para el intérprete Bash [35], con una serie de comentarios que indican al sistema de colas Slurm qué recursos necesita. A continuación se muestra un ejemplo concreto donde se indica que necesitaremos 10 procesadores y 2 Gb de memoria RAM durante 10 horas:

```
#!/usr/bin/env bash
```

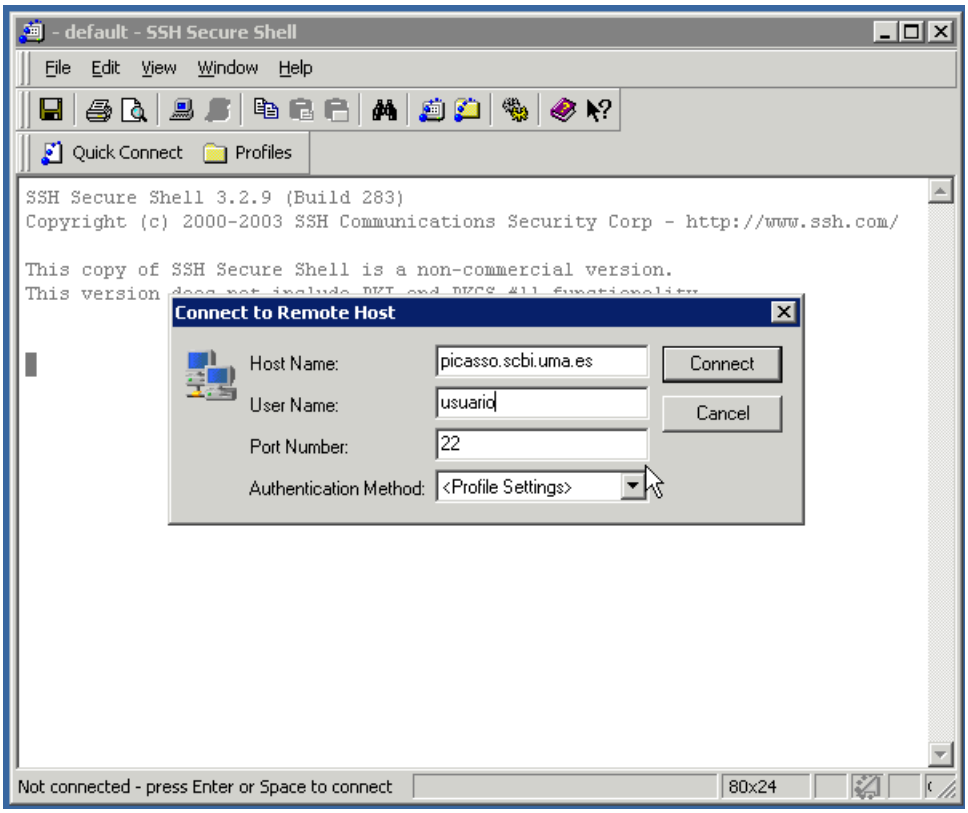


Figura 2.6: *Acceso por SSH a los ficheros almacenados en Picasso con el programa de terceros SSH Secure Shell*

```
# Number of desired cpus:
#SBATCH --cpus=10

# Amount of RAM needed for this job:
#SBATCH --mem=2gb

# The time the job will be running:
#SBATCH --time=10:00:00

# Load desired software
module load blast_plus/2.2.28+

# the program to execute with its
    parameters:
blastp -db swissprot
       -query prot.fasta
       -out test1.blast
       -num_threads 10
```

Este script se guardará en un fichero, por ejemplo `ejemplo.sh`, para poder enviarlo al sistema de colas con el comando

```
sbatch ejemplo.sh
```

Cuando el usuario estime que debería estar terminado o reciba una notificación de que ha finalizado, entrará de nuevo en la máquina de entrada, y preguntará al sistema de colas qué ha pasado con su trabajo. En Slurm se usa el comando

```
squeue
```

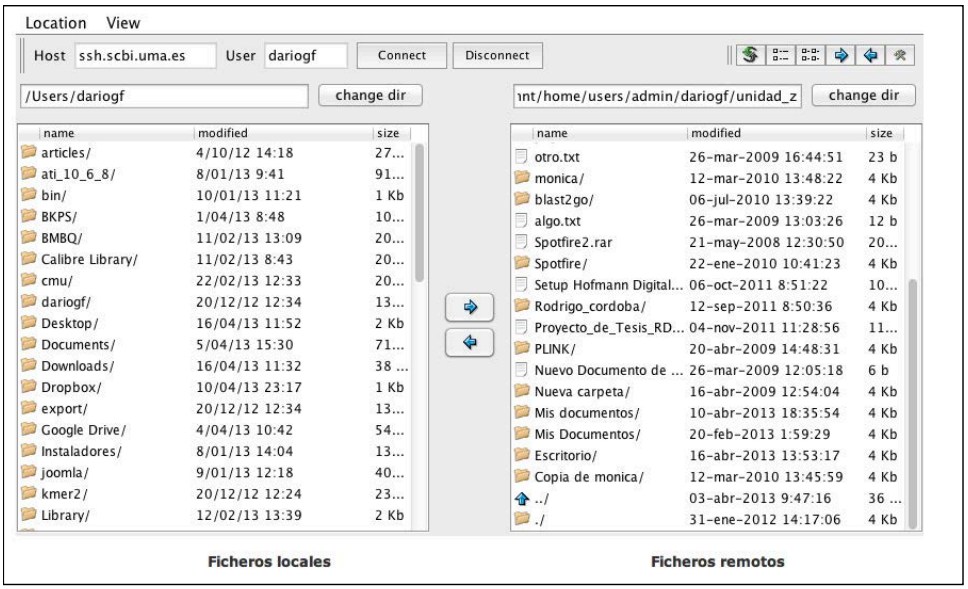


Figura 2.7: *Acceso remoto por SFTP a los ficheros almacenados en Picasso. De esta forma se pueden subir o bajar ficheros desde el ordenador personal al supercomputador.*

para un resultado similar al del siguiente listado:

```
picasso$> squeue -a

JOBID   NAME          USER    STATE CORES
=====================================
54327   ejemplo.sh    dario    [R]      4
54425   gaussian.sh   almu     [R]     64
```

Si el trabajo ha terminado, encontrará los ficheros de resultado en su almacenamiento privado, junto con los ficheros de salida que el usuario podría haber visto por la pantalla de su ordenador, o los posibles mensajes de los errores que se hayan producido.

Si tuviera que ejecutar un trabajo repetidas veces con diferentes datos de entrada, se puede echar mano de grupos de trabajos o *arrayjobs*. Gracias a algunas extensiones en Slurm, es tan sencillo como añadir una línea indicando el rango de trabajos a ejecutar y se lanzará un trabajo independiente por cada valor del rango. Cada trabajo recibirá dicho valor en una variable de entorno para saber qué paquete de datos tiene que utilizar. En el siguiente ejemplo de código se ilustra un *arrayjob* en el que se solicita el envío de 50 trabajos independientes que serán ejecutados por el sistema de colas a medida que vayan quedando huecos libres. Si existe sitio para ejecutarlos todos a la vez y no hay ninguna restricción en la configuración del sistema de colas, todos los trabajos podrían ejecutarse en paralelo:

```
#!/usr/bin/env bash

# Number of desired cpus:
#SBATCH --cpus=10

# Amount of RAM needed for this job:
#SBATCH --mem=2gb

# The time the job will be running:
#SBATCH --time=10:00:00

# MAKE AN ARRAY JOB, SLURM_ARRAYID will
      take values from 1 to 50
#SARRAY --range=1-50
```

```
# Load desired software
module load blast_plus/2.2.28+

# the program to execute with its
    parameters:
blastp -db swissprot
    -query prot${SLURM_ARRAYID}.fasta
    -out test${SLURM_ARRAYID}.blast
    -num_threads 10
```

El fichero que define el *arrayjob* se envía al sistema de colas con el comando

```
sarray fichero.sh
```

con lo que se añadirán n trabajos al sistema de colas que aparecerán como trabajos independientes cuando se consulte su estado con squeue:

```
picasso$> squeue -a

JOBID   NAME        USER    STATE CORES
=====================================
54327   array[1]    dario    [R]      4
54328   array[2]    dario    [R]      4
54329   array[3]    dario    [R]      4
54330   array[4]    dario    [R]      4
54331   array[5]    dario    [R]      4
54332   array[6]    dario    [R]      4
  .        .          .       .       .
  .        .          .       .       .
  .        .          .       .       .
54376   array[50]   dario    [R]      4
```

Capítulo 3

Ultrasecuenciación para principiantes

3.1. Del nucleótido a la secuencia

Aunque no sean la única fuente de enormes cantidades de datos, vamos a centrarnos en las técnicas de secuenciación de alto rendimiento, también denominadas de ultrasecuenciación o de secuenciación de nueva generación, secuenciación de nueva generación (del inglés *next generation sequencing*) (NGS). Se trata una tecnología que cada vez se usa más dentro y fuera de la Universidad de Málaga que plantea importantes retos a la hora del almacenamiento y procesamiento de los datos generados. Entre los posibles procesamientos, el alineamiento y el ensamblaje [123] son los que más recursos vienen exigiendo.

La secuenciación consiste en la obtención de la secuencia de nucleótidos que forma el genoma de una muestra biológica, o el transcriptoma de dicha muestra en el momento concreto de la secuenciación [100]. Existen muchas diferencias en las herramientas que se utilizan para procesar el genoma o el transcriptoma, y en la forma de tratar las muestras, pero para nosotros los informáticos, la abstracción del objetivo es simple: obtener la cadena de nucleótidos o aminoácidos que representan el genoma o transcriptoma. Puesto que a día de hoy no existe ninguna técnica de secuenciación en el mercado capaz de proporcionar el genoma completo en una sola cadena, se sigue el criterio de divide y vencerás (tan habitual en programación) [206], mediante el cual se obtiene un conjunto desordenado de cadenas de nucleótidos llamadas lecturas. El punto de inicio de cada lectura dentro del genoma es aleatorio, por lo que tendrán zonas comunes entre ellas que ayudarán a reconstruir la secuencia final mediante un procedimiento de ensamblaje [138] que consiste en la búsqueda de las regiones que solapan entre sí mediante criterios y heurísticos apropiados para ir formando una cadena de nucleótidos cada vez mayor hasta conseguir formar la cadena completa. En el caso de que la cadena completa sea imposible de formar, nos conformaremos con conseguir los fragmentos más grandes que sea posible. Los fragmentos obtenidos se denominan *contigs* para dar la idea de que son todos los nucleótidos contiguos que se han logrado yuxtaponer.

3.2. El diseño del experimento de laboratorio implica tener en mente el análisis bioinformático posterior

Antes de empezar a secuenciar, debemos determinar el objetivo del experimento y los métodos que vamos a utilizar para conseguirlo. Esto ayuda a ahorrar recursos, puesto que existen muchas formas de conseguir un resultado, pero no todas son igual de costosas ni sus resultados se analizarán de la misma manera ni, lo que es más importante, darán la misma información. Es necesario conocer bien las técnicas disponibles para poder hacer una decisión correcta que no complique posteriormente el procesamiento informático y la búsqueda de resultados y, sobre todo, que pueda dar respuesta al problema biológico que se quiere abordar.

Dado que el gasto de fungibles en cualquier experimento de ultrasecuenciación es muy significativo, manejándose cifras con tres o más ceros, la única manera de hacer un uso óptimo de los recursos pasa por diseñar correctamente el experimento desde un principio [156, 201]. De esta forma se evitará descubrir a mitad del recorrido (que pueden ser varios años) que la forma en que se está haciendo no es la correcta o que va a requerir un esfuerzo computacional o económico mucho mayor, o incluso que dicho esfuerzo sea un factor limitante. Por tanto, la elección del protocolo de secuenciación y las muestras será tan crítico como la elección de una buena arquitectura de programación para un desarrollo en informática, ya que será algo que lastrará el proyecto durante el resto de su vida [77, 67].

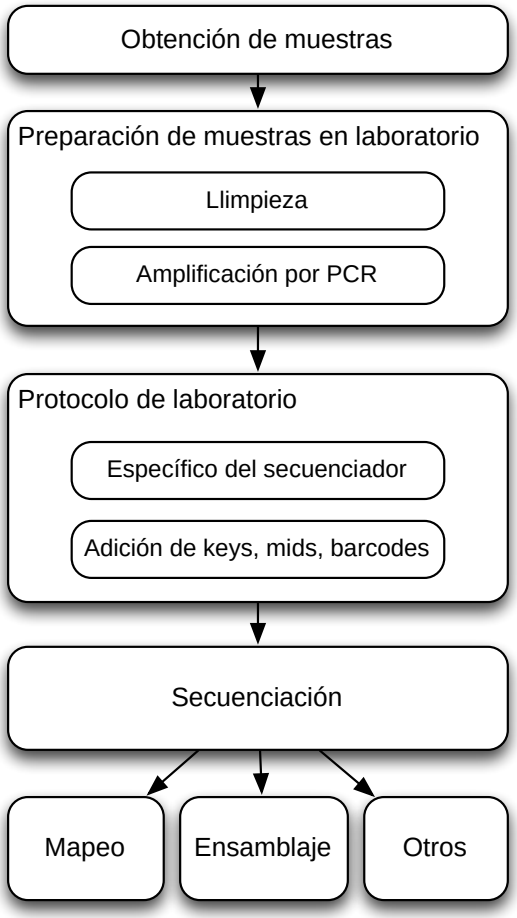


Figura 3.1: *Esquema de un experimento típico de ultrasecuenciación*

3.3. Las muestras a secuenciar

Una vez que se ha diseñado el experimento y se ha determinado que la secuenciación es el modo apropiado para abordarlo, necesitas muestras biológicas relacionadas con el objeto del estudio y su obtención es el primer paso de un experimento de secuenciación típico como el mostrado en la figura 3.1.

La obtención de las muestras puede ser tan rápido como tomar una muestra de un pacien-

te, o tan lento como esperar que una planta, cultivo bacteriano, o animal de laboratorio con una mutación concreta crezca. El único requisito es que la muestra contenga ácido desoxirribonucleico (ADN)/ácido ribonucleico (ARN), como pueden ser muestras de un tejido complejo (por ejemplo, muestras de sangre, saliva, tejidos extraídos por biopsia, vegetales, raíces, ...), orgánulos específicos o células concretas de un tejido (como células cancerígenas de un tejido afectado que pueden seleccionarse y cortarse con un microtomo) o muestras medioambientales (por ejemplo, tierra con microorganismos, en cuyo caso hablamos de experimentos de metagenómica en los que se secuencia la muestra completa para estudiar las poblaciones bacterianas que existen).

Se extraen los ácidos nucleicos de las muestras mediante diferentes técnicas de laboratorio [213] y la posterior amplificación del ADN resultante por reacción en cadena de la polimerasa (PCR). La amplificación por PCR es necesaria porque la cantidad de ADN que se obtiene es bastante escasa, y las máquinas de secuenciación necesitan una cantidad mínima para poder trabajar. Por eso se hace una PCR, que consiste en utilizar una serie de ciclos de temperatura y tiempo determinados junto con un entorno con los nutrientes necesarios, para simular las condiciones naturales bajo las que el ADN se multiplica en presencia de una ADN polimerasa [147]. Así se consigue amplificar la muestra de origen y obtener la cantidad necesaria para alimentar el secuenciador.

Una vez purificado el ADN, cada tipo de secuenciador exige un protocolo de preparación distinto, que incluyen el concatenar con nuestro ADN en bruto determinadas secuencias artificiales (figura 3.2) que se detallan a continuación:

- Clave (*key*): se trata de una secuencia artificial conocida que, en determinado tipo de secuenciadores, se añade siempre al inicio de cada fragmento a secuenciar con el objetivo de facilitar el reconocimiento posterior del inicio de la secuencia y comprobar que la reacción transcurrió correctamente.
- Adaptadores: secuencias fijas que se añaden siempre a uno o ambos extremos del ADN a secuenciar para poder amplificarlo por PCR o directamente secuenciarlo.
- Identificadores moleculares (identificadores multiplexados (MID), del inglés *molecular identifiers*) o códigos de barras (*barcodes*): secuencias artificiales conocidas que solo se añadirán a las muestras de diferentes experimentos cuando se van secuenciar a la vez en una misma reacción. Gracias a los MID se podrá posteriormente diferenciar y separar las lecturas por su correspondiente experimento. Por ejemplo podemos añadir el MID1 a muestras de hojas y el MID2 a muestras de raíz, realizar todo el protocolo de secuenciación y, al final, separar las lecturas de hojas y de raíz con herramientas bioinformáticas (figura 3.2).
- Conectores (*linkers*): secuencias artificiales conocidas que se añaden solo en los experimentos de secuenciación de lecturas pareadas para dividir cada lectura en la parte derecha y la parte izquierda de la misma y conocer la posición relativa de cada componente del pareado (figura 3.2).

3.4. Plataformas disponibles

Las muestras obtenidas de la fase anterior deben ser secuenciadas con alguna de las

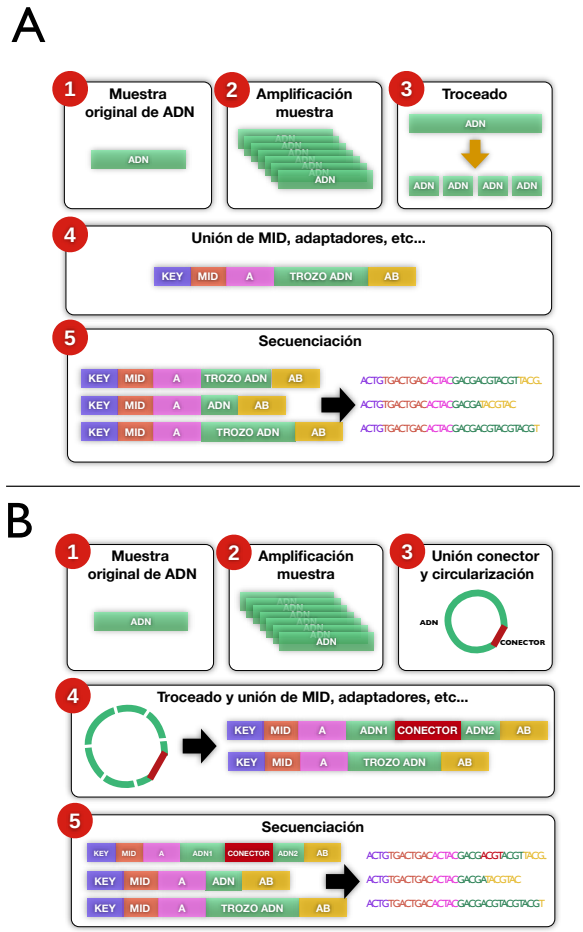


Figura 3.2: *Fases de las dos principales estrategias de secuenciación.* A*) Adición de secuencias artificiales durante el protocolo de secuenciación 454/FLX+,* B*) Adición de secuencias artificiales durante el protocolo de secuenciación 454/FLX+ para lecturas pareadas*

máquinas existentes en el mercado. Hasta ahora, la ultrasecuenciación se ha repartido entre muy pocos competidores que utilizan métodos completamente diferentes de secuenciación masiva. El primero que surgió está hoy en manos de Roche, con su secuenciador 454/FLX Titanium+ basado en la pirosecuenciación [230]; proporciona aproximadamente 1 millón de lecturas de hasta 700-1.000 nt de longitud por cada lectura y por ahora es el único que proporciona lecturas tan largas. Lamentablemente, Roche ha anunciado que en 2016 dejará de darle soporte para volcar sus esfuerzos en los métodos de secuenciación de molécula única. La tecnología Ion Torrent^{TM} de Life Technologies, desarrollada por los mismos que inventaron la pirosecuenciación 454, usa el mismo tipo de reacción de secuenciación, pero acoplada a la nanodetección de cambios de pH en un chip, en lugar de emisión de luz (que es lo que usa la pirosecuenciación). Está enfocada a proporcionar tecnología de ultrasecuenciación a pequeña escala con dos plataformas distintas: Ion Proton^{TM} que en una reacción produce entre 60 y 80 millones de lecturas de 200 nt de longitud en un volumen de hasta 14 Gnt [30], e Ion PGM^{TM} que proporciona desde 400.000 hasta 5,5 millones de lecturas de entre 200 o 400 nucleótido (nt) de longitud con un volumen total de 2 Gnt [33].

Las otras dos tecnologías que se asocian a la ultrasecuenciación se dice que son de lecturas pequeñas, y están representados por las plataformas de Illumina y SOLiD^{TM}. La tecnología de Illumina, adquirida a Solexa, se basa en la reacción de secuenciación reversible [137, 234], que ellos llaman secuenciación por síntesis (Sequencing by Synthesis (SBS)), y está implementada en las plataformas MiSeq, MiSeqDx, MiSeq FGx, NextSeq 500, HiSeq 2500, HiSeq 3000, HiSeq 4000. HiSeq X Five y HiSeq X Ten [141]. Dentro de la serie HiSeq nos encontramos modelos con capacidad para proporcionar entre 4.000 millones y 6.000 millones de lecturas, pero de una longitud (de 125 a 150 nt) que es mucho menor que la tecnología de Roche. El menor de ellos, el HiSeq2500, es capaz de proporcionar 1.000 Gnt en una reacción, mientras que el más potente, el HiSeq X Ten, es capaz de producir hasta 1.800 Gnt por reacción en sólo 3 días [`http://www.illumina.com/systems/sequencing.html`]. Es obvio que este sistema está ya generando problemas de al-

macenamiento y procesamiento. La plataforma SOLiD^{TM} (Sequencing by Oligonucleotide Ligation and Detection), original de Aplied Biosystems y recientemente fusionada con Life Technologies, dispone actualmente de dos secuenciadores, el 5500xl System, que proporciona hasta 30 Gnt con lecturas de 75 nt de longitud, y el 5500 System, que sólo se diferencia por tener la mitad de capacidad de secuenciación y por lo tanto proporciona la mitad de datos que su hermano mayor [163].

Todas las tecnologías anteriores forman parte de lo que se conoce por tecnologías NGS *(Next Generation Sequencing)* o de segunda generación. Pero ya han emergido nuevas tecnologías de secuenciación que algunos denominan de tercera generación, o sistemas de molécula única (Single Molecule Sequencing (SMS)) [194], que se caracterizan por eliminar la amplificación de las muestras de ADN a secuenciar, limitándose a secuenciar una única molécula de ADN de la propia muestra. Como resultado, se obtienen secuencias mucho más largas y más fáciles de ensamblar, aunque la frecuencia de errores que proporcionan aun es demasiado elevada. La tecnología de secuenciación de molécula única por fluorescencia® (Single Molecule Fluorescent Sequencing (SMFS)) de la empresa Helicos [200] fue el primer sistema en usar SMS, pero manteniendo todavía los sistemas de reacción/parada/lavado/escaneado utilizado por los sistemas de segunda generación. La necesidad de parar la reacción hacía que la reacción de secuenciación completa tardara mucho tiempo, y que la longitud de las lecturas obtenidas fuera demasiado corta, por lo que el mercado se decantó por la plataforma Single Molecule Real-Time sequencing (SMRT) de Pacific Biosciences [181]. Pacific Biosciences ofrece un sistema SMS capaz de determinar en tiempo real las bases que se van incorporando a una única molécula de ADN polimerasa con un consumo minimo de reactivos y sin necesidad de parar la reacción con cada incorporación de nucleótidos. Como consecuencia, el tiempo de reacción es mucho menor. El secuenciador PacBio RS II produce 400 Mnt de lecturas muy largas (desde 8500 nt hasta 40.000 nt), pero lastradas todavía por una frecuencia de errores demasiado alta, alrededor del 13 % (el resto de los sistemas no supera de hecho el 1 % [173]) .

Basados también en la SMS, pero con otra técnica de detección diferente, nos encontramos con los sistemas basados en nanoporos [134] que funcionan haciendo pasar una molécula de ADN nucleótido a nucleótido a través de un nanoporo que, por detección óptica, biológica o por fluctuaciones en campos eléctricos, es capaz de determinar el nucleótido que está pasando en ese momento [81]. Esta nueva generación está liderada por el Oxford Nanopore [57] que utiliza tres moléculas biológicas que forman un poro que hace de detector. Estos poros se excitan con voltaje y cambios de la concentración salina del medio en que se encuentran, y en ese estado son capaces de capturar el punto de ruptura del nucleótido a causa de la corriente que está pasando por él. Puesto que cada nucleótido tiene un punto de ruptura distinto, se puede determinar la base que le corresponde.

La conclusión que debemos extraer de este apartado es que el volumen y el tipo de datos varía continuamente a un ritmo que harán que estén obsoletos incluso cuando se publique esta tesis. La idea principal que pretendo aquí aportar es que se puede optar por técnicas que producen lecturas más largas o muchas lecturas más cortas, y cada elección tiene su aplicación concreta (figura 3.3). *A priori* parece sensato pensar que lo ideal sería elegir la que nos proporciona la mayor cantidad de datos al menor coste, pero veremos que es aún más importante conocer cuál es la más apropiada para

cada estrategia experimental, y que esta elección influirá directamente en el procesamiento bioinformático de los resultados.

3.5. Aplicaciones

Según la tecnología y las necesidades experimentales, las aplicaciones de la ultrasecuenciación se pueden dividir en tres grandes grupos.

Secuenciación *de novo*: una secuenciación *de novo* tiene sentido cuando se pretende descubrir el genoma o transcriptoma de un organismo sobre el que no existe este tipo de información. En estos casos obtendremos mejores resultados y de forma más rápida si se emplean dos técnicas de secuenciación diferentes, ya que cada tecnología suplirá sus defectos con la otra [236, 12]. Por ejemplo las secuencias obtenidas con el 454 tienen una mayor longitud, lo que facilita el ensamblaje inicial y produce un conjunto desordenado de *contigs* de semilla. Al complementar esta información con secuencias pareadas (ya sean de 454 o Illumina), conseguiremos determinar el orden lineal en que se sitúan dichos *contigs* que conforman el boceto del genoma y, posteriormente, podemos ampliar su cobertura y rellenar los huecos. Utilizar las secuencias de un HiSeq para el ensamblaje inicial en estos casos puede ser perjudicial debido a que poseen una longitud menor que dificulta el proceso de ensamblaje cuando existen grandes zonas repetitivas en el genoma que se está estudiando, aunque los genomas relativamente simples puedan ser ensamblados sin mucha dificultad con secuencias cortas [166].

Resecuenciación: se utiliza para secuenciar organismos del que ya disponemos de un genoma o transcriptoma fiable, con el objetivo de estudiar cambios entre individuos concretos, como SNP o microvariaciones [157, 179], realizar análisis de genómica funcional para determinar cómo interactúan los genes y las proteínas, estudiar las modificaciones evolutivas que ha sufrido una especie [73], realización de genotipado para diferenciar organismos biológicos [148], o incluso estudios de genómica de poblaciones que proporcionan información sobre cómo afectan las variaciones genéticas a grupos de población [152]. Por norma general, la resecuenciación es mucho más económica que una secuenciación *de novo* y proporciona un reto computacional bastante menor [36]. Si bien este tipo de secuenciación podría hacerse sin problemas con cualquiera de las tecnologías descritas en la Sección 3.4, las secuenciaciones de HiSeq nos proporcionan mayor cantidad de datos a menor coste.

Expresión génica (RNA-seq): los avances en secuenciación permiten aplicarla a áreas que antes sería impensable. Por ejemplo, antes de aparecer la ultrasecuenciación, para estudiar los genes que se expresaban ante un estímulo concreto se utilizaban micromatrices [113, 51, 69], una técnica muy delicada que permitía conocer la expresión simultánea de un conjunto finito de posibles genes candidatos sobre los que se tenían indicios de que podrían expresarse. Hoy día la ultrasecuenciación permite de una forma mucho más inmediata y con menos manipulación manual secuenciar las muestras sometidas a un estímulo concreto y después cuantificar qué genes (de forma global, sin necesidad de seleccionar candidatos) se estaban expresando más en el momento de la secuenciación en función de la cantidad de lecturas que se obtienen de cada transcrito. El concepto general parte de que si no se realiza ninguna manipulación de la muestra para evitarlo, la probabilidad de que las lecturas pertenecientes a cada gen aparezcan es la misma

	Segunda generación						Tercera generación	
Fabricante	**Roche 454**	**Illumina**		**SOLiD**	**Life Technologies**		**Pacific Biosciences**	**Oxford Nanopore**
Secuenciador	454 GS FLX	HiSeq 2500	HiSeq X Ten	5500 XL	Ion DPM	Ion Proton	PacBio RS II	MinION
Tecnología	Pirosecuenciación	Secuenciación por síntesis (SBS)		Ligación, codificación de 2 bases	Secuenciación por síntesis en chip		Molécula única en tiempo real - SMRT	Molécula única (SMS) nanoporo
Nº lecturas	1 M	4000 M	6000 M	*400 M*	6 M	80 M	0.05 M (50 K)	0.06 G (60 K)
Longitud de las lecturas	700 - 1000 nt	125 nt	150 nt	75 nt	400 nt	200 nt	8500 nt (up to 40000 nt)	5000 nt (up to 90000 nt)
Total nucleótidos	1 Gnt	1000 Gnt	1800 Gnt	30 Gnt	2 Gnt	14 Gnt	0.4 Gnt (400 Mnt)	0.2 Gnt (200 Mnt)
Duración de la reacción	24 horas	6 días	3 días	2 - 7 días	5 horas	5 horas	Minutos	Desde minutos a horas

Figura 3.3: *Resumen de las técnicas de secuenciación de segunda y tercera generación (resumido de [125, 174, 194, 126]).*

en todos los casos. Por ese motivo, si hacemos una secuenciación y las secuencias pertenecientes a un gen aparecen relativamente más veces, es porque ese gen está más expresado. Respecto a la técnica de secuenciación a utilizar, nos encontramos en un caso similar al caso de la resecuenciación, donde lo importante es la cantidad de secuencias, por lo que a mayor cantidad de datos, más fiable será el resultado. Tiene por tanto sentido utilizar técnicas que produzcan gran cantidad de secuencias pequeñas como pueden ser HiSeq o SOLiD^{TM}.

3.6. Análisis bioinformáticos

Una vez obtenidas las lecturas, todo el análisis posterior recae en la bioinformática. Con respecto a las secuencias, lo podemos dividir en las siguientes grandes etapas: preprocesamiento, ensamblaje y anotación (o mapeo), y análisis estadístico o funcional. A continuación se describen con más detalle.

3.6.1. Preprocesamiento

Todas las lecturas de ultrasecuenciación necesitan un preprocesamiento por más que los fabricantes digan que lo que proporcionan ya es adecuado para su uso. Gracias al preprocesamiento, se incrementará la calidad de los resultados, lo que facilitará el posterior tratamiento de los mismos. En la etapa de preparación de muestras se añadieron diferentes elementos artificiales (figura 3.2) que habrán de desaparecer de la parte útil de la secuencia para no proporcionar resultados indeseables o artefactuales. En esta etapa también hay que eliminar cualquier secuencia incluida de forma no intencionada, como pueden ser contaminantes de otros organismos que no nos interesan para el experimento [58].

3.6.2. Ensamblaje

El ensamblaje consiste en obtener cadenas de ADN/ARN relativamente grandes (*contigs*) que idealmente estarían ordenadas (gracias a las secuencias pareadas) en forma de *scaffolds*, a partir de los pequeños trozos de secuencias que hemos obtenido de uno o varios experimentos de secuenciación (figura 3.4; [149]). Existen varios tipos de algoritmos para realizar este proceso, pero básicamente consisten en la comparación de todas las secuencias obtenidas en el experimento y la creación de diferentes

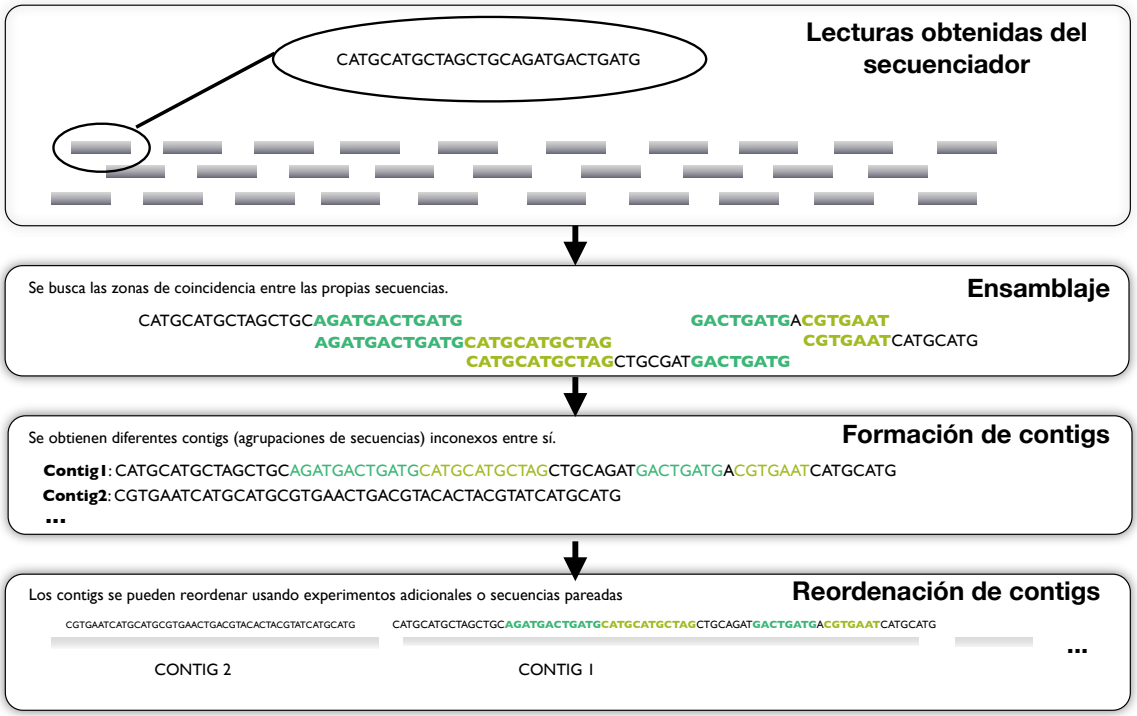


Figura 3.4: *Distintas etapas del ensamblaje, desde las lecturas originales (arriba) a los* contigs *ordenados en* scaffolds *(abajo).*

grafos o tablas de relaciones ponderadas para determinar qué secuencias están solapadas con otras y en qué medida lo están. Siguiendo las relaciones de solapamiento y algunos heurísticos para acelerar las decisiones, se consigue formar una cadena mayor. Como es de suponer, los algoritmos más antiguos no están preparados para utilizar múltiples CPU y necesitan mantener todas las secuencias en memoria durante la fase de cálculo de los solapamientos. Esto implica que deben utilizarse máquinas con mucha memoria RAM (incluso del orden de 1 o 2 terabyte, unidad de almacenamiento equivalente a 1000 o 1024 gygabytes (TB)) para realizar ensamblajes *de novo* complejos. El resultado de este paso será un conjunto de *contigs* que se podrá usar para otros análisis comparativos, e incluso para realizar ensamblajes incrementales o mapeos de experimentos de resecuenciación posteriores.

El problema del ensambaje está lejos de solucionarse, al menos hasta que la tecnología de secuenciación SMRT baje del 13 % de error que presenta actualmente [173]. Además, no todos los organismos se ensamblan con la misma facilidad [45] ni todos los ensambladores funcionan igual en todos los organismos [55, 32].

Para ensamblar dos lecturas hay que detectar las regiones en las que solapan, que deben tener una longitud mínima para ser fiables, siempre teniendo en cuenta el tamaño de las lecturas que tenemos disponibles. Si son lecturas de 50 nt de longitud no sería razonable exigir un solapamiento de 40 nt, mientras que ese solapamiento sí es perfectamente válido para ensamblar lecturas largas. Por la forma de detectar estas regiones solapantes, podemos encontrar diferentes tipos de algoritmos [140]:

Algoritmos voraces En los algoritmos de ensamblaje voraces (figura 3.5) se parte de una lectura inicial y después se selecciona el mejor candidato entre el resto de lecturas para extender la cadena. El mejor candidato será aquella lectura que solape más nucleótidos con la lectura inicial. El proceso se repite de forma iterativa tomando como secuencia inicial el resultado de la iteración anterior, hasta que ya no queden más candidatos posibles para extender la cadena. Aunque no se construya una estructura de grafos propiamente dicha, sí que se está recorriendo un grafo virtual al realizar en cada paso la búsqueda del mejor candidato. Este sistema de ensamblaje es el más primitivo de todos y presenta claros problemas con las zonas repetitivas, que quedarían reducidas a una sola repetición. Ejemplos de algoritmos que encajan en esta categoría son SSAKE [229], VCAKE [91] o SHARCGS [54].

Algoritmos Overlap/Layout/Consensus (OLC) La aproximación OLC (figura 3.6) se realiza en tres etapas (*Overlap/Layout/Consensus*): *a*) En la primera etapa se buscan las regiones solapantes entre secuencias, para lo cual se realiza una comparación de cada secuencia contra todas las demás. Esta comparación se realiza en funcion de los porcentajes de identidad y del tamaño mínimo del solapamiento junto con un parámetro muy importante para los ensamblajes denominado tamaño de k-mero, que normalmente elige el usuario, pero otras veces se encuentra fijado en el código fuente del algoritmo. El tamaño de k-mero lo podemos considerar como la unidad básica de comparación que se utilizará en la búsqueda de solapamientos [44]. Al utilizar un k-mero de longitud n, se calculan todas las posibles combinaciones de nucleótidos de dicha longitud, y después se procesan todas las lecturas determinando los k-meros que contienen en común entre ellas. Esta abstracción acelera los procesos de ensamblaje y permite un ahorro considerable de memoria. *b*) En la segunda etapa se procesa el grafo de las relaciones establecidas entre las lecturas para establecer la forma más probable del ensamblaje atendiendo al número de k-meros que comparten los posibles solapamientos. *c*) Finalmente se alinean las secuencias que solapan entre sí para elegir las secuencias consenso que serán los *contigs* resultantes del ensamblaje. Algunos ensambladores que utilizan esta aproximación son CAP3 [85], PCAP [86], Celera Assembler/CABOG [139], Arachne [21] o Newbler [106].

Algoritmos basados en grafos de De Bruijn Un grafo de De Bruijn se utiliza para representar solapamientos entre cadenas de símbolos, lo cual encaja perfectamente con el problema del ensamblaje [46]. Los ensambladores basados en grafos de De Bruijn suelen utilizarse para lecturas cortas. En la figura 3.7 se representa de forma simplificada cómo se añade cada lectura al grafo, primero se divide en oligomeros (k-meros) de un tamaño k elegido por el usuario, y de cada k-mero se extraen los extremos izquierdo y derecho de tamaño $k-1$, extremos que se almacenan como nuevos nodos en el grafo de *De Bruijn* unidos entre sí con un arco dirigido desde el extremo izquierdo al derecho. Si los nodos ya existen en

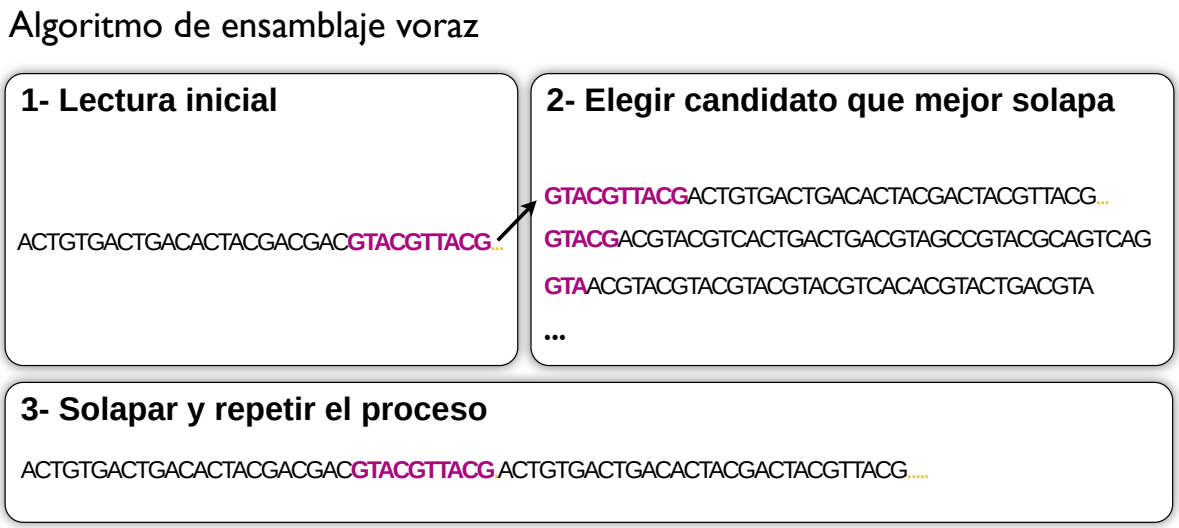


Figura 3.5: *Procedimiento de ensamblaje con un algoritmo voraz*

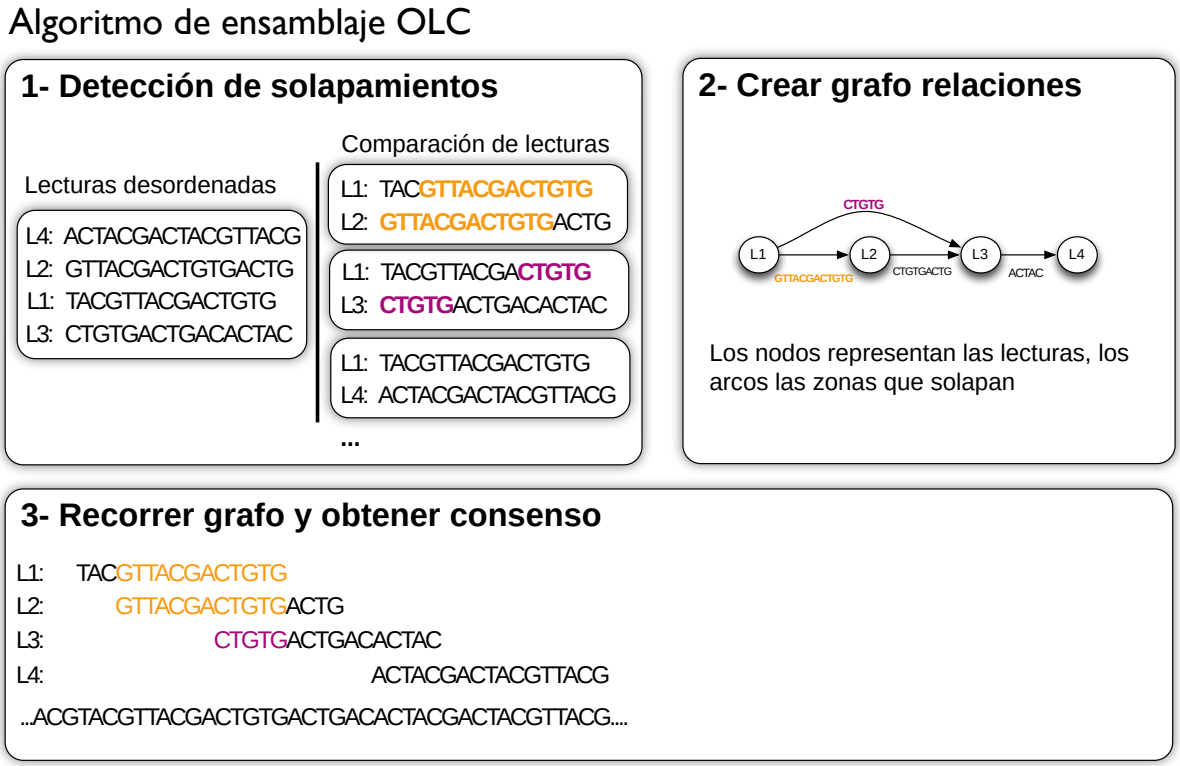


Figura 3.6: *Procedimiento de ensamblaje con un algoritmo OLC*

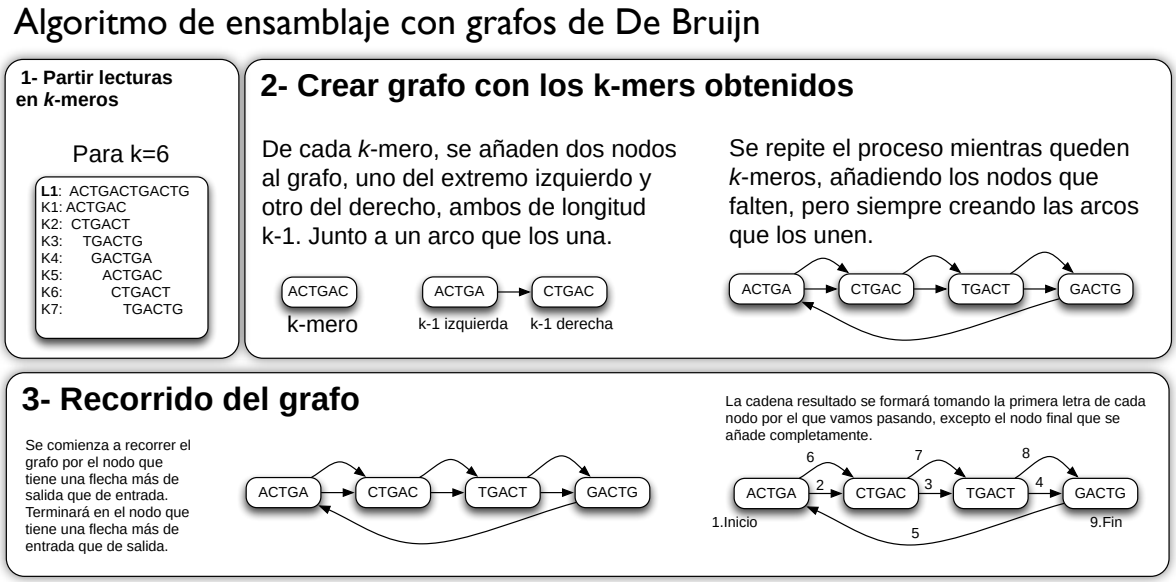


Figura 3.7: *Procedimiento de ensamblaje con grafos de De Bruijn*

el grafo, únicamente se añade el arco que los une. La secuencia ensamblada se obtiene recorriendo el grafo desde el extremo inicial hasta el final, concatenando la primera base del k-mero almacenado en cada uno de los nodos recorridos, excepto el k-mero del nodo final que se une por completo al resultado. Al formar el grafo mediante la adición de las lecturas de forma iterativa, se evita hacer comparaciones de todas las secuencias entre sí. Además es habitual que las secuencias no se almacenen en el grafo y que las repeticiones sean anotadas con un contador numérico en vez de utilizar arcos adicionales, estrategias que ayudan a ahorrar memoria durante la ejecución del algoritmo [122]. Ensambladores que usan esta técnica son Velvet [240], ALLPATHS2 [133], Euler-SR [40], ABySS [203], SOAPdenovo [130] o PASHA [127].

3.6.3. Mapeo

Realizar un mapeo es mucho más simple que un ensamblaje (figura 3.8; [185]). En este caso se dispone de un genoma o transcriptoma de referencia ya ensamblado que ayuda a realizar el alineamiento. Así se consigue saber a qué zona de ese genoma corresponde cada una de las secuencias que hemos obtenido del experimento. Para ello se compara cada secuencia con el genoma de referencia para así determinar a qué zona del genoma de referencia se parece más (de forma ideal será una coincidencia exacta, aunque pueden existir pequeñas variaciones, una veces debidas a errores de secuenciación y otras a polimorfismos). Puesto que la comparación es uno a uno, sólo es necesario mantener un conjunto mínimo de secuencias en memoria (la referencia y la secuencia activa), y el consumo de recursos por lo tanto es mucho menor. En cuanto al uso de procesadores, este tipo de problemas es fácilmente paralelizable ya que las comprobaciones se pueden ha-

cer varias a la vez sin que el resultado de una comparación tenga influencia alguna sobre las demás.

Cada tecnología de secuenciación dispone de sus propios programas de mapeo comerciales, como son Eland de Illumina, Corona de SOLiD™, o Reference Mapper de 454/Roche, pero existe una gran cantidad de programas de código abierto que suponen alternativas viables. Igual que ocurría con los algoritmos para ensamblaje, existen diferentes aproximaciones en función de la forma en que se realizan los alineamientos de las lecturas con los genomas o transcriptomas de referencia:

Indexación de las lecturas Todas las lecturas se indizan en un *hash* contra el cual se alinea el genoma o transcriptoma de referencia. Ejemplos de programas de mapeo que encajan en esta categoría son MAQ [118], RMAP [205] o SHRiMP [186].

Indexación del genoma Se crea el *hash* al indizar el genoma o transcriptoma usado como referencia, justo al revés que en la primera aproximación, y son las lecturas las que se alinean de forma iterativa contra el *hash* del genoma. PASS [38] y GASSST [180] son algoritmos que siguen esta estrategia.

Búsqueda inversa Utiliza la transformada de Burrows–Wheeler [189] para encontrar todas las coincidencias exactas entre las lecturas y la referencia, que posteriormente se complementa con un algoritmo de búsqueda inversa (*backtracking*) para encontrar coincidencias, aunque exista una pequeña cantidad de errores en las lecturas. Este tipo de algoritmos engloba herramientas como Bowtie [108] [109] y SOAPaligner [121].

Híbridos Algoritmos como GenomeMapper [196] o Stampy [129] combinan diferentes estrategias de búsqueda junto a modelos estadísticos para mejorar la precisión y velocidad de los algoritmos tracionales. GenomeMapper incluso permite el mapeo simultáneo con varias referencias, por lo que el tiempo de mapeo se reduce considerablemente. Algunos algoritmos, como SOAP3-dp [131], incorporan el uso de GPU para acelerar los resultados de búsquedas.

En esta etapa podemos obtener diferentes resultados: un archivo de *contigs* o *scaffolds* ya ensamblados, o datos de mapeo que pueden servir para cuantificar expresión o determinar variaciones (SNP o mutaciones) de la muestra procesada respecto al genoma de referencia. Los resultados de mapeo tradicionalmente se proporcionaban en un fichero de texto tabulado en un formato llamado Sequence Alignment Map (SAM) [190], pero a causa del incremento en el volumen de datos producido por las técnicas de ultrasecuenciación se hizo necesario el desarrollo de su equivalente binario Binary Alignment Map (BAM) que ocupa menos espacio. Para manipular ficheros en ambos formatos existen las librerías SAMtools [117] y BAMtools [16].

3.6.4. Anotación y análisis

Los ensamblajes, sean de genomas o transcriptomas, necesitan una anotación [208], que consiste en comparar los *contigs* obtenidos con otras secuencias, principalmente de bases de datos públicas, para establecer si el parecido entre la secuencia nueva y la conocida es suficiente para permitirnos asignarle las mismas funciones que tiene la secuencia conocida [187]. Las anotaciones también son bastante costosas computacionalmente hablando, ya que las bases de datos con las que se suelen comparar

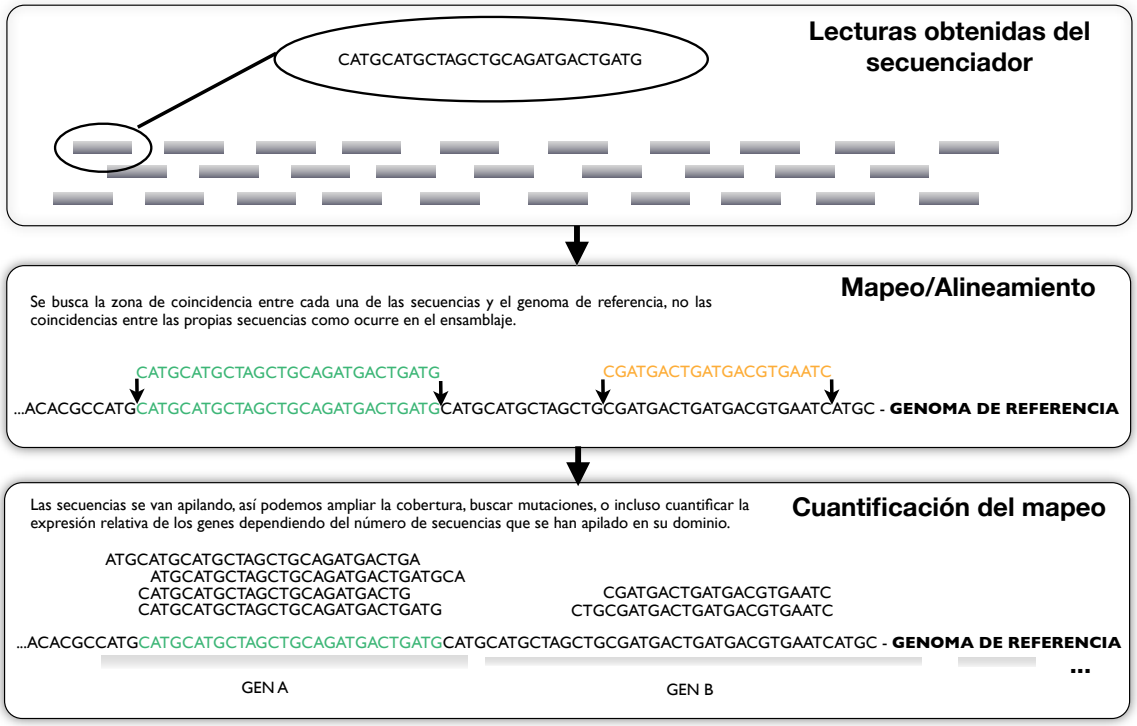


Figura 3.8: *Mapeo de lecturas sobre una secuencia de referencia*

son bastante grandes y dependiendo del grado de exactitud que se le exija a la comparación, podemos obtener más o menos resultados.

Podemos agrupar el tipo de anotaciones en dos grandes grupos (figura 3.9). Por un lado, las anotaciones más elementales que se realizan con programas tipo BLAST+ [37], hmmer [93] o exonerate [204], que realizan comparaciones de secuencias contra bases de datos conocidas [231] como son GenBank [22] o RefSeq [168] de NCBI, o UniProtKB [10, 14]. Estas anotaciones dan una idea inicial en lenguaje natural a los investigadores sobre qué contienen sus secuencias, pero el lenguaje humano es muy difícil de interpretar por parte de los programas de análisis (muchas de estas anotaciones están escritas en lenguaje natural y cada persona las escribe de la forma que le parece más correcta).

Por otro lado, se pueden realizar anotaciones que siguen una ontología determinada [193] como pueden ser Gene Ontology (GO) [75] o InterPro [145], o determinar la ruta metabólica para obtener los mapas Kyoto Encyclopedia of Genes and Genomes (KEGG) [95] y códigos Enzyme Commission (EC) de la actividad enzimática asociada. Con este tipo de anotaciones tenemos la ventaja de que cada anota-

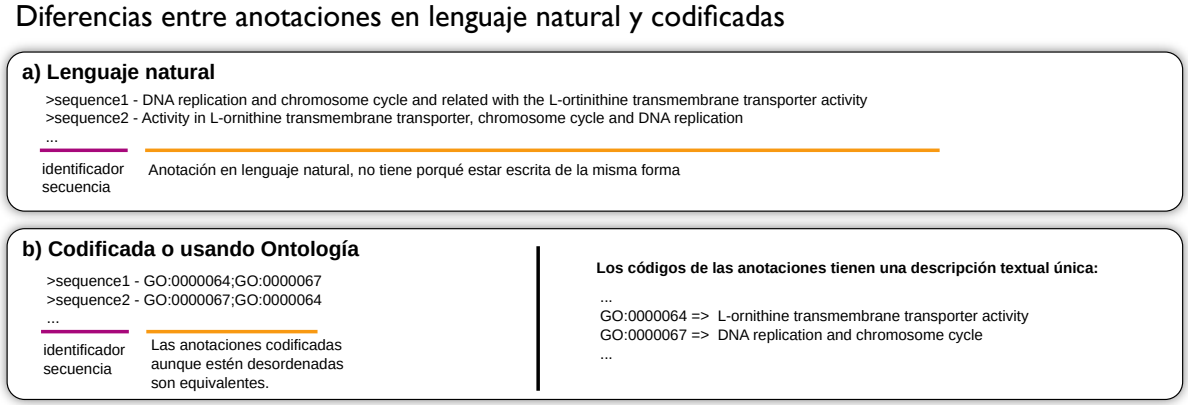


Figura 3.9: *Representación de las diferencias entre una anotación textual y anotaciones que siguen una codificación u ontología.*

ción realizada tiene un código único en vez de una descripción de texto, y por lo tanto son útiles para posteriormente realizar un análisis informático de los resultados. Los programas habituales para anotar con texto natural y con códigos son Maker [79], Sma3s [144], Blast2GO [48] y Autofact [104].

De esta etapa idealmente obtendríamos un listado de contigs con una serie de anotaciones que nos indicarán para qué sirven, incluso la proteína que codifican, si hemos conseguido secuenciar el gen completo o sólo una parte, etc. Lo más normal es que se anoten una serie de *contigs* y otros queden sin anotaciones, por lo que si éstos se consideran de interés sería necesario realizar estudios y experimentos adicionales para determinar en qué están implicados [23].

Con los *contigs* ya anotados se puede empezar a interpretar y hacer análisis de los resultados, ya sean análisis estadísticos o búsquedas en bases de datos especializadas (David, Cytoscape, String, Ingenuity Pathways, Gene Investigator, etc.) que pueden añadir información más detallada, mostrar las rutas metabólicas en las que están implicados dichos contigs (los genes representados por ellos), o incluso aportarte bibliografía relacionada con los mismos.

3.7. Difusión de resultados

Una labor muy importante dentro de la investigación (y que también está presente en las solicitudes de financiación de proyectos de investigación) es la presentación y difusión de los resultados finales, así como de todos los datos que se han obtenido durante el estudio. Para presentar las conclusiones y resultados científicos se utilizan las vías habituales: artículos en revistas especializadas, aportaciones a congresos y presentaciones de tesis o proyectos.

La difusión de los datos obtenidos, por ejemplo los datos de secuenciación suele realizarse por medio de organismos especializados (NCBI, Laboratorio Europeo de Biología Molecular (EMBL), Japan Data Bank, GenBank, etc...), que centralizan todos los datos sobre genes y proteínas y los ponen disponibles para la comunidad científica, de modo que otros investigadores puedan apoyarse en tu trabajo para avanzar.

También existen repositorios de datos brutos de secuenciación o de experimentos con micromatrices, como son SRA [110], ArrayExpress [165] y GeneOmnibus [56], para que el investigador los descargue y manipule según su criterio para el mismo u otros objetivos que el que llevó a su creación. No es menos importante que revistas como *Nucleic Acids Research* y *Plant Cell Physiology* dediquen un número anual a las bases de datos biológicas, o que Oxford University Press haya dado salida a una revista dedicada exclusivamente a bases de datos, *Database*.

Como parte de esta necesidad de difusión, cada vez son más los proyectos en los que desean realizar bases de datos con una interfaz sencilla para exponer sus resultados y promocionar su investigación lo máximo posible. Estos proyectos pueden beneficiarse de herramientas que han emergido en los últimos años para facilitar la creación de aplicaciones web que presenten los datos almacenados en bases de datos. Entre dichas herramientas nos encontramos con Ruby on Rails [74], que es una combinación del lenguaje de programación Ruby [62] junto a unas librerías de programación (Rails) que siguen un paradigma de programación llamado "convenio mejor que configuración" (*Convention over configuration* [232]) y una arquitectura de programación basada en modelo/vista/controlador [53] que ayudan en la organización de código y proporcionan un aumento de la productividad del programador. A partir de los cimientos plantados por Ruby on Rails, han aparecido librerías que siguen los mismos paradigmas para otros lenguajes de programación, y ya podemos encontrar alternativas maduras como Django [78] para Python [171], o CakePHP [26] para PHP [112]. Para ahorrarse desarrollos completos, nos encontramos con herramientas como GBrowse [209], que permiten publicar una base de datos genómicos con anotaciones gracias a una interfaz autoadaptable.

Capítulo 4

Objetivos

La motivación principal al desarrollar este estudio fue la de incrementar la presencia de la supercomputación y las técnicas de virtualización emergentes en las investigaciones que requieran la aplicación de aproximaciones bioinformáticas, sobre todo las relacionadas con la ultrasecuenciación, una tecnología cada vez más usada en los laboratorios. Además, había que procurar que fueran accesibles a los investigadores que se inicien en la bioinformática, sin que eso implique unos amplios conocimientos de programación. Por tanto, queríamos ofrecer un sistema optimizado con herramientas de alto nivel de fácil acceso y uso, con lo que los objetivos específicos que nos planteamos son:

1. Facilitar el acceso a los recursos de supercomputación para los usuarios finales mediante la creación de un sistema de autoservicio de programas bioinformáticos y mediante interfaces de usuario amigables para las herramientas que utilicen recursos de supercomputación.
2. Optimizar los recursos de supercomputación para la avalancha de datos de la ultrasecuenciación mediante el desarrollo de herramientas que faciliten la programación de arquitecturas paralelas y distribuidas, así como aprovechar mejor los siempre escasos recursos de almacenamiento.
3. Desarrollar nuevos algoritmos que resuelvan problemas biológicos concretos.
4. Proporcionar soluciones de almacenamiento ordenado a largo plazo y la difusión de los resultados de los proyectos de investigación mediante el acceso a bases de datos especializadas con interfaz web.

Parte II

FRAMEWORKS PARA FACILITAR EL USO DE LA BIOINFORMÁTICA

Capítulo 5

Autoservicio de bioinformática

5.1. Introducción

En bioinformática se contemplan dos extremos muy opuestos en necesidades de cálculo. Existen muchos experimentos relacionados con la ultrasecuenciación donde es imprescindible recurrir a recursos de supercomputación, con el engorro que eso significa a la hora de preparar los flujos de trabajo, envíos a sistemas de colas, etc. Pero hay otras situaciones en las que la cantidad de datos a procesar es relativamente pequeña y puede realizarse con comodidad en un ordenador personal, con una visualización más agradable e intuitiva para el usuario. El problema surge a la hora de instalar dichas aplicaciones, mantenerlas funcionando correctamente, o incluso la imposibilidad de comprar una licencia de pago para su uso puntual. La solución ideal a este problema sería que los usuarios tuviesen acceso instantáneo a cualquier programa que necesiten, ya sean de pago o gratuitos, y que este se encuentre ya instalado y listo para utilizar en un entorno como el de su propio ordenador personal.

En este capítulo se explicará la arquitectura que se ha diseñado para proporcionar a los usuarios acceso remoto a un conjunto de máquinas virtuales idénticas con programas para análisis bioinformáticos ya instalados que a su vez les permite compartir los datos y por lo tanto realizar procesos complejos de cálculo directamente con los recursos de supercomputación.

5.2. Elección del acceso remoto a los recursos

La mayoría de los programas bioinformáticos con entorno gráfico para ordenadores personales se encuentran disponibles en exclusiva para el sistema operativo Windows®, por lo que las máquinas virtuales se basarán en él, al igual que los programas instalados. Esta decisión ha condicionado las opciones que podemos elegir para el protocolo de comunicación que permita el acceso remoto a las máquinas, la forma de acceder al almacenamiento compartido y la creación del *broker* de máquinas virtuales.

Entre los protocolos más generales de acceso remoto de escritorio nos encontramos con:

SSH con interfaz gráfica Interfaz gráfica para sistemas Linux/UNIX (X11): El servidor SSH [238] se encuentra disponible de serie en todas las plataformas Linux. Es útil para ejecutar programas individuales sin mucha carga gráfica en un entorno remoto, pero no da una experiencia de escritorio completa y las cargas iniciales de los programas puede

ser desesperantemente lentas debido a que X11 transfiere todos los datos relacionados con la aplicación para su ejecución remota, aunque la ejecución suele ser fluida, pues la comunicación con el servidor una vez terminada la carga inicial es mínima. El usuario final se conecta por ssh a una máquina linux remota, ejecuta la aplicación gráfica con un comando en el terminal y en ese momento empieza la descarga de la mayoría de datos necesarios para ejecutar el programa en el ordenador cliente y realizar la ejecución directamente allí. El ordenador cliente debe tener instalado un entorno X11 para poder ejecutar la aplicación, que suele venir instalado en Linux/UNIX, pero no en el resto de sistemas operativos. Esta solución no se adapta a lo que necesitamos al no permitir el acceso a servidores con sistemas Windows®.

Virtual Network Computing (VNC): Puede instalarse para todos los sistemas operativos actuales, incluso móviles o *tablets* [214]. Consta de dos partes: un servidor que se instala en la máquina que se desea controlar remotamente, y un cliente que se instala en la máquina que se va a usar para controlar la máquina remota. En el cliente, el usuario abre un programa, escribe la dirección Internet Protocol (IP) o nombre del servidor al que quiere conectarse, se identifica con su usuario y clave de VNC que le garantiza el acceso al equipo, y entonces se le abrirá una ventana con la pantalla completa del equipo remoto. Esta solución da una experiencia de usuario mejorada respecto al X11 en lo relativo a la velocidad de arranque inicial y a la interacción con la máquina remota, puesto que el usuario ve el escritorio completo, tal como lo estaría viendo si estuviese sentado delante del monitor del ordenador remoto. El funcionamiento de VNC se basa en el envío del *bitmap* completo de la pantalla a través de la red (y no los datos), por lo cual solo parecerá lento en las redes de poca velocidad. Para mejorar este aspecto, actualmente existen implementaciones (como la utilizada por Apple Inc. en su OS X), que disponen de algoritmos de calidad adaptativa y que además detectan qué zonas de la pantalla es necesario redibujar para que se envíen únicamente los trozos de pantallas que deben refrescarse. Esta opción sería perfectamente válida para nuestro cometido, pero su integración con dominios Active Directory® de Microsoft® para permitir que varios usuarios se conecten a la vez a un mismo equipo no es totalmente transparente. Exigiría además la instalación de un programa cliente en todas las máquinas de usuario.

Remote Desktop Protocol (RDP): Implementado por Microsoft®, el Remote Desktop Protocol [222] viene instalado de serie en los sistemas operativos Windows actuales®, por lo que se evitaría la instalación de un programa cliente en la mayoría de las máquinas. Los equipos con OS X o Linux sí que necesitarían la instalación de un pequeño programa gratuito. El RDP se integra correctamente con Active Directory [155], lo que permite usar las mismas cuentas de usuario que estén dadas de alta en el supercomputador para acceder a la propia máquina. Siempre que sea posible, RDP no enviará una imagen de la pantalla completa a través de la red, sino una primitiva que indica al programa cliente qué hacer, como crear una ventana de tales dimensiones, moverla, cerrarla, poner un botón o texto en tal sitio o una barra de desplazamiento en el otro. Esto hace que su uso en redes lentas sea muy eficaz. Además, para mejorar el rendimiento, incluye opciones para limitar el número de colores a mostrar, ignorar adornos de ventanas y fotos de fondo de escritorio del equipo remoto (no ocupa lo mismo una foto que enviar una primitiva que dice: fondo azul), e incluso pueden traspasar el canal de sonido del equipo remoto al cliente, integrar los dispositivos de almace-

namiento local del cliente en el equipo servidor, o utilizar múltiples monitores. Esta es sin duda la opción elegida, aunque sólo sea posible utilizarla para controlar remotamente máquinas con Windows®.

Otras opciones comerciales: Existen otros programas que podrían suplir perfectamente las necesidades de un protocolo de acceso remoto, pero a costa de esquemas de licencias y pagos periódicos basados en el número de clientes conectados y en los servidores disponibles. Esto los hace menos atractivos para entornos de investigación donde los recursos económicos siempre son muy escasos. Así, la compañía Citrix Systems, Inc., tiene el programa Citrix Workspace Suite para que se pueda acceder de forma segura a aplicaciones, escritorios, datos y servicios desde cualquier dispositivo. Los usuarios (todos) tendrían que instalarse el programa gratuito Citrix Receiver. Otra opción hubiera sido elegir TeamViewer [72], que aunque permite un uso esporádico gratuito, requiere un pago periódico si se pretende hacer un uso regular.

5.3. Elección del directorio de usuarios centralizado

Cuando hay un conjunto de máquinas diferentes, lo deseable es permitir que cualquier usuario pueda utilizar cualquier máquina con las mismas credenciales, tanto porque facilita la gestión de los usuarios y las máquinas, como porque simplifica los accesos a los usuarios. Esto conlleva la existencia de un directorio de usuarios centralizado. El administrador realiza el alta del usuario en el sistema central y todos los equipos cliente identifican a sus usuarios en este servidor, por lo que no es necesario dar el alta de forma individual en cada equipo. Existen diferentes alternativas en función de la plataforma que se utilice, entre las que cabe destacar:

Network Information Service (NIS): Antes se le denominaba páginas amarillas [94] y era muy utilizado. Todavía hoy día existe una gran cantidad de servidores que lo usan, a pesar de que se trata de una organización plana, y la tendencia general actualmente es instalar LDAP que proporciona una configuración mucho más flexible. Se puede utilizar para identificar usuarios en redes de equipos Linux/UNIX.

Active Directory®: Se trata de la solución comercial de Microsoft® [155] para los servidores con Windows NT o Windows Server, y puede utilizarse para controlar la identificación de los usuarios y máquinas cliente que usen un sistema operativo Windows® que se encuentren asociadas al dominio del servidor. Se trata, sin embargo, de un sistema con licencia variable respecto al número de máquinas o usuarios y que sólo permite identificar máquinas con Windows® instalado.

LDAP: Organiza el directorio de usuarios y sus datos en un árbol de entradas jerarquizadas reconfigurable [199]. Por ejemplo, en el nivel más alto de la jerarquía podría encontrarse un centro de investigación, bajo el cual podrían definirse diferentes grupos de investigación, que a su vez pueden agrupar personas, máquinas, impresoras, servicios DNS o DHCP, o cualquier entrada que se nos ocurra definir. También se pueden definir campos con información personalizada para cada objeto que se almacena en un directorio LDAP. A la hora de realizar la autentificación, las máquinas cliente pueden definir en qué rama del árbol va a comenzar la búsqueda, lo que permite establecer con facilidad diferentes niveles de acceso a di-

ferentes grupos de máquinas. Por ejemplo las máquinas de un laboratorio estarían configuradas para comenzar la búsqueda a partir del nodo de ese laboratorio, por lo que no tendrían acceso a los datos de un laboratorio distinto. LDAP puede integrarse con un servidor Samba para ofrecer servicios de identificación de usuarios simulando una máquina Active Directory, pero además lo puede utilizar el resto de máquinas Linux/UNIX u OS X para identificar sus usuarios o incluso ofrecer servicios de listín telefónico. La elección por tanto para el sistema de directorio centralizado de usuarios será LDAP (en concreto su implementación OpenLDAP) y un controlador de dominio Samba.

5.4. Elección del almacenamiento compartido

Acceder a diferentes máquinas con el mismo usuario sin disponer de un sistema de almacenamiento compartido es algo que no tiene mucho sentido porque los archivos se quedarían repartidos por las diferentes máquinas que el usuario hubiese utilizado. Para ello existen protocolos que permiten que distintas máquinas compartan los mismos archivos. A su vez, cada usuario debe tener su espacio de almacenamiento privado en un servicio de almacenamiento compartido. Los servicios de almacenamiento compartido más habituales para los entornos de escritorio son los siguientes:

NFS: Se trata del protocolo más utilizado en entornos Linux/UNIX y consiste en un servidor NFS al que se conectan los diferentes usuarios desde sus máquinas cliente mediante un cliente NFS que suele venir instalado en el sistema [202].

Apple File Protocol (AFP): Se trata del protocolo que los equipos con sistemas operativos de Apple utilizan para compartir archivos entre ellos. Desde la llegada de OS X, éstos también pueden acceder a NFS y Samba sin ningún problema.

Server Message Block (SMB)/Common Internet File System (CIFS): El protocolo Server Message Block (SMB) lo utilizan las redes de Windows® para compartir archivos. También es el protocolo utilizado en los dominios Active Directory® para proporcionar a los usuarios el acceso a sus datos y directorios compartidos [153].

Samba: Además de actuar como un controlador de dominio equivalente a Active Directory®, proporciona los mismos servicios que SMB/CIFS para entornos Linux/UNIX y OS X [172]. Como podemos ver en la figura 5.1, el servidor permite definir los directorios y permisos que cada usuario tendrá disponible, pero además, al integrarlo con LDAP, nos permitirá que un usuario obtenga automáticamente un punto de montaje con sus datos cuando se conecta a cualquiera de las máquinas. En realidad, los datos están alojados en el mismo sistema de almacenamiento centralizado usado en la parte de supercomputación, situación que permite que el usuario tenga a su disposición en una máquina con entorno visual y herramientas gráficas los mismos datos que en el entorno de sistema de colas y línea de comandos del supercomputador. Esta es por tanto la opción que hemos elegido para nuestro sistema de almacenamiento compartido.

Figura 5.1: *Almacenamiento compartido para clientes basado en Samba*

5.5. Elección del sistema de virtualización

La virtualización [167] consiste en dividir un ordenador real, con su memoria, almacenamiento, procesadores y red, en un conjunto de máquinas ficticias o virtuales, e instalar en cada una de esas submáquinas otro sistema operativo (que se dice que está virtualizado) que pueden ser iguales o distintos entre sí, con sus aplicaciones y programas correspondientes. Podemos por lo tanto considerar esta técnica como un modo de consolidar recursos que permite reducir los costes de adquisición de equipos y de gestión de su funcionamiento a cambio de un impacto mínimo en el rendimiento de la máquina virtualizada.

Para usar virtualización necesitamos usar un hipervisor, que es un programa que se sitúa debajo del sistema operativo virtualizado y que se encarga de proporcionarle los distintos servicios que ofrecería una máquina real: acceso a disco, acceso a memoria, ejecución de código, acceso a la red, etc. El hipervisor será el encargado de interceptar las instrucciones que ejecute la máquina virtual y realizar las adaptaciones necesarias antes de pasarlas al *hardware* subyacente. La intervención del hipervisor es tan liviana que el rendimiento de una máquina virtual es muy cercano al de una máquina real [87]. El buen rendimiento también se debe al uso de procesadores y *chipsets* modernos en los que los fabricantes incorporan extensiones para mejorar aun más el rendimiento de las máquinas virtualizadas [4], como son las tecnologías VT-X [151] y VT-d [2] de Intel® o AMD-V [5] del fabricante AMD®.

En función de la capa en la que se sitúe el hipervisor entre el sistema virtualizado y el hardware real distinguimos dos tipos principales [11]:

Hipervisor de tipo 1 o *baremetal*: el hipervisor se instala directamente sobre el *hardware* de la máquina (figura 5.2), sin necesidad de tener un sistema operativo anfitrión que le dé soporte. El hipervisor captura las peticiones del sistema virtualizado y las pasa directamente a los elementos correspondientes de su *hardware*, realizando las tareas de planificación y encolado necesarias. Los *baremetals* son los más eficientes al no requerir un sistema operativo intermedio. Como ejemplos tenemos VMware ESX o VMware ESXi [154], XEN [169], MS Hiper-V [97] o Kernel-based Virtual Machine (KVM)[101]. Hay quien considera que KVM es un hipervisor tipo 2 (véase el párrafo siguiente), pero en realidad es tipo 1 ya que

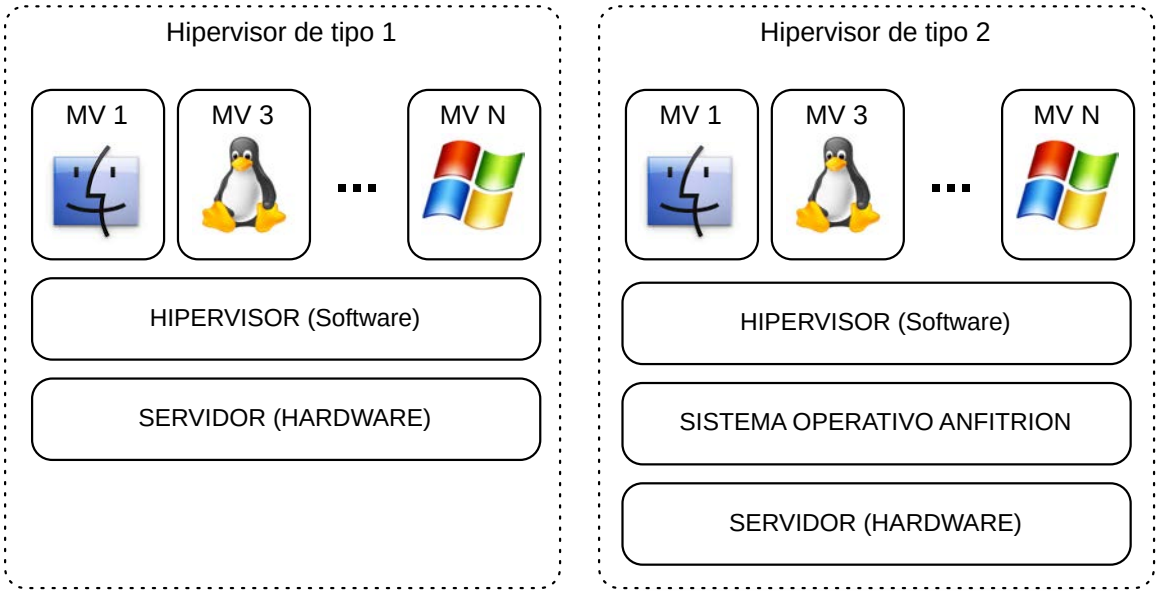


Figura 5.2: *Comparación entre los hipervisores de tipo 1 y 2*

se ejecuta en el mismo nivel que el *kernel* del sistema operativo y no encima de éste. Este sistema es el más utilizado para virtualización de servidores y es el que utilizaremos en nuestro sistema porque proporciona mayor rendimiento y aprovechamiento de los recursos, lo que permite alojar más máquinas virtuales por cada servidor real utilizado.

Hipervisor de tipo 2 o *hosted*: En este caso, el hipervisor se instala sobre un sistema operativo (figura 5.2) ya existente (ya sea Linux/UNIX, OS X, Windows,...). El hipervisor pasa las peticiones del sistema operativo virtualizado hacia el sistema operativo real que las pasa a su vez al *hardware* de la máquina. Este tipo de sistema tiene un rendimiendo algo inferior al tipo 1 debido a que hay recursos que ya se están utilizando para el sistema operativo anfitrión y además existe una capa adicional que puede ralentizar las peticiones. Por otro lado, tienen la ventaja de que el hipervisor puede convivir con los demás programas de usuario que estén en dicha máquina y esto proporciona mayor comodidad para virtualización en equipos de escritorio. Ejemplos son VMware Workstation, VMware Player, VMware Fusion o VMware Server, QEMU o VirtualBox.

¿Virtualización completa o paravirtualización? El sistema operativo virtualizado puede ser un sistema operativo sin modificaciones, en cuyo caso hablamos de virtualización completa, o puede ser un sistema operativo especialmente modificado para comportarse mejor en entornos virtualizados, en cuyo caso hablamos de paravirtualización. Existen también *drivers* paravirtualizados para dispositivos específicos (como la tarjeta de red) que ayudan a incrementar el rendimiento cuando se utiliza virtualización completa.

En el sistema que vamos a desarrollar en

este trabajo usaremos la virtualización completa con hipervisores de tipo 1, en concreto con el sistema ESX de VMware instalado en un *cluster* de dos servidores HP DL380-G6, con 8 núcleos y 32GB de RAM cada uno, que proporcionan las ayudas de Intel® para la virtualización.

5.6. *Broker* de máquinas virtuales

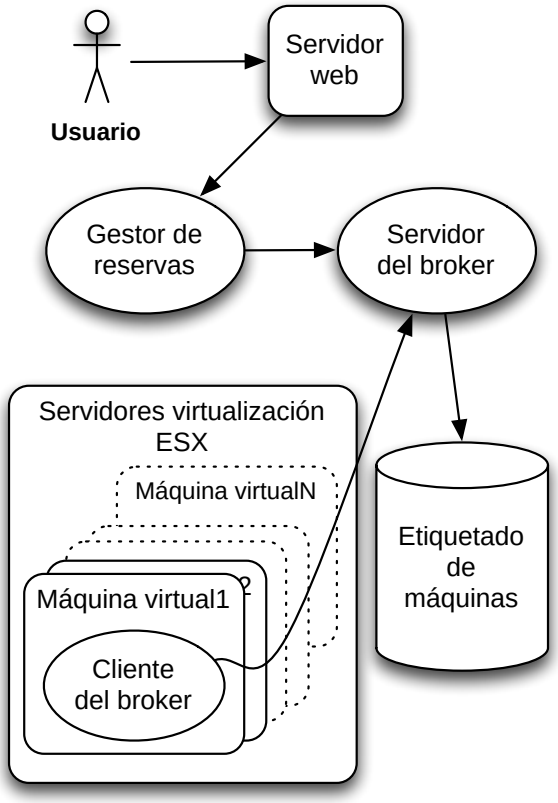


Figura 5.3: *Arquitectura del* broker *de máquinas virtuales*

Una vez definido que utilizaremos máquinas virtuales con un sistema operativo Windows® y un dominio Samba con autentificación LDAP para el acceso de los usuarios y los datos, surge el problema principal: son máquinas virtuales a las que se va a acceder con un protocolo de acceso remoto (RDP), lo que requiere la constante vigilancia de quién está utilizando cada máquina, qué máquinas están libres y cuáles están en uso.

Para que este problema no repercuta en la facilidad del uso del sistema ni en la comodidad de los usuarios, hemos desarrollado lo que llamamos un *broker* de máquinas virtuales con una interfaz web que tiene la siguiente arquitectura (figura 5.3):

Servidor del *broker*: Se trata de un servidor escrito en perl que recibe periódicamente el estado de cada una de las máquinas virtuales. El estado incluye información sobre el posible usuario que está registrado en la máquina, y sirve para mantener una base de datos con los usuarios activos y máquinas ocupadas. Las máquinas tendrán etiquetas para determinar su tipo, pues no todas tendrán las mismas características ni las mismas aplicaciones instaladas.

Cliente del *broker*: Se trata de un programa en Delphi que se ejecuta como un servicio en cada una de las máquinas Windows®, y se conecta con el servidor del *broker* para facilitarle los datos del usuario que está usando la máquina en ese momento. Si no hay usuario, el *broker* entenderá que la máquina está libre.

Servidor web: Muestra a través de la web un listado de los tipos de máquinas disponibles correspondientes a cada etiqueta, basándose en la información que mantiene el servidor del *broker*. Desde esta web podrá seleccionar el tipo de máquina deseado, lo que desencadena la ejecución del gestor de reservas (apartado 5.6), y se descarga un archivo con extensión .rdp con

los datos de acceso necesarios para conectarse a una máquina virtual libre. Este fichero se abrirá automáticamente (en Windows®) o con un doble clic en el programa cliente del protocolo RDP y le pedirá las credenciales de acceso para abrir la máquina. En la figura 5.4 se muestra el proceso de conexión a al escritorio remoto. El usuario podrá elegir entre: (1) una máquina BÁSICA que contiene todas las aplicaciones que tenemos en la Plataforma Andaluza de Bioinformática para genómica, transcriptómica, expresión génica, etc., que se puede consultar en la página web `http://www.scbi.uma.es/site/scbi/software`; (2) una máquina GEDECYDER, que consiste en una máquina básica más la aplicación GE DeCyder-2D para análisis diferencial de geles de proteínas; y (3) la máquina DISCOVERY que está también basada en la máquina básica y contiene los programas Discovery Studio, HyperChem y PDQuest.

Figura 5.4: *Etapas de la conexión a los escritorios remotos a través del portal web.*

Gestor de reservas: Se trata de un programa escrito en Perl que hace de intermediario entre la página web que presenta los datos al usuario y el servidor que recibe la información de estado de las máquinas. Este programa se comunica con el servidor del *broker* para obtener el número de máquinas disponibles de cada etiqueta y, posteriormente, cuando el usuario realiza la petición de una máquina con una etiqueta concreta, se encarga de solicitar que esa máquina sea retirada del conjunto de máquinas disponibles y reservada para que el usuario pueda conectarse.

El control proporcionado por el *broker* permite además monitorizar cuantas máquinas se están utilizando en cada momento. Además, se le pueden añadir *scripts* al sistema para arrancar o apagar máquinas adicionales cuando sea necesario. El *broker* dará de baja automáticamente las máquinas apagadas o incorporará las máquinas recién arrancadas a la reserva de máquinas disponibles.

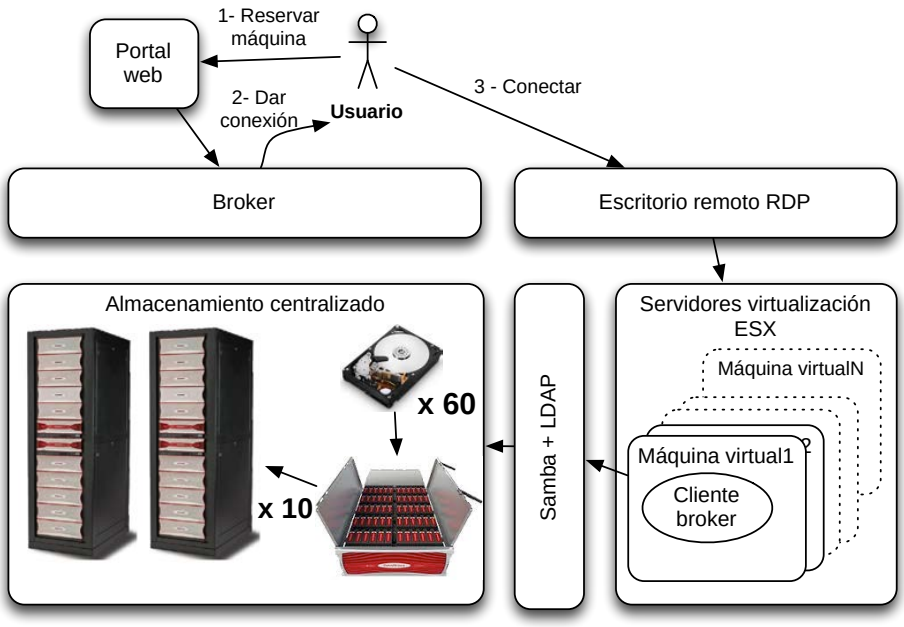


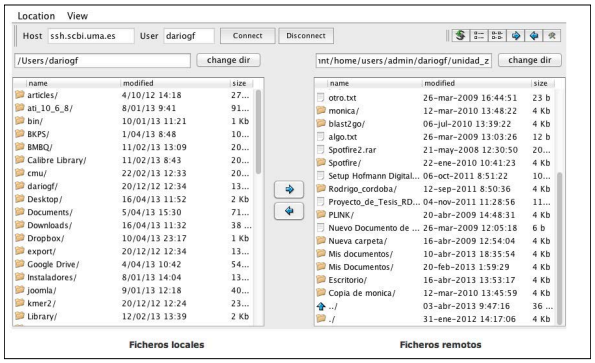
Figura 5.5: *Esquema de funcionamiento del* broker *de máquinas virtuales*

5.7. Conclusiones

Hemos creado un sistema de escritorios virtuales basado en el uso de hipervisores de tipo 1 (sección 5.5) con acceso remoto mediante protocolo RDP (sección 5.2) en combinación con un sistema de autentificación LDAP (sección 5.3) y almacenamiento compartido basado en Samba (sección 5.4) que permite solicitar los recursos a través de la web gracias al *broker* descrito en la sección 5.6, y proporciona a los usuarios la comodidad necesaria para acceder a programas instalados, ya sea desde sus despachos, laboratorios o incluso desde su propia casa. En la figura 5.5 podemos ver cómo se conectan entre sí los diferentes elementos que hemos ido introduciendo en las secciones anteriores: el usuario inicialmente reserva una máquina a través de un portal web, el portal consulta al *broker* e indica al usuario los datos de conexión a la máquina. Cuando el usuario se conecta mediante su programa de escritorio remoto (RDP) a la máquina virtual indicada por el *broker*, se encuentra con que su espacio reservado en el almacenamiento centralizado ya ha sido montado a través de Samba y por tanto tiene sus datos disponibles en la unidad Z: de la máquina (figura 5.6). Para intercambiar archivos deberá usar un programa de SFTP conectándose al almacenamiento centralizado del supercomputador, como explicamos en el apartado 2.5.4.

Además, al disponer de máquinas con distintos conjuntos de aplicaciones, los propios usuarios decidirán qué tipo de máquina y herramientas usarán para realizar su trabajo. Como ya hemos indicado, los datos de usuario que se utilizan en todas las máquinas están almacenados en un almacenamiento compartido, con lo que no tienen que estar copiando o moviendo datos de una máquina a otra.

1 - Acceso a archivos compartidos por SFTP



2- Acceso a los archivos desde el escritorio remoto

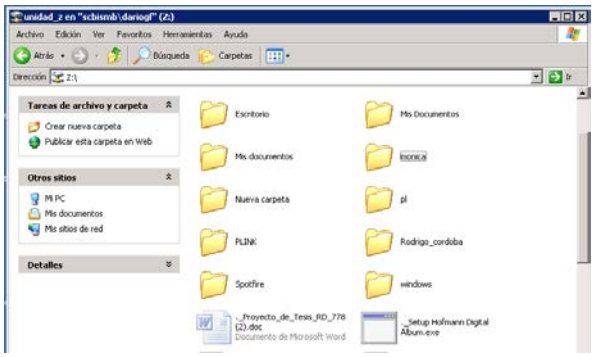


Figura 5.6: *Acceso a los archivos por SFTP y a través de la conexión a escritorio remoto de la máquina virtual.*

Este trabajo planta la semilla del "Autoservicio de supercomputación para bioinformática", que seguirá evolucionando con el resto de trabajos aportados en esta tesis.

El sistema implementado actualmente permite el acceso a tres tipos de máquinas virtuales distintas, aunque este número puede configurarse simplemente añadiendo una nueva etiqueta para la IP de la máquina en el servidor del *broker*.

Capítulo 6

Generador de interfaces web para dar acceso a programas desarrollados en línea de comandos

Mi contribución: El desarrollo completo de la herramienta InGeBIOL.

InGeBIOL: A web interface generator for command line tools

Diego Darío Guerrero Fernández (dariogf@uma.es)[1]
M. Gonzalo Claros (claros@uma.es)[1,2]

[1] **Supercomputación y Bioinformática (SCBI), Plataforma Andaluza de Bioinformática (PAB)**
[2] **Departamento de Biología Molecular y Bioquímica**
Universidad de Málaga, 29071 Málaga, Spain

Keywords: command line, interactive user graphic interfaces, web interfaces, rest web services

Abstract

Motivation: Interfaces are an important factor in the success of an application. Many competent command-line applications are usually underused because of their complex interface that frightens users with countless switches and options. Surely, command-line tools will be more popular if an attractive and easy-to-use interface were provided. Web interfaces can be the smarter solution, since they do not require the installation of additional software in client computers and their flexibility is rapidly increasing due to new technologies like AJAX. The aim of this work is to provide a framework for programmers that can serve to offer web interfaces for new and legacy command-line applications and tools with a minimum effort.
Results: An easy and flexible framework to create web interfaces for command line tools called InGeBIOL has been developed. It also includes some powerful features that are essential in supercomputing centers and other centralized facilities, such as job management, queue system integration, automatic REST web service compatibility, data-storage in private repositories and LDAP user authentication. Interfaces generated with InGeBIOL can be interactive, with automatic input data validation, conditional visibility, on-screen help to illustrate the tool usage via web, or centralised data repositories. InGeBIOL can also provide web interfaces for workfows when treated as a "black box" or when they are simple pipelines. It has been used to build a web portal for the bioinformatic tools developed at the Plataforma Andaluza de Bioinformática (SeqtrimNEXT, FullLenghterNEXT, GENote β.1) and others third party tools (CAP3, CABOG, Maker,...) in a hierarchical organisation. Those services can be accessed at http://www.scbi.uma.es/ingebiol/ A step by step guide is also provided to develop a web interface for a simple command line application. **Availability:** InGeBIOL Can be downloaded at `https://github.com/dariogf/ingebiol`

1 Introduction

It has been described that there is an increasing need for the development of data analysis software that provides bioinformatics functionalities without requiring prior knowledge of programming and scripting languages. In fact, it has been demonstrated that most used software for phylogenetic analyses is correlated to the user-friendliness and not the appropriateness of software [18]. Therefore, complete bioinformatic tools do not have to be necessarily the most used one and that is the reason why there is always room for proprietary software for bioinformatics. This is in agreement with the fact that users prefer less powerful software provided that it is easier to deal with it, rejecting cumbersome, or command-line tools [25]. This situation is prompting for an important problem in bioinformatics: there are many useful and effective tools available as command line tools without an user interface that most scientist ignore, are not eager to use, or refuse their use [1].

Not less important is that bioinformatic tools

need to be useable on different operating systems, and they need to provide a natural language description of the results produced in order to state assumptions made in the analysis [18]. One possibility could be to develop tools in Java™, enabling the use of a graphical interface and obtaining an executable command for all platforms. While in theory Java™ will allow to code a program once, and run anywhere, in practice, programmers need to be careful about their coding if they want their application to display the same interface in all platforms without errors. This is the reason why most Java™ programs do not look fine in any platform. Another drawback to this approach is that users would need to install the correct version of Java™, and the application itself on their machines, and tackle with potential compatibility issues. Nowadays, web applications are accessible to a wide number of users: they are easy to use, do not require users to install additional software in their computers, and provide a decentralised access to the application. All together, they make web applications able to exploit resources with effectiveness, it has become the best solution for platform-independent software that can also take advantage of supercomputing facilities. Furthermore, new techniques like AJAX [29] have been used to improve web interfaces and mimic a desktop user experience.

Some efforts have been performed concerning automatic web interface generators. Pise [13], JEMBOSS [7], Galaxy [5] or Bioweb [24], despite being examples of the most complete and flexible tools available at the moment, do not offer neither a rich user interaction via AJAX, nor job management or an explicit queue system integration. Most of them are also based on XML configuration that is not very pleasant to be modified by human administrators. RGG [27] and fgui [15] are GUI generators but they are only available for R programs.

In our research group, we have developed command-line and web tools like AlignMiner [14], SeqTrim [9, 10], Full-Lengther [19] to cite a few examples, as well as a framework for automatising workflows: AutoFlow [26]. Moreover, our experience in providing bioinformatic services to users at PAB (Plataforma Andaluza de Bioinformática; `http://www.scbi.uma.es`) revealed that third-party software such as Maker [6], Cap3 [16], Cabog [21] will be more used if a web interface was available. Since none of the previously described software had the features that we wished, we decided to create a general framework that can be used to provide a web interface for any command-line tool.

Here it is presented InGeBIOL, that is not a program library but a kind of web application that generates web interfaces from a bunch of configuration files. It provides a framework to a) easily add web interfaces with modern look for legacy and new command-line tools, b) transparently submit jobs to queue systems used in supercomputing facilities, c) be able to manage queued jobs in order to browse, download, or delete them, d) offer developers an easy-to-use and flexible environment to setup web interfaces for their existent or future developments, e) automatically define a webservice with a REST API to access each command, and f) organise the software in a customisable hierarchy. It is expected that InGeBIOL could serve to promote interesting command-line tools for bioinformatics and any other field of science and research.

2 Implementation and installation

InGeBIOL has been developed in Ruby-on-Rails 2.3 (RoR [2], a framework based on Ruby [12]), and making use of AJAX technology [29] to provide interactivity. Since Ruby is preinstalled in most Linux/UNIX operating systems (and can also be installed in Windows), it is possible to install InGeBIOL on any platform. However, most command-line tools, supercomputers and queue systems run on Linux/UNIX systems, and thus a Windows installation does not seem appropriate in the bioinformatics field.

Since InGeBIOL works on top of an HTTP web server, and HTTP is a disconnected and stateless protocol, it can handle concurrent users out of the box, the same way that multiple users can navigate any website simultaneously. Furthermore, InGeBIOL can cope with concurrent jobs from multiple users thanks to the use of a queue system. Submitted jobs are sent to the queue system and the server is immediately released of its execution so it can handle additional jobs or user requests. However, HTTP also imposes some size limits for big file uploads. To cope with this situation, a personal repository where the users can upload files with SFTP can be configured by system administrators.

InGeBIOL is dependent on the presence of Ruby and Ruby-on-Rails. Ruby is preinstalled on most Linux operating systems but, if required, the easiest way to install it is by using a Ruby installation manager like RVM (`https://rvm.io`), and executing `rvm install ruby 1.9.3`. Then, RoR can be installed by issuing a simple `gem install rails -v 2.3.8`. Since the required environment is already setup, InGeBIOL source can then be downloaded from github (`https://github.com/dariogf/ingebiol`), and the server can be started by navigating to the `gui` folder and executing the

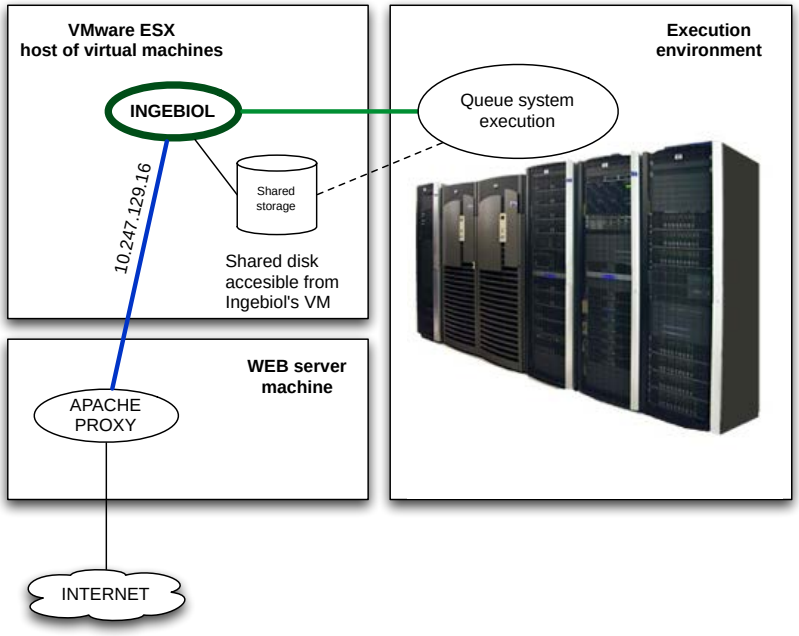


Fig. 1: Recommended deployment environment of InGeBIOL

`script/server` command. The web site will be accessible at http://localhost:3000.

InGeBIOL server can be setup behind an HTTP proxy to make it accessible from the outside, following an schema similar to the one described in Figure 1. In this case, the virtual machine with the RoR server is hosted on a VMware ESX hypervisor, with one network adapter to gain access to the supercomputers queue system, and another one to connect to the Apache proxy. The proxy will receive request from the Internet, and redirect them to the RoR server and the response will follow the reverse path.

3 Results and discussion

3.1 Configuration is based on folders and JSON files

A custom interface generator must gather information about each tool (how many input fields should be generated, type of each field, titles, data validation, hints and help, commands to be executed with the input data, results to be shown, etc.). In order to facilitate installation and administration tasks, all this information (and more) is not stored in a database (that requires additional installation and managing) but in a JSON file format [17] that has been extended with comments support. The possibility of using XML files [28] that already accepts comments was discarded because they tend to grow

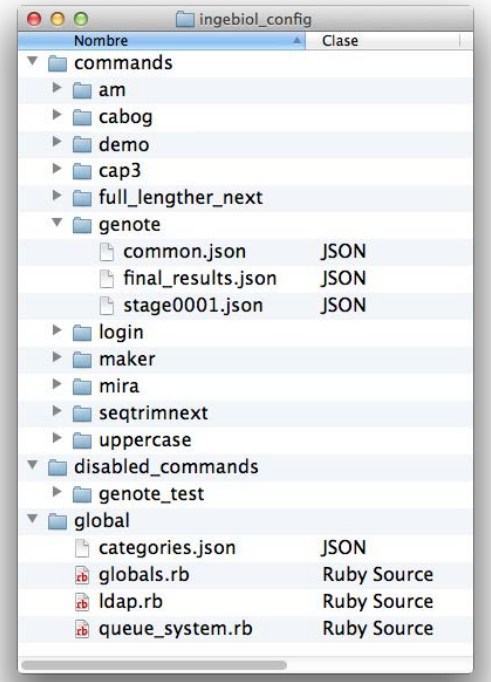


Fig. 2: Example of file and folder hierarchy where an InGeBIOL portal is defined to contain a set of commands (am, cabog, cap3...) and a disabled command (genote_test).

exponentially as more nested information is saved.

Configuration files are stored on a separate folder out of the source tree of the application and organised following the hierarchy shown on Figure 2. The "global" folder contains the configuration files used to adapt InGeBIOL to most environments. The "commands" folder contains one subfolder for each active command and its configuration files. Finally, the "disabled_commands" folder contains temporarily disabled commands.

3.2 A web portal of bioinformatics tools

The main application of InGeBIOL is to provide a web portal and REST web services for bioinformatics tools, with both, private or public access gathering multiple command-line tools in the same installation, powered with LDAP user authentication to be integrated on existing working environments, queue system support to delegate job execution to supercomputing facilities, a dock where every tool can be selected, and a direct URL to every command-line.

3.2.1 Common interface

When the web portal of InGeBIOL is reached with a web browser (`http://your_ingebiol_site`, for example `http://www.scbi.uma.es/ingebiol`), the default home page is presented (Figure 3). It contains four main parts:

- a navigation dock where the user can select different commands;
- a login panel to authenticate registered users or give guest access to them;
- the name and description of the default login command; any particular command defined in InGeBIOL can also be used to login the web portal;
- the REST API link that defines the REST web service [3]; it is automatically created for each command.

General InGeBIOL capabilities can be configured in the "global" folder in Figure 2 as explained in next sections. They will affect to all commands covered by the same installation.

The `globals.rb` file defines in Ruby the basic configuration parameters. They have default values that will fit most environments, but they can be customised. The file is self-explanatory concerning modifiable parameters, although most will not need any intervention. The complete description can be found on Appendix A.4.1.

3.2.2 LDAP authentication

Since LDAP is an widespread way of authentication, and it is the protocol installed at PAB, the capability to be integrated with already existing LDAP servers has been included in InGeBIOL. The file `ldap.rb` within the folder "globals" (Figure 2) contains the parameters to configure the LDAP authentication mechanisms (a complete description of this file is given in Appendix A.4.2). The most important parameters to be configured are the address of the host where the LDAP server is running (LDAP_HOST) and a search string pointing to the branch containing the users in the LDAP tree (LDAP_USERS_UID). For example, this are the parameters for the LDAP server at PAB:

```
# file: globals/ldap.rb

#LDAP authentication server
LDAP_HOST = '10.200.190.1'

# How to query users to LDAP server (may
    change between organisations or
    OpenDirectory and OpenLDAP servers)
LDAP_USERS_UID='uid=%s, ou=people, dc=mylab
    , dc=uma, dc=es'
```

3.2.3 Navigation dock and direct URLs

All commands implemented in a InGeBIOL web portal should be organised in a hierarchical tree (Figure 4), so that they can be browsed with the navigation dock (Figure 3). This hierarchy is defined in the file `categories.json` within the folder "globals" (Figure 2). Each item of the tree must have a `tag` that should be included in the corresponding commands to locate them in the corresponding branch of the tree and a `title` to define the text that the user will see. A category can also have a `tooltip` that will appear as a tooltip on the web to inform users about the category. Categories are considered 'parent' so that they can have one or more 'children'. An example of two categories, one of them with two children, is defined in the following code:

```
# file: globals/categories.json

[
  #first category
  {
    tag: "ANNOTATION",
    title: "Seq annotation",
    tooltip: "Software to annotate genomics
        or transcriptomics data",
    children: [
        {
          tag: "ANNOTATION_GENOMICS",
          title: "Genomics"
        }
```

Fig. 3: Example of login and presentation page of a command in InGeBIOL.

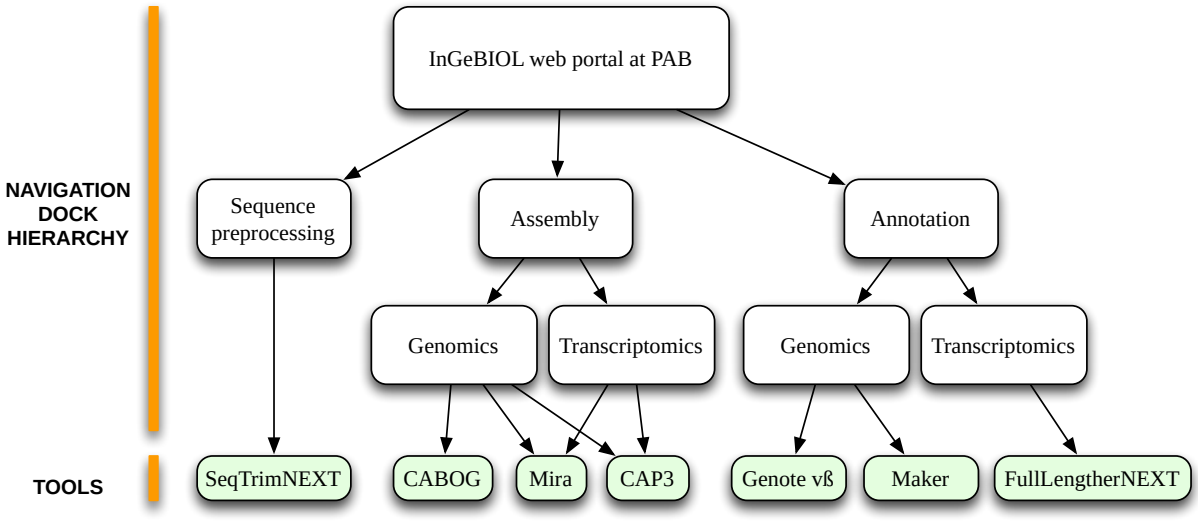


Fig. 4: Navigation dock hierarchy and tools classification of the InGeBIOL instance running at PAB.

```
        ,
        {
          tag: "ANNOTATION_TRANSCRIPTOMICS"
               ,
          title: "Transcriptomics"
        }
    ]
  },

  # second category
  {
    tag: "ASSEMBLY",
    title: "Assembly tools"
  }
]
```

A more complex hierarchy with three levels of categories is exemplified in Appendix A.4.4.

Every command defined in InGeBIOL is expected to be tagged with at least one category. Therefore, the command will appear on the branches corresponding to those tags. If no category (tag) is defined for a command, it is then not shown in the dock but it is still accessible by using its direct URL such as `https://your_ingebiol_site/sessions/new/Command_folder_name` (to know about command folder names, see subsection 3.3).

3.2.4 Data storage configuration

The file `globals.rb` within the folder "globals" (Figure 2) requires modifications to indicate where jobs (DATA_PATH) and private repositories (PRIVATE_DATA_PATH) are going to be stored. For example, the installation in the PAB environment requires that jobs and the private repository are saved in the $SCRATCH disk; thus, the corresponding environment variables are modified as follows:

```
# file: globals/globals.rb

# Path where jobs are going to be saved
DATA_PATH = '/mnt/scratch/ingebiol_data'

# Where to save private repository files
PRIVATE_DATA_PATH = '/mnt/scratch/
    ingebiol_priv'
```

3.2.5 Queue system

The job sent from the web interface of InGeBIOL is directed to the computer CPUs. If InGeBIOL is installed in a location with access to a supercomputer, as is the case in PAB, the job must use the corresponding queue system syntax. Therefore, it must be specified in the QSUB_CMD environment variable declared in `queue_system.rb` file within the "globals" folder (Figure 2). The minimum customisation requires to uncomment the corresponding environment variable (PBS and Slurm), while other queue systems require defining the command. In the PAB, the queue system Slurm is used, therefore, the modification will be the following:

```
# file: globals/queue_system.rb

# command used to send job to slurm
QSUB_CMD = '/usr/bin/sbatch'

# command used to send job to PBS
# QSUB_CMD = '/usr/pbs/bin/qsub'

# Command used to launch local jobs
LOCAL_CMD = 'bash'
```

Additionally, the shell used to execute local jobs must be declared in the environment variable LOCAL_CMD; by default, the most common shell for linux, bash, is declared. If any other shell is used, it can be specified in this variable. A complete description of parameters within this file is given in Appendix A.4.3.

3.3 Defining new commands

Each command handled by InGeBIOL gathers all configuration files within a folder inside the "commands" folder (Figure 2). As a quick example, we are going to describe the changes necessary to define a simple web interface for GENote β.1 [11]. The most important files are `common.json`, to control the name and appearance of the command on the presentation page (Figure 5A); one or more numbered `stageXX.json` files to define the input files and parameters that will be requested to the user in the submit form (Figure 5B) as well as the command line tools to be executed with submitted data; and a `final_results.json` file, where it is defined how to show on the web page the results obtained after the execution (Figure 6). As a result, adding a new tool to a running InGeBIOL installation is very easy and fast, as it only requires duplication of an already existing folder (for example the "demo" folder (Figure 2) corresponding to a generic command, see Appendix subsection A.3) and the modification of the desired parameters taking advantage that JSON files are human-readable.

3.3.1 Customising the command presentation page

The command line GENote β.1 [11] is defined by the files within the "genote" folder within the "commands" folder in Figure 2. The `common.json` configuration file serves to define the presentation page,

Fig. 5: Interface of the GENote $\beta.1$ command implemented in InGeBIOL: A) Login and presentation page, B) Job submission form shown after user login, composed of different input widgets.

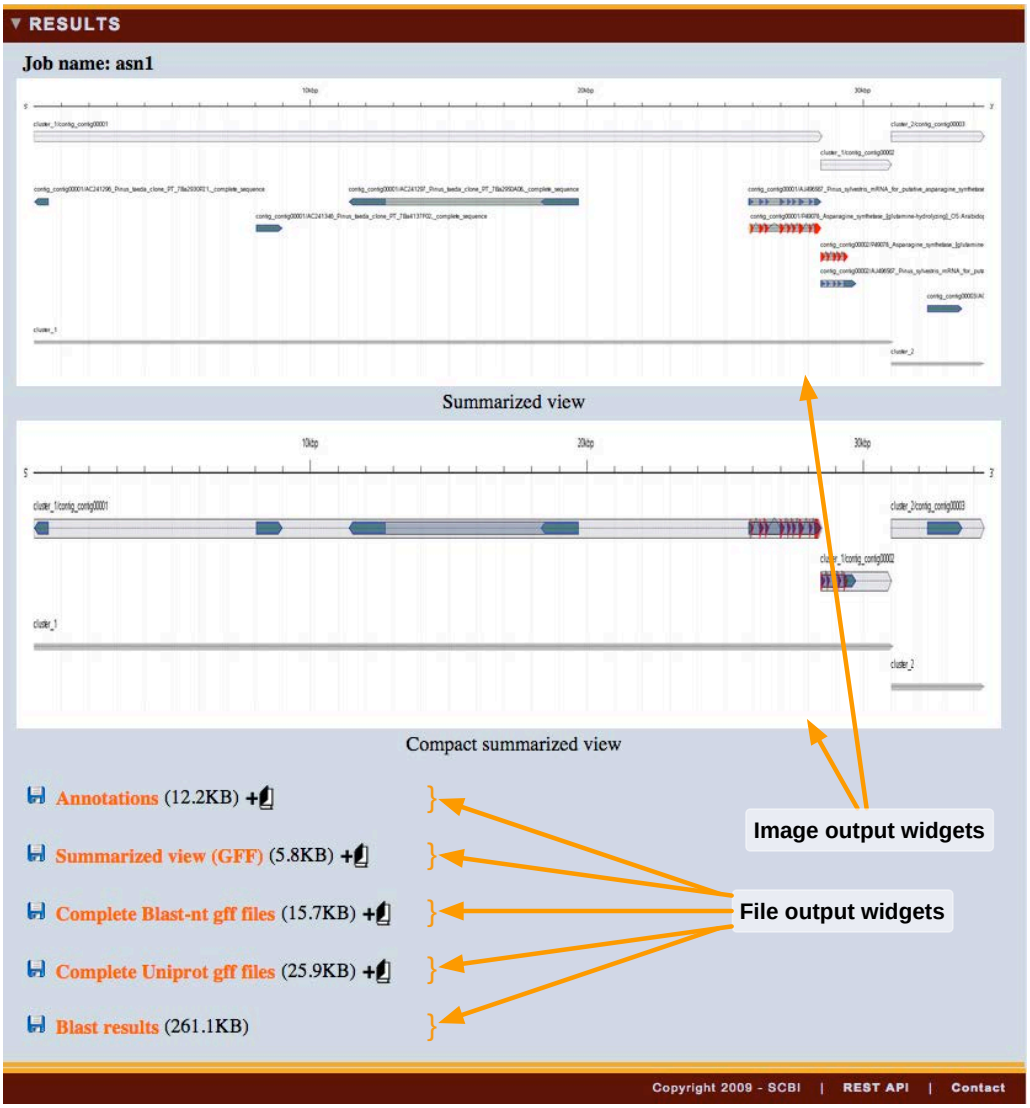


Fig. 6: Display of results for GENote β.1 using output widgets

as shown in Figure 5A, including name (`title`), descriptions (`short_description`), logotypes, etc., as well as the category or categories (`category_tags`) of the navigation dock into which this command will be included. Here it is the important part of the file whose modification customises the web page:

```
{
# Write here the desired title for the
    command
title: "GENote v.&beta;",

# Provide a short description
short_description: "A WEB INTERFACE FOR
    GENote",

# Command version
version: "v &beta;0.1",

# Show command in the dock
in_dock:'true',

# Under these categories:
category_tags: ['ANNOTATION_GENOMICS']
}
```

The complete set of parameters that can be defined in this file is detailed in Appendix subsection A.3.

3.3.2 Input data form

A command can be presented as a single-page HTML form (such as in Figure 5B) or as multiple forms shown in consecutive pages called 'stages'. Every stage is defined in its corresponding `stageXX.json` file (where XX are numbers used to sort the stage files and determine their presentation order). Each stage is comprised of one or more input widgets to capture input data from the user and one or more commands with the calls to the command line tools that will be sent for execution to the queue system. A definition in JSON code requires that a name (`title`) is given to the stage, the list of widgets is declared (`input_params`), the calls to command line tools is defined (`command_list`), and the use of a queue system and necessary resources are configured (`submit_file_header`). InGeBIOL can also submit jobs for execution to the local machine, but this is not a good practice since it could collapse the server and should only be used in exceptional situations (only with very fast commands). A generic example is as follows:

```
# stage template
{
# title for this stage
title:'SUBMIT FILES',

# list of input widgets
input_params: [
  INPUT_WIDGET_1,
  INPUT_WIDGET_2,
  ...
  INPUT_WIDGET_N
],

# commands to be executed with input
    widgets values
command_list:[
  COMMAND_LINE_CALL_1,
  COMMAND_LINE_CALL_2,
  ...
  COMMAND_LINE_CALL_M
],

# send this command_list to queue system
use_queue_system:true,

# request 6 cpus and 10gb for 10 hours to
    queue system
submit_file_header:[
  '#SBATCH --cpus=6',
  '#SBATCH --mem=10gb',
  '#SBATCH --time=10:00:00'
  ]
}
```

When users gain access to this command via web, they are required to fill at least the compulsory fields (in red) and then press "Send" button. If the error validation implemented in each widget encounters any issue, the user is notified to fix it. When no validation error is found, the command line calls, input data and parameters are converted to a queue system script and sent for execution. If two or more stages are defined for the command, they will appear successively in lexicographical order.

Defining input widgets: We call 'input widget' to a small piece of JSON that defines the widget properties using "parameter:value" tuples in this way:

```
# INPUT_WIDGET_1 definition
{
  "input_type": 'text',
  "id":'field1',
  "title":'Text field'
}
```

And renders this widget on screen:

These three properties in the listing are mandatory for all widgets and their meanings are the following:

- **input_type**: is the field type, selected from `text`, `integer`, `float`, `file`, `file_popup`, `checkbox`, `radio`, `popup` or `separator`.

- **id**: a unique identifier to distinguish between all input parameters of a stage. This identifier will be used to place each input widget value in the command line calls, by replacing any `$widget_id` found in the command line call definition with the corresponding value.

- **title**: this text string is shown on screen to inform the user about the widget content.

Widgets may also contain the following optional properties that present a default behaviour if they are not specified:

- **default_value**: The default value for the input widget that is shown in the submit form so that the user can modify it before submission. The use of default values can speedup user interaction with the submission form, and give them a hint about what is a valid value for this input widget.

- **required**: Widgets can be mandatory or optional. When *required* is true, the field content is checked by the validation engine, and the submission process is aborted if empty. If it is false or not defined, the widget is optional and can have an empty value and no error will be reported.

- **size**: It serves to indicate the width of the field in the form. If no size is specified, then the default value stablished by the HTML specification is used.

- **tooltip**: When specified, a small info icon is placed near the field, and the tooltip text is shown when the mouse is moved over it.

- **command_switch**: It defines how the value of this widget will replace its respective `$widget_id` placeholder when placed in the definition of a command line call. By default it will take the value `%s`, that indicates to replace the whole value of the widget without modifications. A value of `''-option %s''` will be replaced by `''-option widget_value''`, this way any command option or flag can be directly generated by the widget.

There are a few optional parameters with the purpose of input data validation in the case of `text`, `integer` or `float` input widgets:

- **validation_regex**: A Ruby regular expression that will be used to validate input data on `text` input widgets. If no value is specified, then no validation will be reported.

- **validation_lower_limit** and **validation_upper_limit**: For integer or float input widgets, these two parameters define the accepted range of input values. If no value is set for the lower limit, then no restriction is imposed for a minimum value. If no value is set for the upper limit, then no maximum value is forced.

- **validation_error_msg**: The error message that will be reported to the user if the validation fails.

A complete list of input widgets already defined in InGeBIOL can be found at Appendix subsection A.1. Of course, other new types of widgets can be defined.

The interface defined for GENote β.1 (Figure 5B) uses one stage with five input widgets: one `text` widget for the job name, two `integer` widgets for minimum identity and hit length, a `float` widget for minimum E-value and a `file_popup` widget to present a file selector to receive the input file.

Executing command-line tools: Usually, only one command line will be executed after a stage, but it could be the case that some selection requires the execution of several equivalent or different tools. Therefore, a list of tools to be executed is defined with a JSON snippet that contains the list of required files (the execution will be halted if any file listed on `required_files` is missing) and the `command` necessary to launch the tool or application.

In the listing below, the definition of the unique command line call necessary to launch GENote β.1 is shown, indicating in `required_files` that `input_file.fasta` is required for execution:

```
# COMMAND_LINE_CALL_1 definition
{
  required_files:['input_file.fasta'],
  command:'. ~genote/init_env; genote.rb
      input_file.fasta $evalue $min_ident
      $min_hit_length 6 > output.txt'
}
```

The defined command line call will be parsed and references to input widgets (`$evalue`, `$min_ident` and `$min_hit_length`) will be replaced by their values or `command_switch` expressions if present. For example, if input widgets for GENote β.1 were defined like this:

```
{
  input_type: 'integer',
  id:'min_ident',
  title:'Minimum identity (%)',
  command_switch:'-i %s',
```

```
  ...
}
,
{
  input_type: 'float',
  id:'evalue',
  title:'Minimum evalue (eg.: 1e-6)',
  ...
}
,
{
  input_type: 'integer',
  id:'min_hit_length',
  title:'Minimum hit length (nt)',
  command_switch:'-L %s',
  ...
}
```

If the user provides the value 90 for `min_ident`, 1e-6 for `evalue` and 15 for `min_hit_length` in their respective input widgets, the executed command line will be:

```
. ~genote/init_env; genote.rb input_file.
    fasta 1e-6 -i 90 -L 15 6 > output.txt
```

Submitting to a queue system: Execution of jobs in any queue system requires the definition of several parameters, such as number of CPUs, memory and execution time. Although they could be introduced by the stage form, we decided to fix these values as a `submit_file_header` variable on the stage configuration file.

In the use case of GENote β.1, we are requesting six cores and 10 GB of RAM with an estimated execution time of 10 hours:

```
submit_file_header:[
'#SBATCH --cpus=6',
'#SBATCH --mem=10gb',
'#SBATCH --time=10:00:00'
]
```

3.3.3 Output definition:

Alongside stage files, there is only one `final_results.json` file, and it is used by InGeBIOL to define the output of the command and render the results page (Figure 6) that is shown when the user clicks on the job name in the jobs list. This file contains a list with the necessary output widgets selected from the predefined ones (Appendix A.2), or new, customly-defined ones. A generic example of `final_results.json` can be:

```
{
# title of the final results
title:'RESULTS',
# define a list of output widgets
params: [
  OUTPUT_WIDGET_1,
  OUTPUT_WIDGET_2
]}
```

where a title is given, and the `params` variable contains a list of the required output widgets. An output widget is defined as follows:

```
# OUTPUT_WIDGET_1 defining an output file
{
 type: 'file',
 file:'output_files/annotations.txt',
 title:'Annotations',
 tooltip:''
}
```

where the items have the same meaning than in the input widgets, and is rendered on screen with this appearance:

The disk icon near the file name allows the user to download the file (the same as clicking on the name), while the bookmark icon will add the file to the user file repository.

The execution of GENote β.1 provides images and text files Figure 6. Therefore, the output stage contains two `image` ouput widgets that directly show the images on the web page, and five `file` ouput widgets that show the name of each file.

3.3.4 REST web service

Since web services are now widely used as a way to support interoperable machine-to-machine interactions and REST (Representational State Transfer) has gained widespread acceptance across the web and has been adopted by mainstream web 2.0 service providers [4], InGeBIOL automatically adds REST compatibility [3] to every configured command. Therefore, all commands accessible via web can also be integrated with other tools that use REST web services to launch jobs, query job status, download result files, or any of the predefined actions.

REST web service protocol, directly based on HTTP requests, is faster and easier to use than the overloaded SOAP based ones [23], but InGeBIOL makes it even easier because, for each command, users can find a specific help page with the parameters and calling examples for a simple REST client like the `curl` command line application (Figure 7).

REST API for genote

Fields for stage 0

Field	Help	Required	Type
-F evalue_field=VALUE	The minimum evalue required to..., in percentage.	true	float
-F job_name_field=VALUE	The job is assigned a number automatically, but you can provide a custom name for it	true	text
-F input_file_field=@PATH_TO_FILE	PATH_TO_FILE must be the path to the local file that you want to upload or a global file name. Select a sequence file in fasta format	true	file_popup
-F api_login_key=your@email.com	User email	true	
-F min_hit_length_field=VALUE	The minimum length of a hit to be considered by genote (nucleotides).	true	integer
-F min_ident_field=VALUE	The minimum identity required to..., in percentage.	true	integer

Submit a job using a REST client like curl

Sending a new job and returns either errors or the JOB_ID if it was successfully uploaded

curl -i -X **POST**
-F min_ident_field=VALUE
-F min_hit_length_field=VALUE
-F api_login_key=your@email.com
-F input_file_field=@PATH_TO_FILE
-F job_name_field=VALUE
-F evalue_field=VALUE
http://www.scbi.uma.es/ingebiol/commands/genote/jobs/0/stage/0.json

Job list

Retrieving the complete job list

curl -i -X **GET** -F api_login_key=your@email.com http://www.scbi.uma.es/ingebiol/commands/genote/jobs.json

Job status

Retrieving status of a job

curl -i -X **GET**
-F api_login_key=your@email.com
http://www.scbi.uma.es/ingebiol/commands/genote/jobs/JOB_ID

Complete job results

Downloading the complete result set of a job

curl -i -X **GET**
-F api_login_key=your@email.com
http://www.scbi.uma.es/ingebiol/downloads/genote/JOB_ID

Single file from job results

Downloading a single file from the complete result set of a job

curl -i -X **GET**
-F api_login_key=your@email.com
http://www.scbi.uma.es/ingebiol/downloads/genote/JOB_ID/ONE_FILE_OR_FOLDER

Deleting a JOB

Delete a job from the job pool

curl -i -X **DELETE**
-F api_login_key=your@email.com
http://www.scbi.uma.es/ingebiol/downloads/genote/JOB_ID.json

Fig. 7: Generated REST API documentation for the customised command of GENote β.1

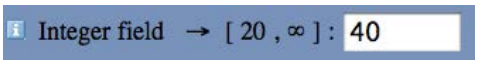
3.4 Interactive user interface

Web interfaces used to be static, but the arrival of AJAX has pave the way to produce interactive web interfaces that can resemble in some way the menu-based software. Here it is explained how to exploit interactivity in InGeBIOL simply adding some lines to the input widget definition in the stage file.

Self-validation: As stated previously (section 3.3.2), InGeBIOL comes with automatic input data validation out of the box. This self-validation can be based on regular expressions (`text`), or value-range definition (`integer` or `float`). Its main advantage is to avoid errors in the execution pipeline due to user mistakes. Defining a range can also give users a clue about valid values for the input widget because the valid range is shown on the interface. For example, if we define a range of values for a integer to be checked adding the following lines in the widged definition:

```
# Input widget with range validation
{
  type:'integer',
  ...
  validation_lower_limit:'20',
  validation_upper_limit:''
}
```

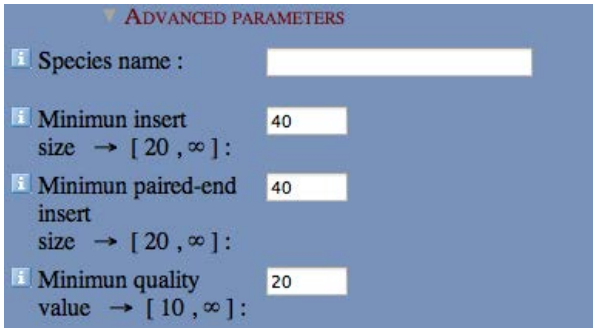
the user will see this limits in the interface:



Regular expression validation can force the user to specify a controlled string. For example, if a field is expected to contain only nucleotides, it can be stated as follows:

```
# Input widget with RE validation
{
  type:'text',
  ...
  validation_regex: '[ACTGactg]*',
  validation_error_msg: 'You must provide a
      valid nucleotide sequence (ACTG)'
}
```

Show and hide parameters: When the interface contains many optional parameters, having all of them in view can be annoying. To prevent this, InGeBIOL can group input widgets into multiple optional groups that will be shown grouped to the user with a button to dynamically toggle their visibility as in the following image:



This can be done by adding a `separator` input widget, with a `title` to be shown on screen and a `folding_name` that will serve to tag other widgets. The default state of this separator is declared on `folding_closed`.

```
# Define a folding separator initially
    closed
{
  input_type: 'separator',
  title:'Advanced parameters',
  folding_name:'advanced',
  folding_closed: true
}
```

As a result, the title with the toggle button is shown on screen (as in the previous image). To control the widgets that will toggle their presence with this separator, the widget must include the field `folding_name` in its definition:

```
# input widget with folding name
{
  input_type: 'text',
  id:'species',
  folding_name : 'advanced',
  title:'Species name',
  ...
}
```

Additionally, individual input widgets can be hidden/shown (without being tagged with any `folding_name`) based on any other input widget value (for example a `popup` selector). This is done using the `visible_if` parameter into the widget definition:

```
# input widget visible only when using the
    fasta option
{
  input_type: 'file_popup',
  id:'quality_file',
  visible_if:'"$file_type"=="fasta"'
  ...
}
```

Execution templates: A command definition can have a conditional execution expression that will be evaluated after replacing the widget placeholders (`$widget_id`) with their values. If the result of evaluating the expression is true, the command line call protected by it will be executed, but if the result is false, its execution will be skipped. This feature enables the definition of execution templates (sets of predefined parameters) by setting up a `popup` input widget with the different available templates, and then using the selected template to conditionally execute the command line application with different predefined parameters.

In the listing below, a `popup` input widget with two assembly options (Unitigger or Best Over Graph) is defined:

```
# popup input widget with two execution
    templates
{
  input_type: 'popup',
  id:'assembly_method',
  title:'Assembly methods',
  default_value: 'bog',
  values: {'Unitigger':'utg',
           'Best Over Graph':'bog'},
  ...
 }
```

This code will show on screen a popup menu to choose one of the available options:

where the option selected on the popup will refer to one of the possibilities of the command line call definitions. In other words, the `exec_if` can use its value to conditionally execute the commands.

The analysis of this widget requires the definition of two command line calls, one to be be executed when `$assembly_method` input widget have the value "bog" (that means that the user has selected the "Best Over Graph" method on the popup), and the other will be executed when `$assembly_method` contains the value "utg" corresponding to the selection of the "Unitigger" assembly method.

```
# this command line call will be executed
    when bog is selected on the popup
{
  exec_if:'"$assembly_method"=="bog" '
  required_files:['pname.frg'],
  command:'runCA-OBT.pl -unitiger=bog
      createACE=1 pname.frg'
},
# this command line call will be executed
    when utg is selected on the popup
{
  exec_if:'"$assembly_method"=="utg" '
  required_files:['pname.frg'],
  command:'runCA-OBT.pl -unitiger=utg
      createACE=0 pname.frg'
}
```

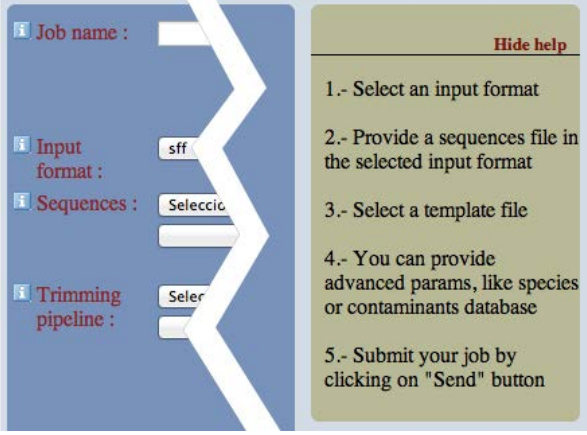
Interative help: One optional help file in HTML or text format can be used to give the user information about the usage of the command. It will be shown in an independent panel side by side to the submit form. The user can dismiss or recall it when necessary. The help file must be placed in the same folder as the stage definition file, and its name specified on the stage definition file by using the `help_file` variable, as stated in the listing below:

```
# stage template with help file
{
  # title for this stage
  title:'SUBMIT FILES',
  help_file: 'help.html',

  # list of input widgets
  input_params: [
    INPUT_WIDGET_1,
    INPUT_WIDGET_2,
  ]
  ...
}
```

In this example, the HTML content of the `help.html` file that is shown in the following listing:

```
1.- Select an input format
<br/>
<br/>
2.- Provide a sequences file in the
    selected input format
<br/>
<br/>
3.- Select a template file
<br/>
<br/>
4.- You can provide advanced params, like
    species or contaminants database
<br/>
<br/>
5.- Submit your job by clicking on "Send"
    button
```

will render a help screen with five steps indicating how to fill the submit form:



3.5 Input and output layouts

The layout of input and output widgets is automatically organised without any user intervention. However, it has been implemented an extra level of interface customisation making use of layout templates that serve to define how input or output widgets should be placed on screen.

To use a custom template, a `template` variable pointing to the template file must be specified on the `stage.json` or `final_results.json` files, depending if the template will be used for input or output widgets respectively. In the listing below:

```
# using an RoR HTML template file in a
    stage configuration
{
  title:'SUBMIT FILES',
  stage_type:'submit',
  enabled: true,
  template:'stage1.html.erb',
  ...
```

a `template` file `stage1.html.erb` is defined for input widgets to declare a table with two columns to place two widgets side by side. It must be in the "commands" folder (Figure 2), and can contain any valid HTML code. Each `<%= html['widget_id'] %>` snippet inside the template will be replaced by the code necessary to show the widget referenced by `widget_id`:

```
<table border="0" width="100%">
  <tr>
    <td>
      <%=html['widget_id_1'] %>
    </td>
    <td
      <%=html['widget_id_2'] %>
    </td>
  </tr>
</table>
```

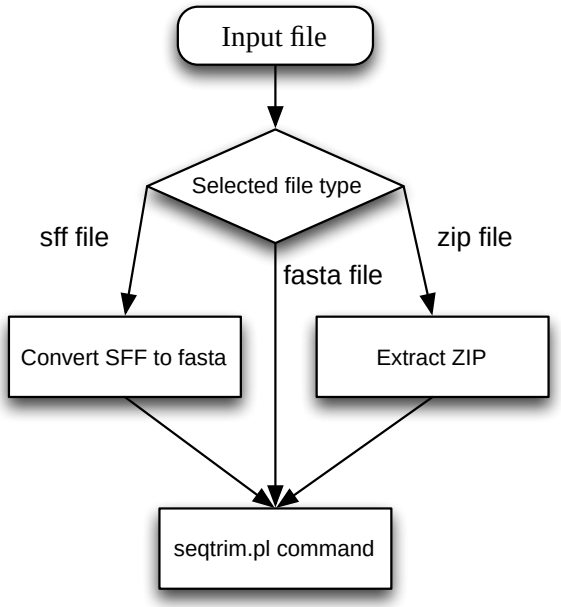
3.6 Workflows can be implemented

The current implementation of InGeBIOL is not only able to provide web interfaces to simple command-line tools and gather them as a portal: workflows and pipelines can also be given web interfaces. It is recommended to describe complex workflows in dedicated tools like Conveyor [20] or Autoflow [26], and then use their command line execution capabilities to call them from InGeBIOL using workflows as if they were a black box.

We mentioned (section 3.4) that the `command_list` is an array of command elements defined in the following way:

```
{
  exec_if:'conditional expression',
  required_files:['list_of_required_files
      '],
  command:'command_to_be_executed'
}
```

where the optional `exec_if` field is a piece of Ruby code that will be evaluated (it can contain references to values of any input widget of the stage), and the associated command line will be executed only when the evaluation returns true. This feature can be used to control the execution path in simple workflows, for example the simple workflow outlined in this schema:



where the command line to "convert SFF" will only be executed if the input type is a `SFF` file, and the command line to "extract ZIP" will be executed if the selected input file is a `zip` file. The result of both alternative branches is a file of sequences in fasta format to feed the `seqtrim.pl`

command line. In any other case, the input file is assumed to be in fasta format and the execution of the `seqtrim.pl` command is done without previous transformations. This sample workflow is specified with the following code:

```
# extract SFF only if input file is SFF
{
  exec_if:'"$input_type_popup_sq"=="sff"',
  required_files:['seqs.sff'],
  command:'~mira/sff_extract -o seqs seqs.
      sff'
},

# extract if input file is a zip file
{
  exec_if:'"$input_type_popup_sq"=="zip"',
  required_files:['seqs.zip'],
  command:'unzip seqs.zip'
},

# always execute final command
{
  required_files:[],
  command:'seqtrim.pl seqs > st_output.txt'
}
```

4 Conclusions

InGeBIOL is able to provide a web interface for command line tools with integrated job management and queue system support for submitting jobs towards supercomputers, handling previous sent jobs and browsing their results (section 3.3.2). It simplifies the use of commands with many input parameters by means of an interactive user interface (subsection 3.4). The automatic implementation of REST web service compatibility for every command boost programmers performance when they need to provide web services for their applications (subsubsection 3.3.4). The LDAP authentication helps system administrators since users can authenticate with the same user-password combination used for supercomputing or other centralised facilities (subsubsection 3.2.2). The use of RoR helped in the implementation of a modern and easy-to-use framework for inexperienced users (section 2.

InGeBIOL can also be used with legacy software or new algorithms, saving programmers's time dedicated to interfaces (up to 50% of the programming time in some cases [22]) and allowing to concentrate all efforts in algorithm and not in usability. As a result, it has been utilised to develop interfaces to promote new algorithms developed as command line tools, such as FullLenghterNEXT, Genote, and SeqTrimNEXT, but also to provide interfaces to different command line applications and workflows related to bioinformatics (MIRA [8], CAP3 [16], CABOG [21], Maker [6]).

The capability of defining tools that will not appear in the dock is very useful since it allows to have multiple versions of the interfaces (subsubsection 3.2.3). One stable version can be exposed to final users, while others can be kept hidden for those tools that are still under development, or in beta testing phases.

Acknowledgement

The authors thankfully acknowledge the computer support provided by the Plataforma Andaluza de Bioinformática of the University of Málaga. This research was supported by co-funding by the European Union through the ERDF 2014-2020 "Programa Operativo de Crecimiento Inteligente") to the projects P10-CVI-6075 of the regional Plan Andaluz de Investigación, Desarrollo e Innovación, and the RTA2013-00068-C03 of the Spanish INIA and MINECO.

A Appendices

A.1 Types of predefined input widgets

All input widgets are defined in a stage configuration file, and are represented by a simple hash of keys and values defined in a JSON-like format. Eg.: see stage0001.json in section A.3. The `demo` command suplied with InGeBIOL contains a `stage.json` file with all possible input widgets ready to use. In the following list, the complete list with their appearance on screen is shown:

1. **Text widget**: it displays a text field where the user can provide some text that can be validated with a regular expression. It is defined by the following code:

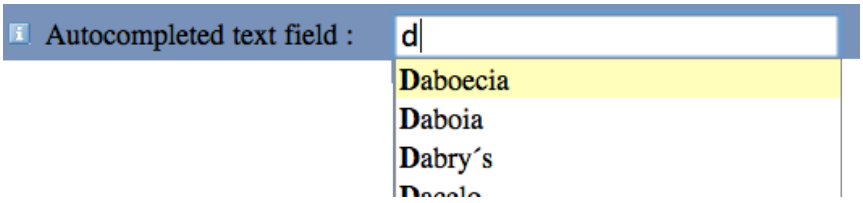
```
{
  "input_type" : "text",
  "id" :"job_name",
  "title" :"Text field",
  "default_value" : "",
  "required" : true,
  "size" :"30",
  "tooltip" :"The job is assigned a name automatically, but you can provide a custom
      one for it",
  "validation_regex" : "",
  "validation_error_msg" : "",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

   and produces the following widget on screen:

2. `Text widget with autocomplete`: it declares text widgets that can be autocompleted with a set of predefined values. It is defined by the following code:

```
{
  "input_type" : "text",
  "id" :"genus",
  "title" :"Autocompleted text field",
  "default_value" : "",
  "required" : false,
  "size" :"30",
  "tooltip" :"Contaminants within this species will be ignored",
  "validation_regex" : "",
  "validation_error_msg" : "",
  "autocomplete_values" :"seqtrimnext/genus.txt",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

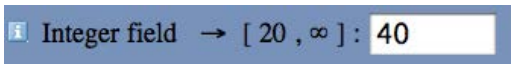
   and produces the following widget on screen:



3. **Integer widget**: it is used to request an integer numeric value that can be validated using a range. It is defined by the following code:

```
{
  "input_type" : "integer",
  "id" :"min_insert_size",
  "title" :"Integer field",
  "default_value" : "40",
  "required" : false,
  "tooltip" :"Final inserts below this limits will be rejected",
  "validation_lower_limit" :"20",
  "validation_upper_limit" :"",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

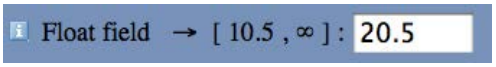
and produces the following widget:



4. **Float widget**: it is equivalent to the integer widget but for float values, maintaining the same validation options with ranges. It is defined by the following code:

```
{
  "input_type" : "float",
  "id" :"min_quality_value",
  "title" :"Float field",
  "required" : false,
  "tooltip" :"Zones with quality value below this limit will be rejected",
  "default_value" : "20.5",
  "validation_lower_limit" :"10.5",
  "validation_upper_limit" :"",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:



5. **Popup widget**: it displays a set of predefined options in a popup menu. The user can select only one of the options. It is defined by the following code:

```
{
  "input_type" : "popup",
  "id" :"file_type",
  "title" :"Popup menu",
  "default_value" : "value2",
  "values" : {"Option 1":"value1",
        "Option 2":"value2",
        "Option 3":"value3"}
            ,
  "required" : true,
  "tooltip" :"Select file type",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:

6. **File widget**: it displays a file chooser to upload a file. A limit for the uploaded file size can be declared. It is defined by the following code:

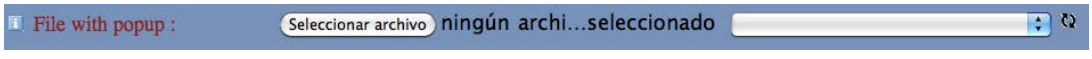
```
{
  "input_type" : "file",
  "id" :"sequence_file",
  "title" :"File selector",
  "default_value" : "",
  "filter" :"*",
  "required" : true,
  "save_as" :"seqs",
  "max_file_size" : 50,
  "tooltip" :"Select a sequence file in fasta format",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:

7. **File selector with repository popup**: it displays a traditional file chooser, combined with a popup menu with files stored on both, the global and user private repositories. Files shown in the popup are filtered by a search expression. It is defined by the following code:

```
{
  "input_type" : "file_popup",
  "id" :"template_file",
  "title" :"File with popup",
  "default_value" : "",
  "filter" :"templates/*.txt",
  "global_filter" :["~seqtrimnext/programs/templates/*.txt","~/progs/ruby/seqtrimii/
      templates/*.txt"],
  "required" : true,
  "save_as" :"params.txt",
  "max_file_size" : 10,
  "tooltip" :"Select a cleaning template or provide a custom one.",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:



8. **Radio button widget**: it displays a set of options as radio buttons, and let the user select only one of them. It is defined by the following code:

```
{
  "input_type" : "radio",
  "id" :"workflow_type_radio",
  "title" :"Radio button group",
  "default_value" : "first_value",
  "values" : {"First title":"first_value",
        "Second title":"second_value"},
  "required" : true,
  "tooltip" :"Select the type of workflow you want to use",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:

9. **Checkbox widget**: it displays a radio button that the user can use to set a boolean parameter. It is defined by the following code:

```
{
  "input_type" : "checkbox",
  "id" :"workflow_type_radio",
  "title" :"Checkbox",
  "default_value" : "seqtrim",
  "required" : false,
  "tooltip" :"Select the type of workflow you want to use",
  "folding_name" : "advanced",
  "visible_if": "",
  "command_switch" :"%s"
}
```

and produces the following widget:

A.2 Types of predefined output widgets

The `demo` command supplied with InGeBIOL contains a `final_results.json` file with all possible output widgets ready to use. In the following list, the complete list with their appearance on screen is shown:

1. **File**: it shows a result file, with a download link and bookmark button to store it on the user file repository. It is defined by the following code:

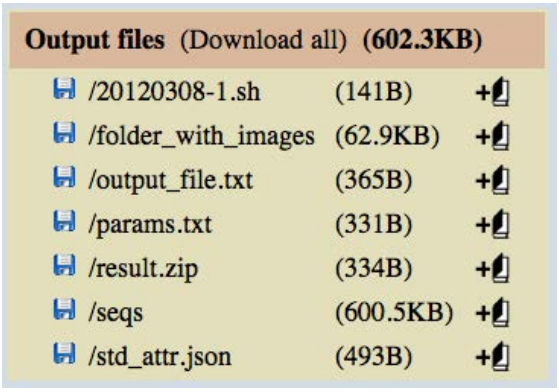
```
{
    "id" : "one_file",
    "type" : "file",
    "file" :"seqtrim_result.zip",
    "title" :"Preprocessing results",
    "tooltip" :""
}
```

and produces the following output widget:

2. **File list**: it serves to show multiple files in a list with only one level, each of them with their respective download and bookmark button. A `Download all` link is also shown. It is defined by the following code:

```
{
    "id" :"output_files",
    "type" : "file_list",
    "file" :"output_files/*",
    "title" :"Output files",
    "tooltip" :""
}
```

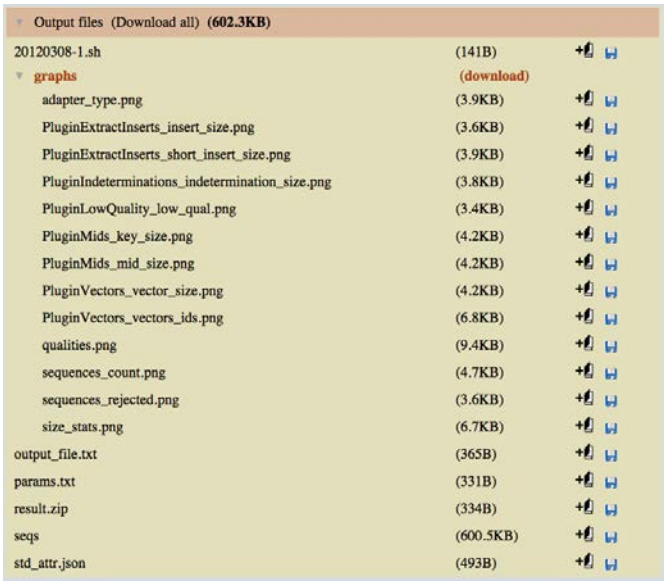
and produces the following output widget:



3. **File browser**: it displays a hierarchical file browser to recursively show all result files and folders contained on a parent directory. It is defined by the following code:

```
{
    "id" :"output_files_browser",
    "type" : "file_browser",
    "file" :"output_files",
    "title" :"Output files",
    "show_closed" : true,
    "tooltip" :""
}
```

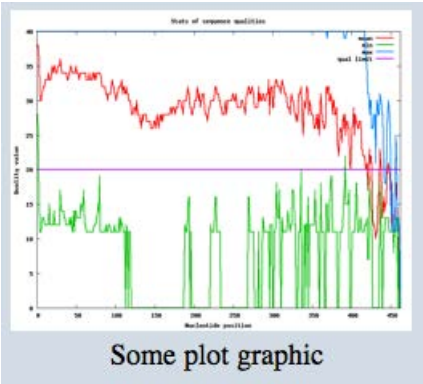
and produces the following output widget:



4. **Image**: it serves to display a single result image in a determined size. It is defined by the following code:

```
{
    "id" :"image1",
    "type" : "image",
    "file_name" :"image_file.png",
    "content_type" :"image/png",
    "title" :"Some plot graphic",
    "options" :{"height":"200","width":"250"},
    "linked" :true,
    "tooltip" :""
}
```

and produces the following output widget:

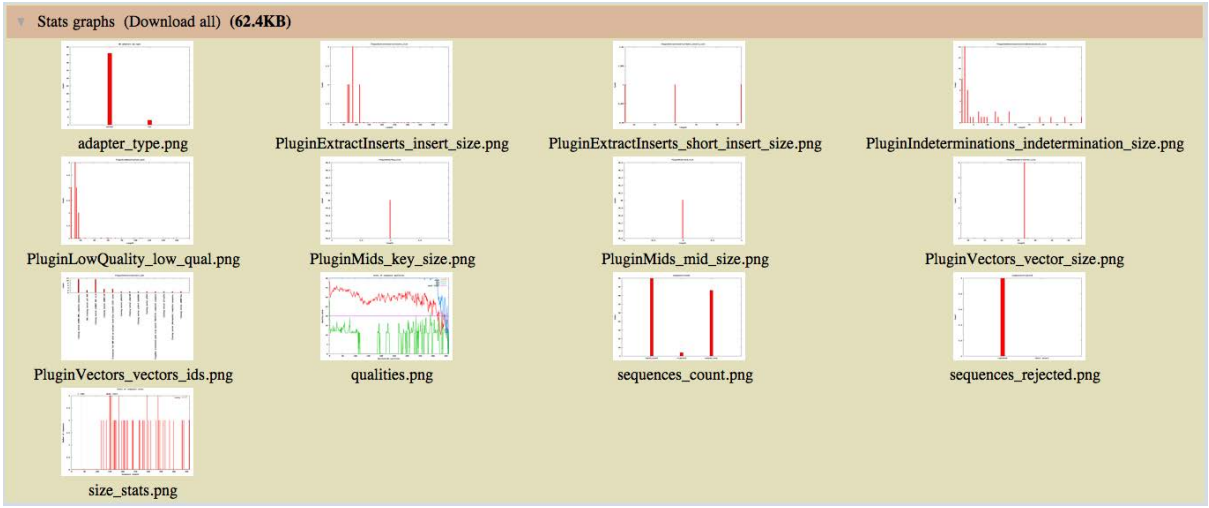


5. **Image browser**: it displays a set of images in a single browser panel, useful when an undetermined number of images need to be shown. It is defined by the following code:

```
{
    "id" :"stat_images",
    "type" : "image_browser",
    "title" :"Stats graphs",
    "file" :"graphs/*",
    "content_type" :"image/png",
    "options" :{"height":"100","width":"150"},
```

```
    "max_images" :0,
    "columns" : 4,
    "linked" :true,
    "show_names" :true,
    "ignored_images" : ["any_file_name.png"]
}
```

and produces the following output widget:



6. **Command output**: it can display the output text obtained after executing a command line. This is not designed to execute full commands here, but to show excerpts of result files using fast commands such as `head`, `tail`, `wc`, `grep`, etc.. It is defined by the following code:

```
{
    "id" : "output1",
    "type" : "command_output",
    "command" :"du -sh *",
    "title" :"Generated files",
    "tooltip" :""
}
```

and produces the following output widget:

Generated files :

```
4,0K 20120308-1.sh
0B folder_with_files
92K folder_with_images
4,0K output_file.txt
4,0K params.txt
4,0K result.zip
604K seqs
4,0K std_attr.json
```

A.3 Implementing a new command

Here it is described a step-by-step implementation of a new command within InGeBIOL. Every command has everything within the configuration folder `config/commands/` that contains a few JSON configuration files defining the input widgets, command-ine calls to be executed and output widgets. When InGeBIOL

is installed, it will come with a `demo` command that will not appear by default in the dock, but that can be accessed at `http://your_ingebiol_site/ingebiol/session/new/demo`). Installation of the very first command can be done simply by duplicating this "demo" folder within the "commands" folder, changing the folder name and editing the JSON files contained in it. For example, the customisation of a command that converts all text found on a file to uppercase using the `tr` linux command is illustrated. The `tr` command simply translates one character into another and prints the result on the screen. Hence, the following command:

```
echo abcabc | tr a A
```

will output the word AbcAbc, because it changes all the lowercase "a" chars with the uppercase version: "A".

To convert all downcase chars to uppercase, we could specify all chars, the range of chars, or use the special lowercase *class* denoted by `[:lower:]` , and its uppercase equivalent `[:upper:]`:

```
echo abcabc | tr [:lower:] [:upper:]
```

that will produce ABCABC, but since we need to convert whole files, the full command will have to receive an `input_file.txt` and output an `output_file.txt` taking advantage of standard input and output redirections:

```
tr [:lower:] [:upper:] < input_file.txt > output_file.txt
```

The third command line call above will be used in the example outlined in the following steps:

1. Go to "commands" folder: `cd config/commands`
2. Duplicate the "demo" folder changing its name to 'uppercase': `cp -r demo uppercase`
3. Go to "uppercase" folder: `cd uppercase`
4. Edit `common.json` file and change the values of the `title`, `short_description` and `long_description` variables to change the appearance of the presentation page used for the command, and set `in_dock` to true so that it is shown on the dock (subsubsection 3.3.1):

```
############################################################
# Common parameters of command
# ---------------------------------------------------------
# You only can modify the values of each key.
#
# Comments must be in their own line preceded with a "#"
# symbol.
#
############################################################

{

  # Write here the desired title for the command
  "title" :"Uppercase converter",

  # Provide a short description
  "short_description" :"A demo interface for UPPERCASE using InGeBiol",


  # And a long one
  "long_description" :"Long description for command",


  # Command version
  "version" :"v 0.9a1",


  # Contact url
```

```
  "contact_url" :"http://www.scbi.uma.es",


  # Corporate logo name, file must be in public/images
  "corporate_logo" :"logogrande.png",


  # Command logo name, file must be in public/images
  "command_logo" :"ingebiolBig.png",


  # Small corporate logo name, file must be in public/images
  "small_corporate_logo" :"logoscbi.png",


  # Small command logo name, file must be in public/images
  "small_command_logo" :"ingebiolSmall.png",

  # Copyright
  "copyright" :"Copyright 2009 - SCBI",

  "in_dock" : true,

  "category_tags": ['ASSEMBLY'],

  # Download links shown under login box
  links:[
  {
    text:'Download',
    url:'http://www.scbi.uma.es/downloads'
  }
  ],

  # Articles shown at bottom of the page
  "articles_title":'If you find this command useful, please cite:',

  "articles":[
  {
    text:'Name of the article',
    url:'http://url_of_the_article'
  }
  ]
}
```

5. Edit the `input_params` variable of the `stage0001.json` file to add a `text` input widget for the job name, and a `file_popup` input widget to receive the `input_file.txt` that will be converted to uppercase (subsubsection 3.3.2). Add the previously discussed `tr` command line call to the `command_list` variable in the same stage file to convert the `input_file.txt` to uppercase and generate the `output_file.txt` (section 3.3.2):

```
############################################################
# Common parameters of command
# ---------------------------------------------------------
# You only can modify the values of each key. Be careful
# with punctuation symbols, since this is a json file and
# needs to keep its structure.
#
# Comments must be in their own line preceded with a "#"
# symbol.
#
# You can add as many stages as you want, they are shown
# in alphabetical order.
#
############################################################

{
"title" :"SUBMIT FILES",
```

```
"stage_type" :"submit",
"enabled" : true,
"help_file" : "help.html",
"submit_button_title" : "Next",

"input_params" : [
 # text input widget for job_name
 {
  "input_type" : "text",
  "id" :"job_name",
  "title" :"Text field",
  "default_value" : "",
  "required" : true,
  "size" :"30",
  "tooltip" :"Provide a custom name for the job",
  "validation_regex" : "",
  "validation_error_msg" : "",
  "standard_attribute" : true,
  "save_in_presets" : true
 },

 # file popup input widget for input_file.txt
 {
  "input_type" : "file_popup",
  "id" :"input_file",
  "title" :"File with popup",
  "default_value" : "",
  "command_switch" : "%s",
  "filter" :"templates/*.txt",
  "global_filter" :[""],
  "required" : true,
  "save_as" :"input_file.txt",
  "max_file_size" : 1,
  "tooltip" :"Any hint"
  }

],

"command_list" :[
  # command line call used to convert input_file.txt to uppercase and store it on
      output_file.txt
  {
    "required_files" :[],
    "command" :"tr [:lower:] [:upper:] < input_file.txt > output_file.txt"
  }
]

,
"use_queue_system" :false
}
```

6. Edit `final_results.json` adding to the `params` variable a `file` output widget to let the user download the `output_file.txt`, and a `command_output` widget to show on screen the first 20 lines of the same `output_file.txt`.

```
#################################################################
# Final results of command
# ---------------------------------------------------------------
# You only can modify the values of each key.
#
# Comments must be in their own line preceded with a "#"
# symbol.
#
#################################################################

{
"title" :"SELECTED RESULTS",
"stage_type" :"final_results",
```

```
"enabled" : true,
"template" :"",

"params" : [
 # file output widget to show a download link to output_file.txt
 {
  "type" : "file",
  "file" :"output_file.txt",
  "title" :"Output file in uppercase",
  "tooltip" :""
 },

 # command output to show 20 first lines of output_file.txt
 {
  "type" : "command_output",
  "command" :"head -n 20 output_file.txt",
  "title" :"Head of output file",
  "tooltip" :""
 }
]

}
```

7. It can be tested by pointing the browser to `http://your_ingebiol_site/ingebiol/session/new/demo`. If there is any error in configuration files, an error message will appear instead of the web interface of the command. If configuration files are correct, a presentation page similar to the Figure 5A will be shown. There, the user can login as a guest introducing any valid e-mail, write a job name, upload a file, and send the job. It will appear on the job list section, where its name can be clicked to view the results page including a download link with the `output_file.txt` file and the twenty first lines of it.

A.4 Global configuration

InGeBIOL have a set of features that affect to the integration with existing environments. Global configuration files are located at */home/rails/ingebiol/config/global* folder (this may change if you installed it on another location).

A.4.1 File: globals.rb

This is a ruby file used to define global configuration parameters that affect to all interfaces.

Example file with comments:

```
# This file defines global configuration parameters

# Path where jobs are going to be saved
DATA_PATH = File.expand_path(File.join(BASE_PATH,'../','guidata'))

# Path where config files are saved
CONFIG_PATH = File.expand_path(File.join(BASE_PATH,'../','config'))

# Where to save repository files
PRIVATE_DATA_PATH = File.expand_path(File.join(BASE_PATH,'../','guidata','private'))

# Path of commands configurations
COMMAND_CONFIG = File.expand_path(File.join(CONFIG_PATH,'commands'))

# Path of globals configuration
GLOBALS_CONFIG = File.expand_path(File.join(CONFIG_PATH,'global'))

# Path to user scripts
USER_SCRIPTS_PATH = File.expand_path(File.join(CONFIG_PATH,'scripts'))

# How are stages numbered
STAGES_PATTERN = 'stage*.json'
```

```
# Standard attributes are saved in a file with this name
STANDARD_ATTR_JSON = 'std_attr.json'

# This script is used to get additional info about a job
GET_JOBINFO_SCRIPT = 'get_job_info.rb'

# These are the titles shown in the job list in the web interface
JOBLIST_TITLES_JSON = 'joblist_titles.json'

# This tag is replaced by all command swithes of every input param
# when it is found in a command description
COMMAND_SWITCHES_TAG = '$COMMAND_SWITCHES'

# This is the default command to show when no one is defined
DEFAULT_COMMAND = 'login'
```

A.4.2 ldap.rb

This is a ruby file used to define the values of the LDAP authentication server and related configuration:
Example file:

```
# This file is used to define the values of the LDAP
# authentication server and related configuration:

# set to true to use LDAP authentication
USE_LDAP_AUTH = false

# set to true to use LDAP authentication while allowing GUEST access
USE_MIXED_AUTH = true

#LDAP authentication server
LDAP_HOST = '10.200.190.1'

# How to query users to LDAP server (may change between OpenDirectory and OpenLDAP
    servers)
LDAP_USERS_UID='uid=%s, ou=people, dc=mylab, dc=uma, dc=es'
```

A.4.3 queue_system.rb

This is a ruby file used to define the commands used to submit jobs to queue systems:
Example file:

```
RUNNING_FILE = 'RUNNING'
QUEUED_FILE = 'QUEUED'

# command used to send job to slurm
QSUB_CMD = '/usr/bin/sbatch'

# command used to send job to PBS
# QSUB_CMD = '/usr/pbs/bin/qsub'
QSUB_CMD = '/usr/pbs/bin/qsub -q routex86'

#sudo command to be used with QSUB_CMD if needed
QSUB_SUDO = '/usr/bin/sudo -u bioperl'

# Command used to launch local jobs
LOCAL_CMD = 'bash'
LOCAL_SUDO = ''
```

A.4.4 categories.json

```
# This file defines an hierarchy of categories. Then each program/interface can be
    tagged to belong to any of these categories.
#
# Programs not tagged with a valid category are found under the "BAD CATEGORY" title.

[
  {
    "tag" :"PREPROCESS",
    "title" :"Seq pre-processing",
    "tooltip" :"Software to preprocess raw sequences prior to assembly"
  }
  ,
  {
    "tag" : "ASSEMBLY",
    "title" :"Seq assembly",
    "tooltip" :"Software to assembly already preprocessed genomics or transcriptomics
        data",
    "children" : [
        {
          "tag" :"ASSEMBLY_GENOMICS",
          "title" :"Genomics",
          "children" :[
            {
              "tag" : "SHORT_SEQUENCES",
              "title": "Using short sequences"
            },
            {
              "tag" : "LONG_SEQUENCES",
              "title": "Using long sequences"
            }
          ]
        }
        ,
        {
          "tag" :"ASSEMBLY_TRANSCRIPTOMICS",
          "title" :"Transcriptomics"
        }
    ]
  }
  ,
  {
    "tag" :"ANNOTATION",
    "title" :"Seq annotation",
    "tooltip" :"Software to annotate genomics or transcriptomics data after assembly",
    "children" : [
        {
          "tag" :"ANNOTATION_GENOMICS",
          "title" :"Genomics"
        }
        ,
        {
          "tag" :"ANNOTATION_TRANSCRIPTOMICS",
          "title" :"Transcriptomics"
        }
    ]
  }
]
```

References

[1] Stephen Altschul, Barry Demchak, Richard Durbin, Robert Gentleman, Martin Krzywinski, Heng Li, Anton Nekrutenko, James Robinson, Wayne Rasband, James Taylor, and Cole Trapnell. The anatomy of successful computational biology software. *Nature Biotechnology*, 31:894–897, 2013.

[2] Michael Bächle and Paul Kirchberg. Ruby on rails. *IEEE Software*, 24:105–108, 2007.

[3] Thomas Bayer. REST Web Services. *... 2002) http://www. oio. de/public/xml/rest-webservices. ...*, pages 3–5, 2007.

[4] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15, 2008.

[5] Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: A web-based genome analysis tool for experimentalists, 2010.

[6] Brandi L. Cantarel, Ian Korf, Sofia M C Robb, Genis Parra, Eric Ross, Barry Moore, Carson Holt, Alejandro Sánchez Alvarado, and Mark Yandell. MAKER: An easy-to-use annotation pipeline designed for emerging model organism genomes. *Genome Research*, 18(1):188–196, 2008.

[7] Tim Carver and Alan Bleasby. The design of Jemboss: A graphical user interface to EMBOSS. *Bioinformatics*, 19:1837–1843, 2003.

[8] B Chevreux, T Pfisterer, B Drescher, A J Driesel, W E G Müller, T Wetter, and S Suhai. Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs. *Genome Research*, 14:1147–1159, 2004.

[9] J Falgueras, A J Lara, N Fernández-Pozo, F R Cantón, G Pérez-Trabado, and M G Claros. SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics*, 11:38, 2010.

[10] Juan Falgueras, Antonio J. Lara, Francisco R. Cantón, Guillermo Pérez-Trabado, and M. Gonzalo Claros. SeqTrim - A validation and trimming tool for all purpose sequence reads. In *Advances in Soft Computing*, volume 44, pages 353–360, 2007.

[11] M.Gonzalo Fernández-Pozo, Noé and Guerrero-Fernández, Darío and Bautista, Rocío and Gómez-Maldonado, Josefa and Avila, Concepción and Cánovas, FranciscoM. and Claros. GENote v.β: A Web Tool Prototype for Annotation of Unfinished Sequences in Non-model Eukaryotes. In *Bioinformatics for Personalized Medicine*, pages 66–71. 2012.

[12] David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language*, volume 159. 2008.

[13] Don Gilbert. Pise: software for building bioinformatics webs. *Briefings in bioinformatics*, 3:405–409, 2002.

[14] Darío Guerrero, Rocío Bautista, David P Villalobos, Francisco R Cantón, and M Gonzalo Claros. AlignMiner: a Web-based tool for detection of divergent regions in multiple sequence alignments of conserved sequences. *Algorithms for molecular biology : AMB*, 5:24, 2010.

[15] T Hoffmann and N Laird. fgui: A Method for Automatically Creating Graphical User Interfaces for Command-Line R .... *Journal of Statistical Software*, 2009.

[16] X Huang and A Madan. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9:868–877, 1999.

[17] JSON.org. Introducing JSON, 2014.

[18] Sudhir Kumar and Joel Dudley. Bioinformatics software for biologists in the genomics era, 2007.

[19] A Lara, G Pérez-Trabado, D Villalobos, S Díaz-Moreno, F Cantón, and M G Claros. A Web Tool to Discover Full-Length Sequences: Full-Lengther. In E Corchado, J M Corchado, and A Abraham, editors, *Innovations in Hybrid Intelligent Systems*, pages 361–368. Springer, 2007.

[20] Burkhard Linke, Robert Giegerich, and Alexander Goesmann. Conveyor: A workflow engine for bioinformatic analyses. *Bioinformatics*, 27(7):903–911, 2011.

[21] J R Miller, A L Delcher, S Koren, E Venter, B P Walenz, A Brownley, J Johnson, K Li, C Mobarry, and G Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.

[22] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *CHI*, pages 195–202, 1992.

[23] Kumar Potti Pavan, Ahuja Sanjay, and Prodanoff Zornitza. Comparing Performance of Web Service Interaction Styles : SOAP vs . REST. In *2012 Proceedings of the Conference on Information Systems Applied Research*, pages 1–24, 2012.

[24] Paolo Romano, Ezio Bartocci, Guglielmo Bertolini, Flavio De Paoli, Domenico Marra, Giancarlo Mauri, Emanuela Merelli, and Luciano Milanesi. Biowep: a workflow enactment portal for bioinformatics applications. *BMC bioinformatics*, 8 Suppl 1:S19, 2007.

[25] Torsten Seemann. Ten recommendations for creating usable bioinformatics command line software. *GigaScience*, 2:15, 2013.

[26] Pedro Seoane, Rosario Carmona, Rocío Bautista, and Others. AutoFlow: an easy way to build workflows.

[27] Ilhami Visne, Erkan Dilaveroglu, Klemens Vierlinger, Martin Lauss, Ahmet Yildiz, Andreas Weinhaeusel, Christa Noehammer, Friedrich Leisch, and Albert Kriegner. RGG: a general GUI Framework for R scripts. *BMC bioinformatics*, 10:74, 2009.

[28] Erik Wilde and Robert J. Glushko. XML Fever, 2008.

[29] J. Sergio Zepeda and Sergio V. Chapa. From desktop applications towards ajax web applications. In *2007 4th International Conference on Electrical and Electronics Engineering, ICEEE 2007*, pages 193–196, 2007.

Parte III

APROVECHAMIENTO DE LOS RECURSOS DE SUPERCOMPUTACIÓN

Capítulo 7

Entorno para paralelizar y distribuir tareas con secuencias

Mi contribución: El desarrollo completo de este entorno de trabajo para paralelizar herramientas bioinformáticas siguiendo una estrategia de granja de tareas. Aunque se ha utilizado en varios programas desarrollados por el grupo de investigación, muestro la utilidad de la herramienta mediante la paralelización de BLAST+.

Título del artículo: SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+
Autores: Darío Guerrero-Fernández, Juan Falgueras, and M. Gonzalo Claros
Publicación: Computational Biology Journal
Volumen:2013
Año: 2013
DOI: http://dx.doi.org/10.1155/2013/707540

Abstract Current genomic analyses often require the managing and comparison of big data using desktop bioinformatic software that was not developed regarding multicore distribution. The task-farm SCBI_MAPREDUCE is intended to simplify the trivial parallelisation and distribution of new and legacy software and scripts for biologists who are interested in using computers but are not skilled programmers. In the case of legacy applications, there is no need of modification or rewriting the source code. It can be used from multicore workstations to heterogeneous grids. Tests have demonstrated that speed-up scales almost linearly and that distribution in small chunks increases it. It is also shown that SCBI_MAPREDUCE takes advantage of shared storage when necessary, is fault-tolerant, allows for resuming aborted jobs, does not need special hardware or virtual machine support, and provides the same results than a parallelised, legacy software. The same is true for interrupted and relaunched jobs. As proof-of-concept, distribution of a compiled version of BLAST+ in the SCBI_DISTRIBUTED_BLAST gem is given, indicating that other blast binaries can be used while maintaining the same SCBI_DISTRIBUTED_BLAST code. Therefore, SCBI_MAPREDUCE suits most parallelisation and distribution needs in, for example, gene and genome studies.

Capítulo 8

Formato comprimido para almacenar y manejar las lecturas de los ultrasecuenciadores en un supercomputador

Mi contribución: En este trabajo desarrollé inicialmente la librería de compresión en Ruby a modo de prototipo para evaluar la viabilidad del proyecto. Posteriormente entre Rafael Larrosa y yo diseñamos el formato comprimido final y realizamos la implementación en C. Una vez terminada la libería en C, realicé las mediciones de rendimiento con los distintos tipos de secuencias y una gema `scbi_fqbin` para facilitar su uso en Ruby.

Título del artículo: FQbin a compatible and optimized format for storing and managing sequence data
Autores: Guerrero-Fernández, D.; Larrosa, R. and Claros, M. G.
Publicación: IWBBIO 2013 - International Work-Conference on Bioinformatics and Biomedical Engineering Proceedings
Volumen:2013 pp. 337-344
Año: 2013
DOI/ISBN: ISBN978-84-15814-13-9

Abstract Existing hardware environments may be stressed when storing and processing the enormous amount of data generated by next-generation sequencing technology. Here, we propose FQbin, a novel and versatile tool in C for compressing, storing and reading such sequencing data in a new and Fasta/FastQ-compatible format that outperforms the existing proposals. It is based on the general-purpose zLib library and offers up to 10X compression. The compressed file is read and decompressed up to 3X faster than a FastQ file is read, and a nearly ínstant'random access to every entry in the FQbin container is allowed. Fast file reading is maintained even in shared storage environments, where dif-ferent processes are simultaneously accessing the same FQbin file. Slow networks can take even more advantage from FQbin.

Parte IV

DESARROLLO DE HERRAMIENTAS BIOINFORMÁTICAS

Capítulo 9

Herramienta web para detectar las regiones más divergentes en un alineamiento de secuencias

Mi contribución: He desarrollado el algoritmo de AlignMiner así como la interfaz web interactiva en la que se incluye tanto la detección de las regiones divergentes, como una herramienta para diseñar los cebadores óptimos que distingan cada secuencia.

Título del artículo: AlignMiner: a Web-based tool for detection of divergent regions in multiple sequence alignments of conserved sequences.
Autores: Guerrero, D., Bautista, R., Villalobos, D. P., Cantón, F. R., and Claros, M. G.
Publicación: Algorithms for Molecular Biology.
Volumen:5
Año: 2010
DOI: http://dx.doi.org/10.1186/1748-7188-5-24

Abstract

Background Multiple sequence alignments are used to study gene or protein function, phylogenetic relations, genome evolution hypotheses and even gene polymorphisms. Virtually without exception, all available tools focus on conserved segments or residues. Small divergent regions, however, are biologically important for specific quantitative polymerase chain reaction, genotyping, molecular markers and preparation of specific antibodies, and yet have received little attention. As a consequence, they must be selected empirically by the researcher. AlignMiner has been developed to fill this gap in bioinformatic analyses.

Results AlignMiner is a Web-based application for detection of conserved and divergent regions in alignments of conserved sequences, focusing particularly on divergence. It accepts alignments (protein or nucleic acid) obtained using any of a variety of algorithms, which does not appear to have a significant impact on the final results. AlignMiner uses different scoring methods for assessing conserved/divergent regions, Entropy being the method that provides the highest number of regions with the greatest length, and Weighted being the most restrictive. Conserved/divergent regions can be generated either with respect to the consensus sequence or to one master sequence. The resulting data are presented in a graphical interface developed in AJAX, which provides remarkable user interaction capabilities. Users do not need to wait until execution is complete and can.even inspect their results on a different computer. Data can be downloaded onto a user disk, in standard formats. In silico and experimental proof-of-concept cases have shown that AlignMiner can be successfully used to designing specific polymerase chain reaction primers as well as potential epitopes for antibodies. Primer design is assisted by a module that deploys several oligonucleotide parameters for designing primers .ºn the fly".

Conclusions AlignMiner can be used to reliably detect divergent regions via several scoring methods that provide different levels of selectivity. Its predictions have been verified by experimental means. Hence, it is expected that its usage will save researchers'time and ensure an objective selection of the best-possible divergent region when closely related sequences are analysed. AlignMiner is freely available at http://www.scbi.uma.es/alignminer webcite.

Capítulo 10

Herramienta modular para preprocesar las lecturas obtenidas por cualquier tipo de ultrasecuenciador

Mi contribución: Me he encargado de diseñar cómo ha de funcionar la arquitectura de *plugins* para añadirlos o cambiarlos sin problema cada vez que aparezca un nuevo ultrasecuenciador. También me he encargado de implementar el sistema de paralelización basado en SCBI_MapReduce (capítulo 7), además de otras gemas necesarias para compatibilizar las entradas y salidas del programa y que se pueden ver en el repositorio de gemas de Ruby `https://rubygems.org/search?query=scbi`. El artículo está en el formato de la revista Bioinformatics porque se envió a ella, pero fue rechazado y estamos mejorándolo para volverlo a enviar a otra revista.

BIOINFORMATICS

Vol. 00 no. 00 2011
Pages 1–3

SeqTrimNext: pre-processing sequence reads for next-generation sequencing projects

Darío Guerrero-Fernández [1], Almudena Bocinos [1,2], Rocío Bautista [1], Noé Fernández-Pozo [3], Juan Falgueras [2] and M. Gonzalo Claros [1,3*]

[1]Plataforma Andaluza de Bioinformática (Edificio de Bioinnovación), [2]Departamento de Leguajes y Ciencias de la Computación, and [3]Departamento de Biología Molecular y Bioquímica, Universidad de Málaga, 29071 Málaga, Spain.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXX

ABSTRACT

Summary: SeqTrimNext is a quick, reliable, distributed and/or parallelised, customisable, pipeline for pre-processing next-generation sequencing reads. It is also available as a friendly web tool and as a REST web service. Minimal user intervention is required for obtaining pre-processed reads, which are ready for any downstream analysis. A detailed summary statistics for quality-control of input and output data is given. SeqTrimNext usage increases the accuracy of consensus sequences and accelerates further assembly, saving curation time for sequencing projects.

Availability: SeqTrimNext is distributed under the GNU Affero Public License (AGPL) as a Ruby gem (`gem install seqtrimnext`) and is freely available to both academic and commercial users. Web site: `http://www.scbi.uma.es/seqtrimnext`.

Contact: dariogf@uma.es

Supplementary information: Supplementary figure is available at the journal's web site

INTRODUCTION

The advent of the technologies so-called next-generation sequencing (NGS) is giving non-specialist laboratories the ability to obtain large amounts of sequences. Despite new pre-processing needs, this has not been accompanied by a quality control of the sequence accurateness (Schmieder and Edwards, 2011). In addition to the typical polyA/T, adaptors, contamination or low quality portions that are a common source of sequencing errors, pre-processing should now take into account new NGS specific features such as (i) processing time and computing capability for a very large number of reads; (ii) presence of artificial read duplicates (e.g. clonality; Huang *et al.*, 2010); (iii) location and removal of sample identifiers (tags, barcodes and key; Schmieder *et al.*, 2010); (iv) grouping reads according to barcodes; (v) managing of paired-end reads; (vi) discarding low complexity artefacts introduced by the sequencing technology; and (vii) coping with standard formats (Cock *et al.*, 2010). Widely-used pre-processors such as SeqClean or Lucy2 do not face most of these particular challenges, since they were developed for Sanger sequences. Others such as SeqTrim (Falgueras *et al.*, 2010), PyroCleaner, or the trimming steps of ngs_backbone (Blanca *et al.*, 2011) and Newbler (Roche), face only some of them. Other recent pipelines were designed only for specific problems [e.g. CANGS (Pandey *et al.*, 2010), TagCleaner (Schmieder *et al.*, 2010), CLOTU (Kumar *et al.*, 2011), PRINSEQ (Schmieder and Edwards, 2011)]. Finally, some authors use in-house scripts for pre-processing (e.g. Diguistini *et al.*, 2009). As a result, even if a lot of time is invested on curation by hand, databases are contaminated with NGS linkers and adaptors (mainly HTGS database, see e.g. AC225459.16, AC122759.1, AC239084.1). Therefore, there is a need for new, fast, efficient, reliable, easy-to-install, user-friendly, pre-processing software for NGS reads for a wide range of computers and experimental conditions (e.g. *de novo* assembling, mapping, amplicons). SeqTrimNext has been developed to fill these necessities and to cope with all NGS peculiarities, providing output files that can be used as input for downstream analyses.

DESCRIPTION

SeqTrimNext is a parallelised evolution of SeqTrim (Falgueras *et al.*, 2010), completely re-written in Ruby 1.9.2, as a command line tool. The web interface and the REST-based web service have been built using InGeBiol —a Ruby-on-Rails 2.0 framework giving a web interface and web service commands for any command-line (Guerrero-Fernández and Claros, in preparation)—. The modular architecture of SeqTrimNext, based on a pipeline of orthogonal plugins, is especially suitable for addition, removal, or reordering of plugins and easy adaptation for future evolution. Internal configuration files are in JSON for efficiency in the storage and parsing phases. Similarity searches were carried out with BLAST+ customised calls. Other required programs are CD-HIT (Huang *et al.*, 2010), GNU_PLOT (`http://www.gnuplot.info`), and optionally latex and the gem *seqtrimnext_report* (on trial) for constructing a summary PDF. Distributed and parallel execution of SeqTrimNext based on one manager and a variable number of independent workers is provided by the Ruby gem *SCBI_MapReduce* (Guerrero-Fernández et al, in preparation). The number of workers can be optimally made equal to the number of cores in each machine, minimising any idle time. Additional acceleration has been achieved by (i) processing chunks of 100

*to whom correspondence should be addressed

sequences instead of one by one, (ii) allocating memory on a per-sequence basis, and (iii) writing only the final results on disk (avoiding temporary files).

Common sequence formats such as FASTQ (the recommended by default due to size efficiency), FASTA + QUAL, SFF (Standard Flowgram Format), and a ZIP containing any of the previous formats are accepted. Input datasets can only contain single-end reads, only paired-end reads, or a mix or both. Input sequences shorter than 40 bp (parametrisable) are filtered during the initial statistical processing; removal of sequences longer than the mode plus two standard deviations is optional. Outputs consist of downloadable files containing (i) trimmed sequences (grouped by barcodes if necessary) in standard FASTQ that can be easily forwarded to downstream analyses, (ii) list of rejected sequences, and (iii) detailed statistics on the input and output datasets (quality plots, cumulative number of reads passing the different trimming steps, and optionally a user readable PDF file intended for easy inspection and interpretation of results). When appropriate, an input file for the *sffile* tool (Roche) is provided in order to reconstruct the output dataset into SFF. Other noteworthy features of SeqTrimNext are: (i) strict analysis of the QV of nucleotides. (ii) Recovery of more true paired-ends than Newbler, providing less mis-classified single-end reads that were really imperfect paired-ends (Table 1). (iii) Exhaustive removal of linkers, adaptors and the like, even if they happen to be repeated (in tandem or interspersed). (iv) Removal of sequences that are more likely to match at a random position by chance (i.e. short, low-complexity and contaminant sequences) to avoid for example mis-mappings and mis-assemblies. Even though personalised vectors, adaptors or contaminant sequences can be uploaded, (v) the software is released with a customised database for contaminants containing representative prokaryotic and fungal sequences (including complete mitochondria and chloroplast genomes), the UniVec database for vector removal, and collections of linkers and adaptors, which are appropriate for most uses. Additionally, as different users and datasets have different demands, (vi) customised configuration files for the most common analyses (i.e. specific pipelines for 454 single-end reads, mixed 454 single- and paired-end reads, amplicons, Illumina reads, etc.) are part of the software; these files can also be edited and created by skilled users.

Pre-processed reads are finally rejected if complying with one of the following: (1) clean insert is too short (parametrisable, 40 nt by default); (2) a segment of low complexity of 30 nt or more occurs after a poly-T because the sequence is an artefact introduced by sequencer; and (3) a significant piece of vector (parametrisable, 50 nt by default) occurs within the sequence, not at the ends. By default, (4) sequences with a significant fragment of contamination (parametrisable, longer than 40 nt with 85% identity by default) are also removed, but it can be disabled. (5) Sequences with interspersed Ns are rejected only when cleaning amplicons.

RESULTS AND DISCUSSION

A first instance of SeqTrimNext computing efficiency was assessed using the paired-end (PE) reads of Table 1, since they are the most complex and artefact-prone type of reads. It takes 21.6 min to completely pre-process 146 189 Titanium reads using 8 cores at 3 GHz with 1 Gb of RAM per core. Using 32 cores, the time was reduced to 8.2 min (the decrease is not strictly proportional since there is a linear processing at the beginning). Another computing efficiency instance was assessed using 5 173 706 Illumina reads, which were pre-processed in 17 min and 13.6 min using 8 and 32 cores, respectively (these reads are simpler to pre-process but waste most of the computation time on reading/writing reads on disk). Therefore, pre-processing of NGS reads using SeqTrimNext could be handled by most laboratories without bioinformatic skills.

Table 1. Assembly of different 454-reads with Newbler 2.3 with (+STN) and without (–STN) SeqTrimNext pre-processing.

	–STN:S[a]	+STN:S[a]	–STN:PE[b]	+STN:PE[b]
Useful reads	100 000	36 319	[c]77 994	[d]50 584
Repeated targets	48	71	13 808	69
Multiply mapped	-	-	12 601	44
N50	1 477	925	2 325	17 835
Contig count	1 756	1 998	35	4
PE distance av.	-	-	2 543	2 880
E. coli contamination (bp)	209	0	102	0
Linkers/adaptors/vector (bp)	26	0	362	0
Newbler execution time[e]	82 s	32 s	20.6 min	6.6 min

[a] 100 000 raw single-end (S) reads from SRA:SRR069473 for 15 mellon BACs
[b] 146 189 raw paired-end (PE) reads reconstructing a single *P. pinaster* BAC clone
[c] Newbler mistakenly qualifies the remaining 68 195 reads as single-end reads, while a significant number of them are still containing linkers (SeqTrimNext only qualifies 37 099 reads as true single-end reads, the remaining having a linker)
[d] 109 090 reads were considered paired-end, but only 50 584 contain useful sequences at both sides, the remaining being classified as single-end reads devoid of any linker sequence at all
[e] Newbler 2.3 was executed using 2 cores at 2.4 GHz with 4 Gb of RAM

Since NGS technologies are not perfect, sequence quality of reads is essential for ensuring that any downstream analysis would not be compromised. All-single-end reads and all-paired-end reads (S and PE in Table 1, respectively) were assembled using Newbler 2.3. The results suggested that (1) despite diminishing the number of useful reads with SeqTrimNext, assemblies were from slightly lower (+STN:S) to clearly better (+STN:PE). (2) The numbers of repeated targets and multiply mapped reads dramatically diminished when paired-end reads were assembled (compare –STN:PE and +STN:PE, Table 1), which accounts for the significantly reduced execution time for Newbler (bottom line, Table 1). In fact, (3) absence of clonal reads, contaminant and linker/adaptor sequences, and mis-classified paired-ends, results in computational benefits by reducing the number of artefactual sequences that need to be processed in the alignment, and by lowering the memory requirements, which accordingly accelerates and facilitates the sequence assembly (as well as mapping, not shown). (4) N50 is significantly increased for paired-ends, together with a diminished count of contigs; the fact that single-end reads behave differently could be interpreted as a failure of SeqTrimNext but, on the contrary, it is an achievement since it has permitted the reduction of *E. coli*, vector, adaptor and linkers in the consensus sequences (0 bp using SeqTrimNext [Table 1], even if filtering with *E. coli* DH10B and the corresponding BAC vector was enabled in Newbler). (5) Mining the genomic sequence for putative genes using GENote (Fernández-Pozo et al., in press) revealed that consensus sequences in –STN cases contain artefactual repetitions of sequence fragments, while these repetitions were diminished or absent in the +STN cases.

Finally (6), pre-processed paired-end sequences reveal a mean fragment length closer to the expected one (3 kb).

Since the same can be concluded using other NGS datasets (Supplementary Figure 1), assemblies after a SeqTrimNext pre-processing seem to be more reliable and accurate, specially in the case of paired-end reads, with a minimal user intervention, making it suitable even for laboratories without bioinformatic staff. Such a pre-processing should avoid the current GenBank database contamination with 454 linkers and adaptors (e.g. FN668944.1, FQ032835.1, HQ671669.1). Moreover, pre-processing using SeqTrimNext will also reduce the subsequent time of manual curation of assemblies and mappings because the obtained results lack misconnections due to artefactual sequences.

ACKNOWLEDGEMENTS

The authors wish to acknowledge C. Avila and F.M. Cánovas for allowing us to share their 454 reads of *P. pinaster* BAC clones, and B. Michel and E. Viguera for sharing with us all their Illumina reads. The authors also thankfully acknowledge the computer resources provided by the Plataforma Andaluza de Bioinformática of the University of Málaga

Funding: Grants from the Spanish MICINN [BIO2009-07490], the European Union [PLE2009-0016], and the Junta de Andalucía [CVI-6075], as well as funding to the research group BIO-114.

Conflict of Interest: None declared.

REFERENCES

Blanca, J. M., Pascual, L., Ziarsolo, P., Nuez, F., and Cañizares, J. (2011). ngs-backbone: a pipeline for read cleaning, mapping and snp calling using next generation sequence. *BMC Genomics*, **12**, 285.

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., and Madden, T. L. (2009). Blast+: architecture and applications. *BMC Bioinformatics*, **10**, 421.

Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2010). The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic Acids Res*, **38**(6), 1767–71.

Diguistini, S., Liao, N. Y., Platt, D., Robertson, G., Seidel, M., Chan, S. K., Docking, T. R., Birol, I., Holt, R. A., Hirst, M., Mardis, E., Marra, M. A., Hamelin, R. C., Bohlmann, J., Breuil, C., and Jones, S. J. (2009). De novo genome sequence assembly of a filamentous fungus using sanger, 454 and illumina sequence data. *Genome Biol*, **10**(9), R94.

Falgueras, J., Lara, A. J., Fernández-Pozo, N., Cantón, F. R., Pérez-Trabado, G., and Claros, M. G. (2010). Seqtrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics*, **11**, 38.

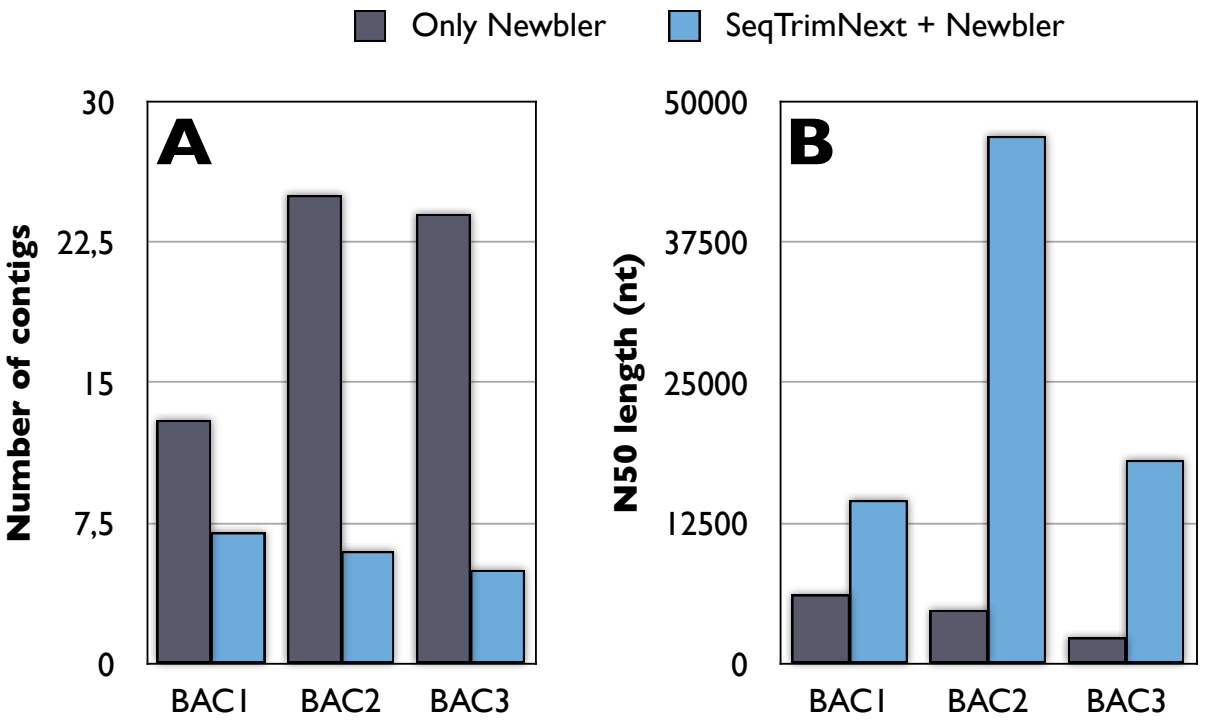
Huang, Y., Niu, B., Gao, Y., Fu, L., and Li, W. (2010). Cd-hit suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, **26**(5), 680–2.

Kumar, S., Carlsen, T., Mevik, B.-H., Enger, P., Blaalid, R., Shalchian-Tabrizi, K., and Kauserud, H. (2011). Clotu: An online pipeline for processing and clustering of 454 amplicon reads into otus followed by taxonomic annotation. *BMC Bioinformatics*, **12**, 182.

Pandey, R. V., Nolte, V., and Schlötterer, C. (2010). Cangs: a user-friendly utility for processing and analyzing 454 gs-flx data in biodiversity studies. *BMC Res Notes*, **3**, 3.

Schmieder, R. and Edwards, R. (2011). Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, **27**(6), 863–4.

Schmieder, R., Lim, Y. W., Rohwer, F., and Edwards, R. (2010). Tagcleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC Bioinformatics*, **11**, 341.



Suppl. Figure 1: Assembly of 3 different *Pinus pinaster* BAC clones using 454 paired-end reads and Newbler 2.3 with and without pre-processing using SeqTrimNext. A) Number of contigs obtained after assembling; clearly, SeqTrimNext pre-processed reads provide fewer contigs than the original reads pre-processed by Newbler. B) N50 value for the same BAC clones, reflecting that SeqTrimNext pre-processed reads can be assembled into longer contigs. Scaffolds of the three BAC clones after SeqTrimNext pre-processing are devoid of linkers, adaptors and *E coli* sequences.

Capítulo 11

Herramienta para analizar transcriptomas de especies no modelo que se han ensamblado *de novo*

Mi contribución: He contribuido a paralelizar y distribuir la ejecución del algoritmo en todas las etapas posibles y a resolver los problemas computacionales relacionados con el manejo de las bases de datos subyacentes. También he desarrollado gemas de Ruby que han servido para compatibilizar las entradas y salidas del programa, que se pueden consultar en `https://rubygems.org/search?query=scbi`. El artículo está en el formato de la revista Bioinformatics porque se envió a ella, pero fue rechazado y lo estamos revisando para volverlo a enviar a otra revista.

BIOINFORMATICS

Vol. 00 no. 00 2012
Pages 1–17

FULL-LENGTHERNEXT: A tool for fine-tuning *de novo* assembled transcriptomes of non-model organisms

Noé Fernández-Pozo [1], Darío Guerrero-Fernández [2], Rocío Bautista [2] and M. Gonzalo Claros [1,2*]

[1]Departamento de Biología Molecular y Bioquímica, Facultad de Ciencias, Universidad de Málaga, 29071, Spain.
[2]Plataforma Andaluza de Bioinformática, Centro de Supercomputación y Bioinformática, Edificio de Bioinnovación, Universidad de Málaga, 29590 Málaga, Spain.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: *De novo* transcriptome assemblies devoid of any genomics or transcriptomics reference occur commonly when working with non-model species. There is no easy way to distinguish between artefacts and species-specific transcripts, or to identify the best possible assembly before deep annotation or complementary laboratory research.

Results: FULL-LENGTHERNEXT has been designed as a parallelisable and distributable command-line, web-tool and REST web-service pipeline adapted to high-throughput transcriptomics. It has applications in (1) the classification of unigenes among coding for complete or incomplete proteins, (2) the construction of an open reading frame fixing the likely frame-shifts in unigene assemblies; (3) the discovery of putative species-specific unigenes; and (4) the extraction of putative non-coding RNAs. Combining the previous information, it provides a quick overview for the selection of the best *de novo* transcriptome assembly. Particularly, it has been shown that independent *de novo* assemblies using MIRA3 and Euler-SR can be reconciled using CAP3 in a more successful transcriptome. Also, unigenes receiving an `Unknown` status usually are short sequences that can be discarded for subsequent processings.

Availability: FULL-LENGTHERNEXT can be freely downloaded or executed via web at http://www.scbi.uma.es/full_lengther2

Contact: claros@uma.com

Supplementary Information: Suppl. Files 1, 2, 3 and 4.

1 INTRODUCTION

Next-generation sequencing (NGS) platforms can sequence a particular transcriptome in a fast and cost-effective way (see for example Angeloni *et al.*, 2011; Malik *et al.*, 2011; Vera *et al.*, 2008; Kristiansson *et al.*, 2009). Assemblers for transcriptomics reads (e.g. Chevreux *et al.*, 2004; Pavel A. Pevzner and Waterman, 2001; Li *et al.*, 2009) will produce contigs (usually considered unigenes) that, in the ideal case, should be close to the number of real genes expressed under the studied conditions. When analysing *de novo* transcriptomes, it is quite difficult to distinguish between artefacts, true new transcripts, and the estimated number of identified transcripts. Annotations cannot be relied on for distinguishing between them since they are based on sequence similarity, and most of the problematic cases will not have any orthologue in databases. In fact, in transcriptomics projects, 40-60% of unigenes do not match any similar sequences in databases (Paterson *et al.*, 2010; Fernández-Pozo *et al.*, 2011; Parchman *et al.*, 2010). Most of the times, researchers hypothesise that they derive from lineage- or species-specific genes, but unfortunately a lot of them might simply be the result of misassembly or assembly of inaccurate reads (Paterson *et al.*, 2010). For that reason, efforts should be invested in distinguishing real new genes from misassembled sequences.

In *de novo* transcriptome assemblies, the identification of reconstructed transcripts coding for a complete protein has not received enough attention, even though they are extremely useful for (i) inferring the protein coded by the transcript, (ii) determining the genomic structure of genes in non-model organisms, (iii) facilitating gene identification efforts, and (iv) catalysing experimental research (Team, 2002). Identification of full-length cDNAs is a milestone in non-model species (Ralph *et al.*, 2008; 1KP Project `http://www.onekp.com`). No bioinformatics tool is available for identification of full-length transcripts in NGS projects. Tools such as TargetIdentifier (Min *et al.*, 2005) or Full-Lengther (Lara *et al.*, 2007), did exist for full-length transcript prediction in classical EST (expressed sequence tag) experiments. These softwares were prepared for low-throughput, cDNA-cloning based ESTs. In spite of their widespread use (Wang *et al.*, 2010; Koop *et al.*, 2008; Kim *et al.*, 2008; Semova *et al.*, 2006), both algorithms are ineffectual for working with unigenes from NGS for two reasons: (1) there is no clone supporting any sequence (which is the main rationale of these algorithms), and (2) they are not prepared for high-throughput, and cannot cope with new sequencing technologies.

Another aspect of *de novo* transcriptome assembly is the constant development and improvement of NGS assemblers. But, there is no easy way to determine which is the best possible assembly when working with a non-model species. It is proposed here that this task could be assisted by determining the amount of long reconstructed transcripts and the number of contigs coding for different, complete proteins.

*to whom correspondence should be addressed

We present FULL-LENGTHERNEXT, a tool designed to meet the need for *de novo* transcriptome assembly of non-model organisms described above. FULL-LENGTHERNEXT is adapted to NGS technologies and can work in parallel and in a distributed way to optimise computing resources. It can classify unigenes depending on their coding content, extract the putative protein from unigene sequences, suggest which unannotated unigenes could be species-specific coding sequences and is able to help in deciding which *de novo* assembly is the best for a non-model organism. Data supporting these capabilities are given.

2 APPROACH

The overall algorithm underlying FULL-LENGTHERNEXT is depicted in Fig. 1. It divides the input file in chunks of `chunkSize` that are distributed in as many `workers` (cores) as were selected (see Suppl. File 1 for details about parameter meaning). When a worker finishes the analysis of a chunk, it receives a new chunk, and so on until all chunks have been analysed (Fig. 1A). Within each worker, sequences are compared successively against modules shown in Fig. 1A until a significant homology is found. Finally data and annotations (including the assigned status) are saved on disk.

The presence of frame-shifts in unigenes is revealed by the partition of a reference sequence in several hits. In order to obtain a contiguous protein sequence for the unigene, FULL-LENGTHERNEXT selects the correct frames (BLASTX hits) and joins them separated by the necessary Xs (representing unknown amino acids) to fill the gap between consecutive hits or to cover the overlapping residues in consecutive hits ('Fix Frame Errors' in Fig. 1B).

Unigenes showing similarity in both strands with the same reference do not allow for a ORF reconstruction. They are tagged as misassembled (Fig. 1B) and are consequently discarded for further processing. Start and stop codons for the reconstructed ORF are then located in the unigene sequence based on the reference sequence. Depending on the cases found, FULL-LENGTHERNEXT will assign one of the following statuses (see "Suppl. File 2" for more detailed description): `N-terminal` for unigenes lacking the 3'-end coding region, `C-terminal` for unigenes lacking the 5'-end coding region, `Complete` for coding for a complete protein, `Internal` for unigenes lacking the 5'-end and 3'-end coding region; and `Misassembled` for unigenes with a clear missassembly.

Unigenes classified as `Complete` are annotated with information from the same reference that enabled its analysis. Sequences that have passed through the three databases without retrieving any similarity could be putative species-specific genes or misassemblies (Paterson *et al.*, 2010). TestCode algorithm (Fickett, 1982) implemented as a Ruby class was used for obtaining the probability that a sequence is coding. Since TestCode only renders reliable results for sequences longer than 200 bp, it was used for the analysis of unigenes whose predicted open reading frame was longer than 200 bp after the analysis of both strands (Fig. 1C). Unigenes can then be given the status of `Coding` if the TestCode score > 0.74 (see "Suppl. File 2" for more detailed description). Unigenes with scores lower than 0.74 are candidates for non-coding sequences or misassemblies (including assemblies of artefactual reads). They, together with unigenes containing ORFs shorter than 200 bp, are then compared with the ncRNA database using BLASTN with an *E*-value of 10^{-3}, to discern true non-coding RNAs from other putatively useless sequences. This analysis can provide the status of `Putative ncRNA`. Finally, unigenes without any similarity in any database and with a TestCode < 0.74 receive the status `Unknown`.

3 METHODS

3.1 Architecture

FULL-LENGTHERNEXT has been developed in Ruby on OSX and SLES Linux. It can be used as a command line, as a Web tool (`http://www.scbi.uma.es/full_lengther2`), or as a REST-based Web service. It can be installed as a Ruby gem using the command `gem install full_lengther_next`, or can also be downloaded from `http://www.scbi.uma.es/downloads`. Installation as a gem is preferred since other gem dependencies (such as SCBI_MapReduce, scbi_blast and scbi_fasta) are automatically installed. Execution requires previous installation of Ruby 1.9.2 or higher, and BLAST+ (Camacho *et al.*, 2009). FULL-LENGTHERNEXT has been designed to use multiple parallel or distributed cores by using the Ruby gem SCBI_MapReduce (Guerrero-Fernandez et al, submitted), based on one manager and a variable number of independent workers. The number of workers can be optimally made equal to the number of available cores in each machine or network. FULL-LENGTHERNEXT has been developed and tested using a cluster of 10 blades with 8 Xeon cores (x86 architecture) at 3.00 GHz and 16 GB of RAM per blade, and using PBS as queuing system. Details about launching FULL-LENGTHERNEXT and the resulting output is described in "Suppl. File 1".

3.2 Reference databases

FULL-LENGTHERNEXT provides one status to any sequence based on comparisons using curated protein databases and a heuristic algorithm. Protein databases contain only complete proteins: incomplete proteins were discarded on the basis of the feature field (incomplete proteins contain *NON_TER* or *NON_CONS* tags), description field (incomplete proteins contain *Flags: Fragment*), and the protein sequence (complete proteins must start by M).

The first protein database is optional and must be constructed by the user. Its rationale is to gather the sequences closest to the studied organism in order to obtain the highest number of unigenes with an orthologue in this small protein database and reduce the time spent on data analysis. The FULL-LENGTHERNEXT gem provides the script `make_user_db.rb` to build a user database; it only need specification of the database division (fungi, plants, invertebrates, etc.) and a specific taxonomic group. For example, the conifer-specific user database used below was constructed using the command:

```
make_user_db.rb plants Coniferales.
```

The following protein database consists of complete proteins derived from the UniProtKB/Swiss-Prot (Apweiler *et al.*, 2004) database, including the different isoforms, from the same division in `DBgroup` mentioned above. The last complete protein database is derived from the non-redundant, automatically annotated UniProtKB/TrEMBL from the same division as above. Being the biggest database, it will be queried only by the small dataset of proteins that have not found a reasonably homologous hit in any previous database. Both customised databases can be automatically generated by means of the script `download_fln_dbs.rb` available with the FULL-LENGTHERNEXT gem.

Sequences without similarity in any database and predicted as non-coding by FULL-LENGTHERNEXT are compared with a database of non-coding RNAs, constructed from the junction of Rfam (Griffiths-Jones *et al.* (2005)) and NONCODE (Bu *et al.* (2012)) databases and filtered with CDHIT (Li and Godzik (2006)) to avoid redundancy. The curated ncRNA database is downloaded when the above mentioned script `download_fln_dbs.rb` is

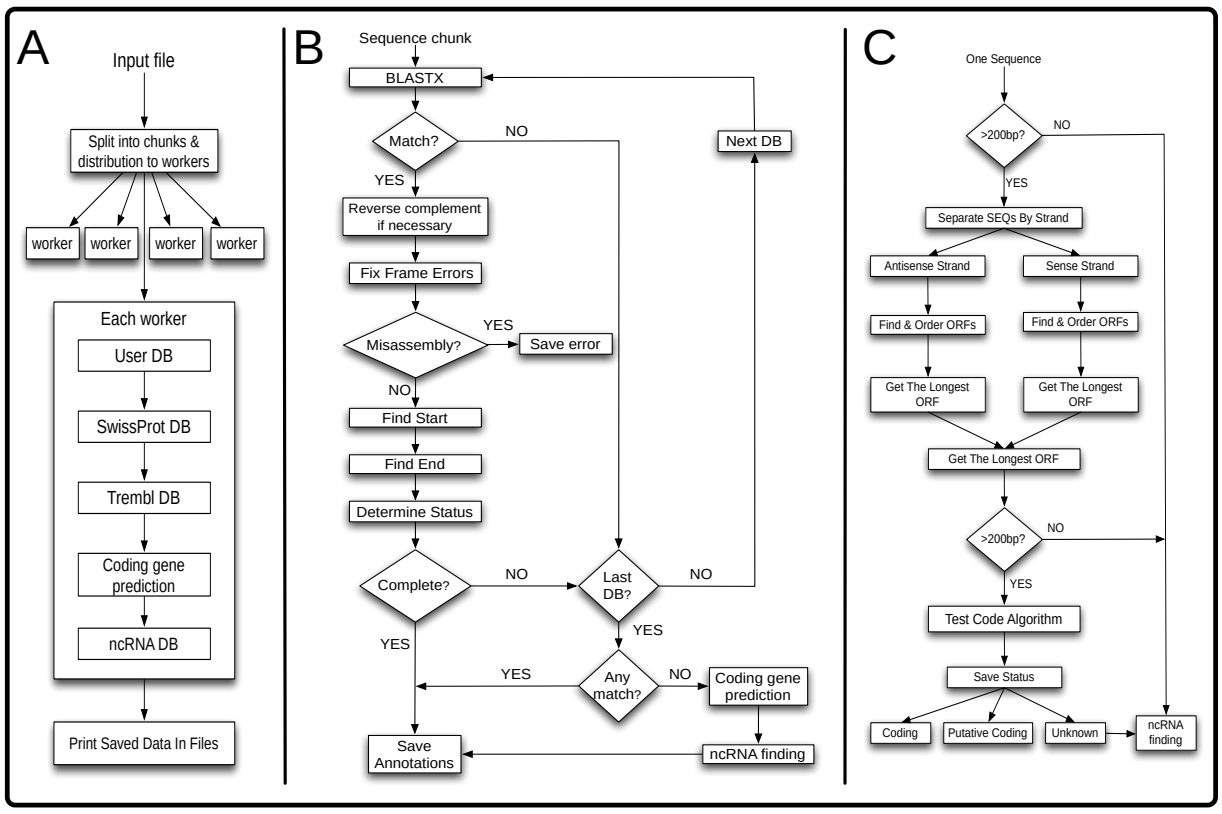


Fig. 1. Flowcharts setting out the FULL-LENGTHERNEXT algorithm. A: general scheme of the algorithm. B: details of the process involving the five modules that are executed within each worker in order to obtain one status for each sequence. C: a detailed scheme of the strategy for finding putative species-specific coding genes and non-coding RNAs; this is applied only to sequences without any similarity in protein databases.

executed, and can also be downloaded from `http://www.scbi.uma.es/downloads/FLNDB/ncrna\_fln\_100.fasta.zip`.

3.3 Sequence datasets for validation

Long-unigene dataset (LUD): this consists of a set of 10 000 sequences of 1202 bp in length (range: 6016-95 bp) on average from the putative transcriptome of *Pinus pinaster* (Fernández-Pozo *et al.*, 2011).

Short-unigene dataset (SUD): this consists of a set of 10 000 sequences of 338 bp in length on average (range: 2145-40 bp) from the putative transcriptome of *Pinus pinaster* (Fernández-Pozo *et al.*, 2011).

Complete gene dataset from GenBank (CGD-GB): this are the first 1000 non-redundant sequences from 5000 GenBank *Homo sapiens* sequences containing the description *complete cds* filtered with CD-HIT (Li and Godzik, 2006) with a similarity score of 80%. They were used as true positives (TP) for `Complete` status. The accession number of all sequences in CGD-GB can be found in the "Suppl. File 2".

Complete gene dataset from the mammalian gene collection (CGD-MGC): this are the first 1000 non-redundant sequences from 5000 *Homo sapiens* full-length sequences `ftp1.nci.nih.gov/fasta/hs_mgc_mrna.fasta.gz` filtered with CD-HIT as above. They were also used as another set of TP for `Complete` status. The accession number of all sequences in CGD-MGC can be found in the "Suppl. File 2".

Incomplete gene dataset (CG-Trimmed): all sequences in CGD-MGC dataset were modified by trimming 100 bp from each one, 50 bp from the start codon and 50 bp from the end codon. Resulting sequences are expected to be devoid of their start and stop codons and should therefore be considered an internal region of the original coding genes. This dataset served as true negatives (TN) for any status of incomplete sequence.

Complete gene dataset simulating an assembly of paired-end reads (CG_paired): assembly of paired-end reads providing transcript scaffolds containing internal blocks of Ns. To simulate such a result, 981 complete sequences qualified as `sure Complete` from CGD-MGC (Table 1) were disrupted at a random place in the second third of the sequence, with a random number of Ns (5, 10, 15, 20, 25, 30, 35, 40, 45 or 50). Ns substitute individual nucleotides so that the sequence length is preserved and a true gap is introduced. They were used as TP for `Complete` when sequences contain blocks of Ns.

Complete gene datasets with indels: the 981 sequences qualified as `sure Complete` from CGD-MGC (Table 1) were manipulated to create 16 new datasets of 981 sequences each by introducing artificial insertions, deletions, and insertions and deletions (see details in Suppl. File 3). They were used as TP for `Complete` status for tolerance to indels. Hence,

Non-coding sequence dataset (NCSD): it consists of 1000 intergenic sequences from *Arabidopsis* downloaded from an intergenic dataset from TAIR site. NCSD was used as TN for `Coding` status.

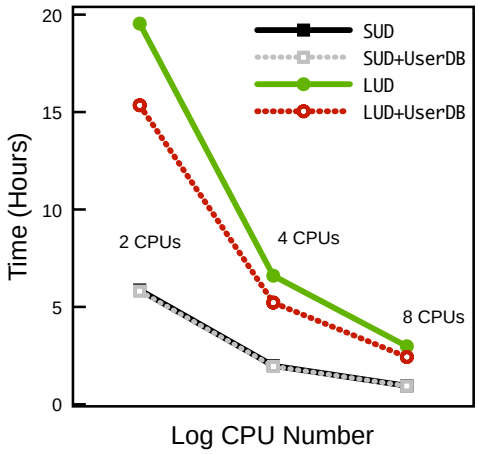


Fig. 2. Efficiency of time usage by FULL-LENGTHERNEXT illustrated as the time (in hours) spent using 2, 4 and 8 CPUs for the analysis of SUD and LUD datasets, depending on the inclusion of the optative user database.

Pyrosequencing reads from a transcriptome (NGS-T): this consists of 746 105 useful reads from a pyrosequencing reaction of the *Pinus pinaster* transcriptome described in (Fernández-Pozo *et al.*, 2011).

4 RESULTS AND DISCUSSION

4.1 Time efficiency

Computation time spent by FULL-LENGTHERNEXT algorithm was assessed with the LUD and SUD datasets of real-world sequences with 2, 4 and 8 CPUs and with and without the conifer-specific user database, UniProt databases being in any case from the plant division. It can be shown (Fig. 2) that the shorter the sequences, the faster the analysis, and that the use of the user database decreases significantly the time spent on analysing datasets of large sequences. The inclusion of a species-specific database allowed 5496 LUD sequences (55%) and 2676 SUD sequences (27%), which could account for the speed-up observed with long sequences. The use of a user database is also providing an additional advantage: it offers more accurate results due to a comparison with closer orthologues (results not shown). In our laboratory, we have built transcriptomes for different non-model species, and the analyses took from 14 h 32 min using 8 cores for 89 544 unigenes (average length: 622 ± 563 nt) to 8 h 25 min using 5 cores for 133 175 unigenes (average length: 256 ± 149 nt), taking into account that execution time will depend on the lengths of unigenes, the amount of similarity matches in databases, and the load of the queuing system. In conclusion, in only a few hours FULL-LENGTHERNEXT analysis can provide a general picture of a transcriptome assembly before the heavy investment of time that is needed for a deep annotation of a transcriptome with a dedicated software.

4.2 Reliability of `Complete` status

When GenBank sequences (CGD-GB) were analysed with FULL-LENGTHERNEXT, 27 of the 1000 sequences were not classified as `Complete` (Table 1). A detailed inspection of these sequences (see Suppl. File 4 for a specific description) revealed that 15 were correctly qualified by FULL-LENGTHERNEXT since the testing sequences were devoid of the true start or stop codon; 6 cases were mispredicted because long isoforms usually give better scores than short isoforms; and the 2 `Coding` statuses were correct, even though no reference sequence was found in the databases. Depending on the criteria, a minimum of 4 and a maximum of 12 could be considered FULL-LENGTHERNEXT failures. However, the accumulation of sequence errors in CGD-GB prompted for another full-length reference dataset source with more accurate metadata. This alternative reference was found in The Mammalian Gene Collection (CGD-MGC), where only 9 of 1000 sequences were apparently misclassified, all of them as `C-terminal` (Table 1). Again, 6 were originated by the preference of long isoforms over short isoforms, 2 were missing the 5'-end of the gene, and the last was a chimeric sequence (see Suppl. File 4 for s detailed description). Since mispredictions could range from 0 to 6 and there are less misannotations, the CGD-MGC dataset seems to be more appropriate for FULL-LENGTHERNEXT tests. This is also supported by the fact that, during the analysis, the warning messages (1) *"frame errors"* appeared 13 times in CGD-MGC and 48 times in CGD-GB; (2) *"unexpected stop codons"* appeared 3 times in CGD-MGC and 18 times in CGD-GB; and (3) *"no M at the beginning"* appeared 7 times in CGD-MGC and 18 times in CGD-GB.

Incomplete sequence status was tested using a collection of true incomplete proteins (CG_Trimmed dataset). The analysis revealed that 983 were distributed amongst different incomplete statuses and 7 were `Unknown` (Table 1). Four cases were misclassified as `sure Complete` because they were similar to unknown TrEMBL proteins of small size (116 amino acids in length on average). Six cases were classified as `Putative complete` because the 5' and 3' deletions removed only a few amino acids from the predicted protein; in fact, FULL-LENGTHERNEXT gives warning messages indicating that these testing sequences are devoid of start and stop codons but could code for a protein of a size close to the reference.

Overall reliability of FULL-LENGTHERNEXT was then assessed using the results for the CGD-MGC and CG_Trimmed datasets (Table 1). Their corresponding sensitivity, specificity, precision and accuracy (Table 2, first row) were over 0.99, close to the ideal value of 1.0, suggesting that the heuristic algorithm of FULL-LENGTHERNEXT produces reliable positive and negative results. The low relative error quotient (REQ = 0.00807; Martin *et al.*, 2004) supports that FULL-LENGTHERNEXT is a useful tool for the automatic and fast identification of sequences coding for complete and incomplete proteins. Further testing of the capabilities of FULL-LENGTHERNEXT was performed with new datasets obtained from the 981 sequences in CGD-MGC that received the `sure Complete` status, excluding the 9 entries classified as `C-terminal` and the 10 entries classified as `Putative complete`.

4.3 Robustness against sequence mistakes

The main source of mistakes in datasets were sequence indels, which provoke frame-shifts giving rise to false start and/or stop codons. Since FULL-LENGTHERNEXT should be able to cope with these errors, the CG_Ins, CG_Dels and CG_Indels datasets (see Suppl. File 3) have been created for testing this possibility. Most sequences in these datasets received a `Complete` status (Table 1), with sensitivity, specificity, precision and accuracy values over

Table 1. Status prediction for all sequence datasets used for FULL-LENGTHERNEXT validation.

Testing dataset	Complete		N-Terminal		C-Terminal		Internal	Coding		ncRNA	Unknown
	S	P	S	P	S	P		S	P		
Complete transcripts											
CGD-GB	940	33	3	1	20	0	0	2	0	0	1
CGD-MGC	981	10	0	0	9	0	0	0	0	0	0
Incomplete transcripts											
CG_Trimmed	4	6	5	20	2	23	933	0	0	0	7
Full-length artificial paired-ends sequences											
CG_paired	968	11	0	0	1	0	1	0	0	0	0
Complete transcripts with insertions											
CG_Ins1	966	12	0	0	3	0	0	0	0	0	0
CG_Ins2	961	16	0	0	4	0	0	0	0	0	0
CG_Ins3	954	19	4	0	4	0	0	0	0	0	0
CG_Ins4	969	8	0	0	4	0	0	0	0	0	0
CG_Ins5	966	10	0	0	5	0	0	0	0	0	0
CG_Ins6	920	53	2	0	4	2	0	0	0	0	0
Complete transcripts with deletions											
CG_Dels1	957	17	1	0	5	1	0	0	0	0	0
CG_Dels2	951	25	1	0	4	0	0	0	0	0	0
CG_Dels3	956	20	0	0	5	0	0	0	0	0	0
CG_Dels4	949	23	4	0	5	0	0	0	0	0	0
CG_Dels5	963	14	1	0	3	0	0	0	0	0	0
CG_Dels6	959	17	1	0	4	0	0	0	0	0	0
Complete transcripts with indels											
CG_Indels1	951	23	2	0	3	2	0	0	0	0	0
CG_Indels2	968	12	0	0	1	0	0	0	0	0	0
CG_Indels3	957	20	0	0	4	0	0	0	0	0	0
CG_Indels4	930	46	1	0	4	0	0	0	0	0	0
Coding status validation[1]											
CGD-MGC	-	-	-	-	-	-	-	725	211	29	35
NCSD	-	-	-	-	-	-	-	10	58	4	928

S: sure; P: putative

[1] CGD-MGC (coding sequences) and NCSD (non-coding sequences) datasets were used for testing the reliability of the `Coding` status based on TestCode score. FULL-LENGTHERNEXT was slightly modified to ignore the comparison with any protein database to simulate the situation of none of the testing sequences matching any hit in databases, meaning that only the C part of the algorithm depicted in Fig. 1 was used.

0.99 (Table 2), and a very low REQ (< 0.0039). Values were statistically equal ($P > 0.05$) with respect to the original sequences, demonstrating that up to 3 indels events in a sequence did not significantly alter FULL-LENGTHERNEXT predictions. Values were also statistically equal ($P > 0.05$) when all modifications were computed as a single case (Table 2, CG_all_indels row).

Another source of confusion could be the presence of blocks of Ns within sequences. This usually occurs when the transcriptome has been constructed using paired-end reads. The capability of FULL-LENGTHERNEXT to deal with this type of sequence was tested with the artificial dataset CG_paired. Again, most sequences in this dataset were assigned a `Complete` status, with sensitivity, specificity, precision and accuracy values over 0.989 (Table 2), and a very low REQ (0.00613), also providing statistically significant equality ($P > 0.05$) to the case of sequences without those N blocks (Table 2).

The success of FULL-LENGTHERNEXT predictions regarding indels and N blocks indicates that it is is useful for working with real results from tentative unigenes obtained by single-reads and

Table 2. Summary of positives and negatives observed in FULL-LENGTHERNEXT tests and its use in algorithm reliability validation

Testing dataset	TP	TN	FP	FN	Sensitivity	Specificity	Precision	Accuracy	REQ
CGD-MGC	991	2	-	6	0.99398	0.99002	0.99001	0.99199	0.00807
CG_Trimmed[1]	-	990	10	-	-	-	-	-	-
CG_ins	5854	-	-	32	0.99456	0.99002	0.99829	0.99390	0.00359
CG_dels	5851	-	-	35	0.99405	0.99002	0.99829	0.99347	0.00385
CG_indels	3907	-	-	17	0.99567	0.99002	0.99745	0.99452	0.00346
CG_all_indels[2]	15612	-	-	84	0.99465	0.99002	0.99936	0.99437	0.00301
CG_paired	979	-	-	2	0.99796	0.99002	0.98989	0.99394	0.00613
coding	936	-	-	64	0.93600	0.92800	0.92857	0.93200	0.07265
non-coding	-	928	72	-	-	-	-	-	-

TP: True Positives; TN: True Negatives; FP: False Positives; FN: False negatives. Sensitivity: the proportion of actual positives, which are correctly identified, as TP/(TP+FN). Specificity: the proportion of negatives, which are correctly identified, as TN/(TN+FP). Precision: the degree to which repeated measurements under unchanged conditions show the same results, that is, repeatability or reproducibility of the measurement, as TP/(TP+FP). Accuracy: proximity of results to the true value, as (TP+TN)/(FP+TP+FN+TN). Relative error quotient (REQ): a value for assessing prediction methods as (FN*W+FP)/TP(1+W), where low REQ represents a low proportion of errors and higher REQ indicates a higher proportion of errors (Martin *et al.*, 2004).

[1] CG_Trimmed dataset was used as source of TN and FP for the CGD-MGC dataset as well as for the derived datasets containing insertions and/or deletions, which is why specificity is the same for the first six values.

[2]CG_all_indels values are the sum of data of all insertion, deletion and indel datasets.

paired-reads assembly, while coping with the presence of sequence mistakes.

4.4 Reliability of `Coding` status

It is in the aim of FULL-LENGTHERNEXT to determine which could be species-specific genes amongst the sequences without similarity in databases. This goal was achieved by the piece of algorithm represented in Fig. 1C. The CGD-MGC dataset was used as source of TP and the NCSD as source of TN. The results of this test can be found in Table 1 (rows below "Coding status validation"). Sensitivity, specificity, precision and accuracy values were over 0.928 (Table 2), values that were statistically lower ($P < 0.05$) than for the `Complete` status, but still close to the theoretical value of 1.0. As REQ is still low (0.07265; values of REQ < 0.4 are considered good predictors [Martin *et al.*, 2004]), it can be suggested that `Coding` status is a successful indication that the transcript is likely to be species-specific and not a result of sequencing or assembling artefacts, laboratory manipulation, sequencing errors, inaccurate pre-processing (Falgueras *et al.*, 2010) or misassembling. The `Coding` status does not refer to the completeness of the sequence, since this deep analysis lacked reliability (results not shown).

Specificity and accuracy values (Table 2) suggest that most unigenes that do not receive the `Coding` status could be discarded for the analysis. However, the TestCode score is based on protein coding genes, although it is now known that there is an increasing number of biologically relevant long and short non-coding RNAs (ncRNAs) in deeply sequenced transcriptomes (Huang *et al.*, 2011). In order to recover likely ncRNAs, unigenes without a status after TestCode analysis were compared with a non-redundant ncRNA database (see Reference databases for details). Unigenes without a ncRNA homologue will receive a final `Unknown` status.

The reliability of `Coding` and `Unknown` statuses have been further studied with unigenes in the "CAP3" column in Table 3, corresponding to a FULL-LENGTHERNEXT analysis of real-world data in NGS-T (see below). Interestingly, the mean length of unigenes having an `Unknown` status is 315 nt (mode = 31 nt), while the mean length of unigenes with any other status is 756 nt (mode = 341 nt). This is evidence that unigenes with `Unknown` status might correspond to short sequences resulting from artefacts or misassemblies, or sequences so short that BLAST was not able to detect any significant similarity. Additionally, unigenes with the status of `Coding` (1495 + 1195) and `Unknown` (9619) were analysed using AutoFact (Koski *et al.*, 2005) with the same stringency than FULL-LENGTHERNEXT (E-value $< 10^{-25}$). Although AutoFact is not maintained and is not designed for high-throughput analysis, it was useful for this purpose since it uses up to 10 databases, including EST, KEGG, Pfam, UniRef90 or NR, for annotation. It must be mentioned that these databases were not filtered for complete proteins, so they may contain sequences that are absent from FULL-LENGTHERNEXT reference databases. A total of 63.9% and 61.5% of unigenes with the `sure Coding` or `Putative coding` status, respectively, were annotated with AutoFact. Annotations were mainly derived from EST databases [96.5% (1119 unigenes) in `sure Coding` and 99.4% (911 unigenes) in `Putative coding`], suggesting that these unigenes could derive from species-specific genes. As for unigenes with the `Unknown` status, 43.1% (4146 unigenes) were annotated by AutoFact, all of them but 1 from EST databases, suggesting that the remaining 5473 unigenes (56.9%) are strong candidates for being artefacts or misassemblies. Due to their short length, they can be discarded for further annotation in order to accelerate the process without a significant loss of information. However, users are invited to perform a relaxed annotation of `Unknown` unigenes using dedicated annotation tools in order to recover the unigenes that might contain useful information.

Table 3. Comparison of NGS-T sequences from a pine transcriptome assembled with Euler-SR, MIRA and CAP3 assemblers

	Euler-SR		MIRA3		CAP3[1]	
	#seqs	%	#seqs	%	#seqs	%
Unigenes	24147	100%	50813	100%	41246	100%
Unigenes > 500 bp	11729	48.57%	18453	36.32%	20115	48.77%
Longest unigene (bp)	4483		7450		7450	
With ortologue[2]	16779	69.49%	35334	69.54%	28314	68.65%
Different ortologue IDs	9997	59.58%	14334	40.57%	13452	47.51%
Complete transcripts	2766	16.49%	6328	17.91%	6164	21.77%
Different complete transcripts	2606	15.53%	4055	11.48%	4640	16.39%
Misassembled	11	0.07%	139	0.39%	30	0.11%
Without ortologue[2]	7362	30.49%	15479	30.46%	12932	31.35%
Coding	785	10.66%	2191	14.15%	1815	14.03%
Putative coding	492	6.68%	1772	11.45%	1490	11.52%
Putative ncRNA	6	0.08%	5	0.03%	8	0.06%
Unknown	6085	82.65%	11511	74.37%	9619	74.38%
Mapped reads[3]	443 976	59.50%	637321	85.41%	651 643	87.33%

[1] Due to its overlap-layout-consensus design for Sanger sequencing, CAP3 cannot be used with the huge amount of reads provided by any NGS method. It has been therefore used for reconciliation of the assemblies obtained from Euler-SR and MIRA3.

[2] Percents for subclassifications of this category were calculated using this line as 100% reference.

[3] Mapping was performed with Bowtie 2.0 using the default parameters (Langmead *et al.*, 2009) and the useful reads as input.

4.5 Identification of the best *de novo* assembly of a transcriptome

Transcriptomics projects based on *de novo* assemblies usually lack external criteria for discerning which is the best possible assembly of sequencing reads. In this context, FULL-LENGTHERNEXT could be used as a first approach for identifying unigenes derived from true transcripts by means of the information contained in the `summary_stats.html` file (see "Suppl. File 1"). As mentioned above, sequences qualified as `Misassembled` can be discarded. Unigenes with an orthologue are good candidates for well reconstructed transcripts. Depending on the assembler or the coverage, real transcripts can be reconstructed as non-contiguous contigs, or as several overlapping unigenes that correspond to the same transcript. It is deduced that the sum of unigenes with an orthologue is giving the incorrect idea that more transcripts have been reconstructed than really exists. Consequently, a better approach might be to pay attention to the number of unique, orthologous IDs, since this will give a better idea of how many different transcripts have been reconstructed. The number of unigenes receiving the `Coding` status could also be a good approach for determining the real number of species-specific genes that contain the studied transcriptome. We propose that the sum of the number of `Coding` unigenes plus the number of different orthologue IDs is the best estimate of the amount of transcriptome that an experiment has revealed, since the higher the sum, the better the assembly appears to be. Unfortunately, this simple sum does not take into account the contiguity of the assembly. We propose that the amount of different `Complete` transcripts reconstructed can give an indication of the contiguity of the assembly when no transcript reference is available. In other words, the greater the number of different `Complete` unigenes, the better the assembly is expected to be, the more reliable the assembly can be considered to be.

The preceding rationale can be illustrated with the real-world data of NGS-T obtained from Fernández-Pozo *et al.*, 2011, where MIRA3 was used for the assembly. In Table 3, the 'MIRA3' column is an assembly equivalent to this one published (Fernández-Pozo *et al.*, 2011), where the final number of unigenes with orthologues (50 813) might suggest that the complete transcriptome has been covered. However, the number of different orthologue IDs (14 334) reflects that there could still be genes to be discovered. The number of `Coding` unigenes (2191 + 1772) does not account for the missing genes if one assumes that the pine genome, such as *Arabidopsis*, could have ∼25 000 genes. Since the objective of MIRA3 is to avoid the co-assembly of different alleles or paralogues, the number of revealed transcripts could be overestimated in the 'MIRA3' column of Table 3 and the cited report (Fernández-Pozo *et al.*, 2011). Therefore, a different assembly was performed using the Eulerian assembler Euler-SR (Pavel A. Pevzner and Waterman (2001)), the algorithm of which is completely different from the overlap-layout-consensus developed for MIRA3. As expected, this assembly ('Euler-SR' column, Table 3) provides lower values than the MIRA3 column for all parameters, since Euler-SR is more sensitive to sequencing errors or allelic variations and stops transcript assembly when data do not clearly favour a sequence instance. However, percentages of unigenes > 500 bp (48.57%), of different orthologue IDs (59.58%) and different complete transcripts (15.53%) are higher, suggesting that this assembly reduces the multiplicity of unigenes for the same original transcript. Consequently, Euler-SR seems to provide the most reliable unigenes contained in NGS-T. The suggested interpretation is that Euler-SR reveals the minimum number of unigenes and MIRA3 reveals and overestimated number of unigenes, while the real number of revealed unigenes lies between both values.

A reconciliation of both assemblies with CAP3 has been carried out, since CAP3 has been described as a highly reliable *de novo*

sequence assembler for establishing unigenes from long reads (Liang *et al.*, 2000), and for this reason seems to be used in several pipelines (e.g., Kumar and Blaxter, 2010; Zheng *et al.*, 2011). The reconciliation obtained with CAP3 (Table 3, "CAP3" column) produced intermediate values for most parameters, with the prominent exception of (1) longer unigenes since 48.77% (20 115) were longer than 500 bp, suggesting that the reconciliation had reconstructed longer transcripts, even though the longest unigene had not been changed (7450 bp); (2) greater percentage of complete transcripts (21.77%), suggesting that reconciliation joins together unigenes reconstructed from the same transcript; and (3), more different complete transcripts in terms of numbers (4640) and percentages (16.39%), suggesting that more transcripts have been completely reconstructed. The useful reads used for assemblies were then mapped on the unigenes produced by the three assemblers (Table 3, "Mapped reads" row). It was observed that CAP3 reconciliation mapped more original reads than Euler-SR or MIRA3 separately, suggesting that the reconciliation provided unigenes closer to the real transcriptome.

Considering the results of Table 3 as a whole, it can be concluded that: (i) CAP3 reconciliation provides better results than each assembly separately; (ii) between 1815 and 3305 unigenes could belong to species-specific transcripts that do not have a similar sequence in databases; (iii) 9619 `Unknown` unigenes could be considered for removal for subsequent analysis. Therefore, this example illustrates the utility of FULL-LENGTHERNEXT for revealing the best approach for a *de novo* transcriptome assembly and for identifying the number of unigenes that reconstruct complete transcripts from this transcriptome. It is also interesting to observe that the approach of using CAP3 to reconcile separate assemblies seems to be a good strategy, since the same results have been confirmed by the assembly of other non-model organism transcriptome projects in our laboratory (results not shown).

5 CONCLUSION

FULL-LENGTHERNEXT, with an easy-to-use command-line, has been developed for the rapid study of eukaryotic transcriptomes from non-model organisms, although as a complement to annotation tools, rather than a substitute. Its architecture (Fig. 1) accelerates the analysis taking advantage of parallelised or distributed computation, so that it is ready to work with a large number of sequences, such as those obtained with NGS projects (Fig. 2). In addition, the optional user database containing protein sequences close to the species under study can accelerate the process and produce more accurate predictions (Fig. 2). In fact, the close-to-1.0 values of sensitivity, specificity, accuracy and precision (Table 2) demonstrated a high capability for detection of true positives and true negatives, even when the DNA sequence contains one or more frame-shifts (Table 1 and Fig. 4B). Therefore, FULL-LENGTHERNEXT should be used before any annotation tool since it is able to produce in a few hours a general picture of the transcriptome assembled, as well as measures that guide scientists in evaluating which assembly is worth further analysis.

The status assigned by FULL-LENGTHERNEXT (Table 1) is useful for retrieving promoter sequences in the laboratory using for example 5'-RACE (Suzuki *et al.*, 2001; Bae *et al.*, 1995), for evolutionary studies (Ponte *et al.*, 1984), for finding specific sequences that discriminate gene and allele-specific transcripts (Eveland *et al.*, 2008), for the generation of gene-specific sequence-tagged sites (STSs) useful in expression maps (Andrea S.Wilcox and M.Sikelal, 1991), or for studying the regulatory roles of 3'-UTRs (Rastinejad and Blau, 1993; Caput *et al.*, 1986; Pesole *et al.*, 2001). `Complete` sequences are useful for determining the genomic structure of genes (Team, 2002) and for the same applications mentioned above. The fact that the start and stop codons are particularly marked in the output `nt_seq.txt` file is useful for the automatic design of primers or probes close to these codons (J. Canales, personal communication).

FULL-LENGTHERNEXT is also useful in the detection of putative species-specific genes (Tables 1 and 3) that deserve a specific and detailed analysis in laboratories. Moreover, it is very useful for discarding putative artefactual sequences based on inconsistent orthology or the lack of significant similarity (Table 3). It has also revealed its capability for detection of the best *de novo* assembly of a transcriptome (Table 3). As a result, FULL-LENGTHERNEXT can help save time in the laboratory in the selection of the best assembly, genes of interest, useful sequences, and it can provide an idea of the transcriptome coverage obtained by any sequencing project.

Evidence has also been presented that the *de novo* assembly of a transcriptome seems to provide better results when combining assembler contigs based on two different algorithms, like Eulerian pathways and overlay-layout-consensus. Separately, these assemblies provide worse results than a reconciliation of both with CAP3. In any case, FULL-LENGTHERNEXT can be used to detect the best assembly of a particular dataset of transcriptomics reads.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the computer resources and technical support provided by the Plataforma Andaluza de Bioinformatica of the University of Malaga, Spain.

Funding: This study was supported by grants from the Spanish MICINN (BIO2009-07490), the European Union (PLE2009-0016), and Junta de Andalucía (CVI-6075), as well as institutional funding to the research group BIO-114.

REFERENCES

Andrea S.Wilcox, Akbar S.Khan, J. A. and M.Sikelal, J. (1991). Use of 3' untranslated sequences of human cdnas for rapid chromosome assignment and conversion to stss: implications for an expression map of the genome. *Nucleic Acids Research*, **19**(8), 1837–1843.

Angeloni, F., Wagemaker, C., Jetten, M., den Camp, H. O., Janssen-Megens, E., Francoijs, K., Stunnenberg, H., and Ouborg, N. (2011). De novo transcriptome characterization and development of genomic tools for scabiosa columbaria l. using next-generation sequencing techniques. *Mol Ecol Resour*, **11**(4), 662–74.

Apweiler, R., Bairoch, A., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M. J., Natale, D. A., O'Donovan, C., Redaschi, N., and Yeh, L.-S. L. (2004). Uniprot: the universal protein knowledgebase. *Nucleic Acids Res*, **32**(Database issue), D115–9.

Bae, H., Geiser, A., Kim, D., Chung, M., Burmester, J., Sporn, M., Roberts, A., and Kim, S. (1995). Characterization of the promoter region of the human transforming growth factor-ß type ii receptor gene. *The Journal of Biological Chemistry*, **270**(49), 29460–29468.

Bu, D., Yu, K., Sun, S., Xie, C., Skogerbø, G., Miao, R., Xiao, H., Liao, Q., Luo, H., Zhao, G., Zhao, H., Liu, Z., Liu, C., Chen, R., and Zhao, Y. (2012). Noncode v3.0: integrative annotation of long noncoding rnas. *Nucleic Acids Research*, **40**(D1), D210–D215.

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., and Madden, T. L. (2009). Blast+: architecture and applications. *BMC Bioinformatics*, **10**, 421.

Caput, D., Beutler, B., Hartog, K., Thayer, R., Brown-Shimer, S., and Cerami, A. (1986). Identification of a common nucleotide sequence in the 3'-untranslated region of mrna molecules specifying inflammatory mediators. *Proc.Natl.Acad.Sci.*, **83**, 1670–1674.

Chevreux, B., Pfisterer, T., Drescher, B., Driesel, A., Müller, W., Wetter, T., and Suhai, S. (2004). Using the miraest assembler for reliable and automated mrna transcript assembly and snp detection in sequenced ests. *Genome Research*, **14**, 1147–1159.

Eveland, A. L., McCarty, D. R., and Koch, K. E. (2008). Transcript profiling by 3'-untranslated region sequencing resolves expression of gene families. *Plant Physiol*, **146**(1), 32–44.

Falgueras, J., Lara, A. J., Fernández-Pozo, N., Cantón, F. R., Pérez-Trabado, G., and Claros, M. G. (2010). Seqtrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics*, **11**, 38.

Fernández-Pozo, N., Canales, J., Guerrero-Fernández, D., P. Villalobos, D., M. Díaz-Moreno, S., Bautista, R., Flores-Monterroso, A., Guevara, M., Perdiguero, P., Collada, C., Cervera, M., Soto, A., Ordás, R., R. Cantón, F., Avila, C., Cánovas, F., and Claros, M. (2011). Europinedb: a high-coverage web database for maritime pine transcriptome. *BMC Genomics*, **12**(366).

Fickett, J. W. (1982). Recognition of protein coding regions in dna-sequences. *Nucleic Acids Research*, **10**(17), 5303–5318.

Griffiths-Jones, S., Moxon, S., Marshall, M., Khanna, A., Eddy, S. R., and Bateman, A. (2005). Rfam: annotating non-coding rnas in complete genomes. *Nucleic Acids Research*, **33**(suppl 1), D121–D124.

Huang, R., Jaritz, M., Guenzl, P., Vlatkovic, I., Sommer, A., Tamir, I. M., Marks, H., Klampfl, T., Kralovics, R., Stunnenberg, H. G., Barlow, D. P., and Pauler, F. M. (2011). An rna-seq strategy to detect the complete coding and non-coding transcriptome including full-length imprinted macro ncrnas. *PLoS One*, **6**(11), e27288.

Kim, H.-J., Baek, K.-H., Lee, S.-W., Kim, J., Lee, B.-W., Cho, H.-S., Kim, W. T., Choi, D., and Hur, C.-G. (2008). Pepper est database: comprehensive in silico tool for analyzing the chili pepper (capsicum annuum) transcriptome. *BMC Plant Biol*, **8**, 101.

Koop, B. F., von Schalburg, K. R., Leong, J., Walker, N., Lieph, R., Cooper, G. A., Robb, A., Beetz-Sargent, M., Holt, R. A., Moore, R., Brahmbhatt, S., Rosner, J., Rexroad, 3rd, C. E., McGowan, C. R., and Davidson, W. S. (2008). A salmonid est genomic study: genes, duplications, phylogeny and microarrays. *BMC Genomics*, **9**, 545.

Koski, L. B., Gray, M. W., Lang, B. F., and Burger, G. (2005). Autofact: an automatic functional annotation and classification tool. *BMC Bioinformatics*, **6**, 151.

Kristiansson, E., Asker, N., Förlin, L., and Larsson, D. (2009). Characterization of the zoarces viviparus liver transcriptome using massively parallel pyrosequencing. *BMC Genomics*, **10**, 345.

Kumar, S. and Blaxter, M. (2010). Comparing de novo assemblers for 454 transcriptome data. *BMC Genomics*, **11**(1), 571.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, **10**(3), R25.

Lara, A., Pérez-Trabado, G., Villalobos, D., Díaz-Moreno, S., Cantón, F., and Claros, M. G. (2007). *A Web Tool to Discover Full-Length Sequences: Full-Lengther*, pages 361–368. Springer.

Li, R., Yu, C., Li, Y., Lam, T., Yiu, S., Kristiansen, K., and Wang, J. (2009). Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, **25**(15), 1966–7.

Li, W. and Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**(13), 1658–9.

Liang, F., Holt, I., Pertea, G., Karamycheva, S., Salzberg, S. L., and Quackenbush, J. (2000). An optimized protocol for analysis of est sequences. *Nucleic Acids Res*, **28**(18), 3657–65.

Malik, A., Korol, A., Hübner, S., Hernandez, A. G., Thimmapuram, J., Ali, S., Glaser, F., Paz, A., Avivi, A., and Band, M. (2011). Transcriptome sequencing of the blind subterranean mole rat, spalax galili: Utility and potential for the discovery of novel evolutionary patterns. *PLoS One*, **6**(8), e21227.

Martin, D. M. A., Berriman, M., and Barton, G. J. (2004). Gotcha: a new method for prediction of protein function assessed by the annotation of seven genomes. *BMC Bioinformatics*, **5**, 178.

Min, X. J., Butler, G., Storms, R., and Tsang, A. (2005). Targetidentifier: a webserver for identifying full-length cdnas from est sequences. *Nucleic Acids Res*, **33**(Web Server issue), W669–72.

Parchman, T., Geist, K., Grahnen, J., Benkman, C., and Buerkle, C. (2010). Transcriptome sequencing in an ecologically important tree species: assembly, annotation, and marker discovery. *BMC Genomics*, **11**(180).

Paterson, A. H., Freeling, M., Tang, H., and Wang, X. (2010). Insights from the comparison of plant genome sequences. *Annu Rev Plant Biol*, **61**, 349–72.

Pavel A. Pevzner, H. T. and Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. *PNAS*, **98**(17), 9748–9753.

Pesole, G., Mignone, F., Gissi, C., Grillo, G., Licciulli, F., and Liuni, S. (2001). Structural and functional features of eukaryotic mrna untranslated regions. *Gene*, **276**(1-2), 73–81.

Ponte, P., Ng, S., Engel, J., Gunning, P., and Kedes, L. (1984). Evolutionary conservaton in the untranslated regions of actin mrnas: Dna sequence of a human beta-actin cdna. *Nucleic Acids Research*, **12**(3), 1687–1696.

Ralph, S. G., Chun, H. J. E., Kolosova, N., Cooper, D., Oddy, C., Ritland, C. E., Kirkpatrick, R., Moore, R., Barber, S., Holt, R. A., Jones, S. J. M., Marra, M. A., Douglas, C. J., Ritland, K., and Bohlmann, J. (2008). A conifer genomics resource of 200,000 spruce (picea spp.) ests and 6,464 high-quality, sequence-finished full-length cdnas for sitka spruce (picea sitchensis). *BMC Genomics*, **9**, 484.

Rastinejad, F. and Blau, H. M. (1993). Genetic complementation reveals a novel regulatory role for 3'untranslated regions in growth and differentiation. *Cell*, **72**, 903–917.

Semova, N., Storms, R., John, T., Gaudet, P., Ulycznyj, P., Min, X. J., Sun, J., Butler, G., and Tsang, A. (2006). Generation, annotation, and analysis of an extensive aspergillus niger est collection. *BMC Microbiol*, **6**, 7.

Suzuki, Y., Tsunoda, T., Sese, J., Taira, H., Mizushima-Sugano, J., Hata, H., Ota, T., Isogai, T., Tanaka, T., Nakamura, Y., Suyama, A., Sakaki, Y., Morishita, S., Okubo, K., and Sugano, S. (2001). Identification and characterization of the potential promoter regions of 1031 kinds of human genes. *Genome Res*, **11**(5), 677–84.

Team, M. G. C. M. P. (2002). Generation and initial analysis of more than 15,000 full-length human and mouse cdna sequences. *PNAS*, **99**(26), 16899–16903.

Vera, J. C., Wheat, C. W., Fescemyer, H. W., Frilander, M. J., Crawford, D. L., Hanski, I., and Marden, J. H. (2008). Rapid transcriptome characterization for a nonmodel organism using 454 pyrosequencing. *Mol Ecol*, **17**(7), 1636–47.

Wang, S., Peatman, E., Abernathy, J., Waldbieser, G., Lindquist, E., Richardson, P., Lucas, S., Wang, M., Li, P., Thimmapuram, J., Liu, L., Vullaganti, D., Kucuktas, H., Murdock, C., Small, B., Wilson, M., Liu, H., Jiang, Y., Lee, Y., Chen, F., Lu, J., Wang, W., Xu, P., Somridhivej, B., Baoprasertkul, P., Quilang, J., Sha, Z., Bao, B., Wang, Y., Wang, Q., Takano, T., Nandi, S., Liu, S., Wong, L., Kaltenboeck, L., Quiniou, S., Bengten, E., Miller, N., Trant, J., Rokhsar, D., Liu, Z., and the Catfish Genome Consortium (2010). Assembly of 500,000 inter-specific catfish expressed sequence tags and large scale gene-associated marker development for whole genome association studies. *Genome Biology*, **11**.

Zheng, Y., Zhao, L., Gao, J., and Fei, Z. (2011). iassembler: a package for de novo assembly of roche-454/sanger transcriptome sequences. *BMC Bioinformatics*, **12**(1), 453.

SUPPLEMENTARY FILE 1: HOW TO USE FULL-LENGTHERNEXT

Unigene sequences to be analysed by FULL-LENGTHERNEXT must be within a multifasta file that is loaded using the option `-f`. Other important parameters must be taken into account: (1) the `-g DBgroup`, indicating the database division (i.e., fungi, human, invertebrates, mammals, plants, rodents or vertebrates) to which unigenes belong, (2) the `-w workers`, indicating the number of cores to be used for parallelisation or distribution, that must be greater than 1 (default = 2), (3) the `-c chunkSize`, indicating the number of sequences that will be sent per CPU core (default = 200), (4) the minimum `identity` considered significant homology (default = 45%), (5) the minimum *E*`-value` for BLAST (default = 1^{-25}), (6) the `-m maxDist`, which is the maximal distance between problem (query or input) and reference (subject) gene boundaries to be qualified as putative (default = 15 amino acids), and (7) the `-u UserDB`, indicating the user database of complete proteins. One example of the minimal execution command line is:

```
full_lengther_next -f input.fasta -g mammals
```

the complete execution command being:

```
full_lengther_next -f input.fasta -g DBgroup -c chunkSize -i identity
                   -e eValue      -m maxDist -u UserDB    -w workers
```

Result files are gathered inside the `fln_results` folder, containing:

- A summary of the unigene statuses for statistical purposes (`summary_stats.html`).
- Detailed information of every unigene with a subject at any protein database. (`dbannotated.txt`); for every entry status, a subject accession number, subject description, warning messages, translated protein and BLASTx indexes are available in a tab-delimited text format.
- Detailed information of unigenes that could code for species-specific proteins (`new_coding.txt`).
- Detailed information of unigenes which could be non-coding RNAs (`nc_rnas.txt`).
- Nucleotide sequences of every coding region (`nt_seq.txt`), highlighting the ORF using marks at the start and stop codons to allow for a rapid identification of the ORF and the untranslated regions.
- The protein sequences coded by unigenes in fasta format (`proteins.fasta`).

SUPPLEMENTARY FILE 2: DETAILED DESCRIPTION OF FULL-LENGTHERNEXT STATUSES

N-terminal : unigenes lacking the 3'-end coding region of the transcript and whose start codon was found at the expected position (not beyond the `maxDist`) compared to a reliable reference (values meeting the minimum `identity` and *E*`-value` parameters), or an in-frame stop codon occurs before the start codon (see Suppl. File 1 for details about cited parameter meaning). This status includes the qualification of `Putative` in unigenes lacking the 3'-end coding region of the transcript and whose start codon, not being beyond `maxDist` positions of the reference start codon, can be contained within the 5'-end of the unigene. Qualification of `Putative` is also given when an in-frame stop codon is located upstream of an ATG that is not located in the expected position; and also when the ATG is correctly located, but there is no in-frame stop codon, or the reference sequence has an identity with the problem sequence below the `identity` threshold.

C-terminal : unigenes lacking the 5'-end coding region of the transcript and whose stop codon was found at the expected position (not beyond the `maxDist`) compared to the reference (values complying the minimum `identity` and *E*`-value` thresholds) (see Suppl. File 1 for details about cited parameter meaning). This status includes the qualification of `Putative` in unigenes lacking the 5'-end coding region of the transcript and whose stop codon was found beyond the expected position by a distance shorter than `maxDist` positions, considering the stop codon of the reference sequence. Qualification of `Putative` is also given to unigenes that clearly extend downstream of the reference sequence, even though the stop codon is not located where expected.

Complete : unigenes coding for a complete protein. Start codon and end codon were qualified as `N-terminal` and `C-terminal`. This status includes the qualification of `Putative` in unigenes that might code for a complete protein but the start or stop codons were qualified as `Putative`, or when the ORF predicted by FULL-LENGTHERNEXT is 100 bp shorter than expected compared to the reference sequence.

Internal : unigenes with a hit in a database that lack the 5'-end (no start codon) and 3'-end (no stop codon) of the transcript.

Misassembled : unigenes showing BLAST hits with the same subject (reference sequence) in sense and antisense.

Coding : unigenes containing an ORF longer than 200 bp without any similarity in protein databases, but whose TestCode score is higher than 0.95. When the TestCode score lies between 0.74 and 0.95, the unigene will receive the qualification of `Putative`.

Putative ncRNA : unigenes with similarity within the ncRNA database.

Unknown : unigenes without any similarity in any database and with a TestCode score < 0.74.

SUPPLEMENTARY FILE 3: SEQUENCE IDS OF ENTRIES IN COMPLETE GENE DATASETS USED FOR FULL-LENGTHERNEXT VALIDATION

IDs of sequences included in CGD-GB

AH013652, AY532090, AY601914, AY724812, AY724935, AY724936, AY724944, AY724945, AY724946, AY724953, AY724954, AY724957, AY724958, AY724959, AY724960, AY724962, AY724966, AY787127, AY792326, AY845176, AY856830, AY916765, AY935536, AY947481, AY971956, AY971957, AY973034, BC000017, BC000053, BC000102, BC000104, BC000173, BC000306, BC000310, BC000322, BC000323, BC000324, BC000325, BC000327, BC000336, BC000341, BC000344, BC000380, BC000393, BC000416, BC000442, BC000478, BC000481, BC000504, BC000506, BC000542, BC000543, BC000554, BC000559, BC000562, BC000635, BC000645, BC000650, BC000661, BC000680, BC000738, BC001009, BC001058, BC001070, BC001075, BC001080, BC001101, BC001119, BC001352, BC001355, BC001357, BC001374, BC001387, BC001430, BC001444, BC001462, BC001463, BC001471, BC001494, BC001519, BC001594, BC001825, BC001843, BC001844, BC001861, BC002427, BC002429, BC002449, BC002466, BC002477, BC002498, BC002509, BC002513, BC002539, BC002594, BC002627, BC002631, BC002648, BC002661, BC002669, BC002675, BC002715, BC002719, BC002723, BC002759, BC002762, BC002778, BC002806, BC002807, BC002809, BC002816, BC002839, BC002842, BC002844, BC002863, BC002869, BC002876, BC002896, BC002909, BC002912, BC003658, BC004121, BC004214, BC004216, BC004485, BC004536, BC004545, BC005137, BC005171, BC005807, BC005821, BC005832, BC005928, BC006096, BC006194, BC006206, BC006215, BC006221, BC006273, BC006324, BC006327, BC006346, BC006362, BC006378, BC006399, BC006490, BC006491, BC006492, BC006495, BC006501, BC006521, BC006525, BC006529, BC006558, BC006564, BC006566, BC007313, BC007451, BC007661, BC007675, BC007679, BC007722, BC007765, BC007810, BC007840, BC007904, BC007978, BC008036, BC008078, BC008716, BC008720, BC008742, BC008770, BC008819, BC008828, BC008851, BC008869, BC008874, BC008881, BC008908, BC009177, BC009194, BC009382, BC009501, BC009503, BC009682, BC009828, BC009850, BC009964, BC009988, BC009999, BC010044, BC010089, BC010101, BC010136, BC010145, BC010167, BC010181, BC010708, BC010874, BC010929, BC010993, BC011171, BC011605, BC011656, BC011669, BC011672, BC011705, BC011710, BC011714, BC011748, BC011760, BC011778, BC011826, BC011835, BC011849, BC011865, BC011934, BC011936, BC012421, BC012481, BC012527, BC012543, BC012618, BC012731, BC012733, BC012801, BC012815, BC013041, BC013044, BC013362, BC013366, BC013375, BC013393, BC013903, BC013983, BC013985, BC013998, BC014004, BC014031, BC014033, BC014044, BC014117, BC014123, BC014133, BC014152, BC014217, BC014225, BC014264, BC014271, BC014272, BC014277, BC014526, BC015041, BC015219, BC015220, BC015371, BC015525, BC015571, BC015621, BC015626, BC015649, BC015650, BC015917, BC015969, BC016320, BC016486, BC016832, BC017089, BC017188, BC017212, BC017304, BC017327, BC017361, BC017365, BC017469, BC017585, BC017693, BC018119, BC018636, BC018700, BC019034, BC019040, BC019043, BC019047, BC020968, BC020994, BC021198, BC021204, BC021262, BC021301, BC021719, BC021962, BC022785, BC023504, BC023520, BC023521, BC023539, BC023546, BC023550, BC023588, BC023977, BC024005, BC024030, BC024181, BC024190, BC024208, BC024243, BC025367, BC025422, BC025978, BC025987, BC026101, BC026156, BC026344, BC027595, BC027603, BC027720, BC027895, BC028364, BC028369, BC028382, BC028391, BC028397, BC028566, BC028599, BC028708, BC028711, BC028740, BC028744, BC030148, BC030572, BC030587, BC030614, BC030707, BC030996, BC031522, BC031820, BC032145, BC032592, BC032835, BC032839, BC032850, BC032861, BC032948, BC033015, BC033034, BC033694, BC033777, BC033785, BC033880, BC034225, BC034957, BC034961, BC034977, BC035698, BC035716, BC036449, BC036457, BC036460, BC036515, BC036680, BC036703, BC036704, BC036757, BC037296, BC038099, BC038100, BC038970, BC040020, BC041338, BC042943, BC043144, BC043273, BC045167, BC046354, BC047681, BC050595, BC051858, BC051883, BC052604, BC052608, BC052812, BC052973, BC052998, BC053316, BC053348, BC053349, BC053508, BC053578, BC053584, BC053617, BC053629, BC053667, BC053857, BC053869, BC053886, BC053983, BC053991, BC053992, BC054003, BC054014, BC054345, BC054347, BC054489, BC054499, BC054865, BC055286, BC055314, BC055396, BC056141, BC056143, BC056152, BC056154, BC056249, BC056257, BC056408, BC056414, BC056878, BC056891, BC056906, BC057237, BC057774, BC057796, BC057801, BC057813, BC058000, BC058009, BC058037, BC058284, BC058862, BC058918, BC058928, BC059375, BC059388, BC060758, BC060759, BC060786, BC060789, BC060804, BC060827, BC060828, BC060838, BC060842, BC060853, BC060855, BC060870, BC061581, BC061584, BC061588, BC061590, BC061903, BC061914, BC062211, BC062223, BC062323, BC062427, BC062565, BC062605, BC062613, BC062625, BC062751, BC062998, BC063037, BC063043, BC063107, BC063130, BC063289, BC063308, BC063407, BC063411, BC063415, BC063423, BC063471, BC063490, BC063513, BC063549, BC063696, BC063697, BC063795, BC063825, BC063829, BC063844, BC063878, BC063882, BC064380, BC064382, BC064420, BC064422, BC064425, BC064528, BC064547, BC064567, BC064608, BC064912, BC064916, BC064945, BC064965, BC064976, BC065192, BC065265, BC065276, BC065294, BC065297, BC065300, BC065499, BC065516, BC065529, BC065544, BC065552, BC065560, BC065563, BC065569, BC065730, BC065832, BC065926, BC065936, BC066116, BC066238, BC066304, BC066334, BC066592, BC066658, BC066929, BC066981, BC067078, BC067084, BC067090, BC067128, BC067255, BC067257, BC067259, BC067260, BC067266, BC067299, BC067350, BC067427, BC067531, BC067731, BC067770, BC067827, BC067861, BC068013, BC068200, BC068449, BC068467, BC068494, BC068534, BC068541, BC068574, BC069012, BC069058, BC069078, BC069156, BC069193, BC069207, BC069233, BC069290, BC069413, BC069476, BC069495, BC069507, BC069598, BC069626, BC069651, BC069734, BC069760, BC069786, BC069817, BC070040, BC070066, BC070075, BC070082, BC070108, BC070149, BC070155, BC070162, BC070182, BC070219, BC070259, BC070260, BC070261, BC070293, BC070294, BC070338, BC070347, BC071181, BC071564, BC071603, BC071604, BC071627, BC071665, BC071694, BC071722, BC071754, BC071761, BC071810, BC071834, BC071857, BC071964, BC072015, BC072385, BC072406, BC072409, BC072420,

BC072428, BC072439, BC073151, BC073156, BC073800, BC073922, BC073964, BC074720, BC074814, BC074863, BC074927, BC074931, BC074957, BC075030, BC075043, BC075047, BC078143, BC078144, BC080187, BC080189, BC080532, BC080554, BC080600, BC082753, BC082755, BC082766, BC082974, BC082985, BC082986, BC082990, BC084567, BC084579, BC084582, BC084583, BC085004, BC085605, BC086306, BC087845, BC089042, BC089400, BC089426, BC090057, BC090871, BC090921, BC090956, BC092404, BC092413, BC092432, BC092445, BC092468, BC092498, BC092512, BC092517, BC093005, BC093009, BC093044, BC093062, BC093066, BC093082, BC093693, BC093727, BC093777, BC093862, BC093864, BC093910, BC093952, BC093981, BC094000, BC094707, BC094715, BC094753, BC094795, BC094823, BC094832, BC094881, BC095394, BC095421, BC095434, BC095442, BC095443, BC095452, BC095482, BC095512, BC095538, BC096070, BC096090, BC096099, BC096101, BC096105, BC096127, BC096139, BC096156, BC096165, BC096175, BC096183, BC096190, BC096207, BC096256, BC096258, BC096260, BC096276, BC096277, BC096295, BC096298, BC096301, BC096327, BC096331, BC096351, BC096733, BC096827, BC096835, BC098377, BC098387, BC098390, BC098398, BC098404, BC098418, BC098562, BC098580, BC099634, BC099650, BC099842, BC100667, BC100772, BC100795, BC100807, BC100809, BC100823, BC100854, BC100867, BC100872, BC100886, BC100893, BC100904, BC100911, BC100918, BC100929, BC100938, BC100944, BC100954, BC100968, BC101000, BC101005, BC101028, BC101029, BC101033, BC101045, BC101069, BC101093, BC101097, BC101107, BC101135, BC101152, BC101158, BC101175, BC101176, BC101188, BC101345, BC101351, BC101353, BC101354, BC101357, BC101466, BC101481, BC101519, BC101551, BC101559, BC101634, BC101645, BC101649, BC101727, BC101728, BC101744, BC101748, BC101837, BC103748, BC103758, BC103812, BC103824, BC103827, BC103845, BC103853, BC103857, BC103859, BC103878, BC103879, BC103890, BC103893, BC103905, BC103908, BC103918, BC103931, BC103947, BC103948, BC103949, BC103968, BC103984, BC104007, BC104040, BC104164, BC104167, BC104188, BC104195, BC104214, BC104227, BC104231, BC104244, BC104247, BC104253, BC104412, BC104420, BC104424, BC104429, BC104443, BC104457, BC104463, BC104465, BC104779, BC104796, BC104823, BC104884, BC104894, BC104906, BC104916, BC104948, BC104957, BC105043, BC105059, BC105088, BC105106, BC105999, BC106065, BC106072, BC106709, BC106712, BC106713, BC106714, BC106722, BC106732, BC106752, BC106758, BC106872, BC106921, BC106924, BC106930, BC106934, BC106942, BC106946, BC107043, BC107075, BC107082, BC107097, BC107111, BC107123, BC107148, BC107158, BC107159, BC107425, BC107586, BC107714, BC107719, BC107737, BC107750, BC107951, BC108255, BC108735, BC108890, BC108896, BC108903, BC108904, BC108909, BC108914, BC108920, BC108927, BC109065, BC109069, BC109080, BC109094, BC109097, BC109101, BC109103, BC109117, BC109125, BC109198, BC109214, BC109222, BC109241, BC109246, BC109251, BC109260, BC109263, BC109273, BC109280, BC109282, BC109296, BC109304, BC110067, BC110297, BC110299, BC110318, BC110325, BC110337, BC110338, BC110351, BC110352, BC110354, BC110374, BC110446, BC110447, BC110452, BC110453, BC110496, BC110539, BC110585, BC110588, BC110597, BC110618, BC110810, BC110834, BC110858, BC110865, BC110873, BC110878, BC110890, BC110898, BC110905, BC110910, BC110995, BC111004, BC111007, BC111021, BC111044, BC111057, BC111517, BC111534, BC111548, BC111550, BC111697, BC111702, BC111743, BC111790, BC111800, BC111936, BC111983, BC112047, BC112059, BC112060, BC112072, BC112089, BC112095, BC112140, BC112171, BC112179, BC112180, BC112198, BC112228, BC112236, BC112272, BC112304, BC112343, BC112392, BC112945, BC112963, BC113875, BC113926, BC113949, BC114213, BC114338, BC114354, BC114429, BC114468, BC114948, BC115380, BC115383, DQ018110, DQ019999, DQ026418, DQ026429, DQ080434, DQ086801, DQ097177, DQ106398, DQ106400, DQ126297, DQ142911, DQ144974, DQ186413, DQ282144, DQ301480, DQ305975, DQ336121, DQ336127, DQ364250, DQ386730, DQ419529

IDs of sequences included in CGD-MGC

BC004361, BC000196, BC000569, BC003596, BC001533, BC000218, BC014514, BC008081, BC001411, BC000585, BC008742, BC003046, BC003184, BC001381, BC003185, BC003186, BC024035, BC003047, BC000242, BC003048, BC001722, BC000637, BC001214, BC002619, BC000691, BC003187, BC013420, BC001470, BC009472, BC006485, BC001420, BC003661, BC003079, BC000618, BC000112, BC000195, BC008732, BC000780, BC001419, BC001594, BC000241, BC000666, BC017594, BC005002, BC002491, BC008367, BC002437, BC005889, BC001215, BC017172, BC002597, BC001100, BC001101, BC002559, BC000781, BC001103, BC002415, BC010886, BC019328, BC007658, BC001104, BC001725, BC004235, BC001379, BC002524, BC002334, BC001804, BC003080, BC000573, BC002481, BC001623, BC000655, BC003052, BC001396, BC000687, BC001390, BC001537, BC000022, BC000684, BC017180, BC005003, BC001016, BC000646, BC017176, BC009929, BC006524, BC009930, BC001024, BC004236, BC003190, BC002502, BC001401, BC017192, BC001958, BC000591, BC000244, BC003599, BC000259, BC018788, BC000668, BC002510, BC003053, BC000616, BC005890, BC001105, BC008082, BC002563, BC003054, BC000785, BC001218, BC003663, BC005174, BC000677, BC005176, BC000245, BC000600, BC001740, BC019330, BC006211, BC019363, BC001687, BC014388, BC000192, BC000246, BC001219, BC004362, BC008971, BC000633, BC007664, BC001539, BC003600, BC001220, BC001608, BC000191, BC000041, BC001716, BC001463, BC000607, BC001959, BC001702, BC009931, BC001106, BC019331, BC002583, BC001427, BC001609, BC000017, BC000653, BC001457, BC007455, BC001221, BC001960, BC000663, BC008746, BC014389, BC006487, BC003601, BC001029, BC018841, BC001467, BC002335, BC003602, BC003665, BC004238, BC008083, BC001375, BC001622, BC001961, BC000008, BC001371, BC003193, BC000786, BC001222, BC001962, BC002542, BC002473, BC003081, BC000048, BC003666, BC003055, BC010887, BC006525, BC001711, BC011607, BC000285, BC002535, BC000049, BC005004, BC000703, BC000704, BC014515, BC001223, BC003082, BC019332, BC002336, BC000561, BC003056, BC003605, BC008084, BC001357, BC003194, BC001225, BC005177, BC013966, BC007456, BC008733, BC006214, BC001473, BC000284, BC000659, BC001428, BC000669, BC005005, BC001108, BC001706, BC001366, BC001442, BC024036, BC001635, BC002948,

BC002469, BC003607, BC000680, BC001040, BC002472, BC000709, BC007313, BC001140, BC009746, BC011596, BC006527, BC002497, BC001471, BC002492, BC000623, BC001586, BC001421, BC001727, BC007314, BC000683, BC000096, BC002950, BC000107, BC001863, BC001141, BC002560, BC007315, BC000024, BC004402, BC001446, BC000711, BC000126, BC001042, BC003547, BC008036, BC009752, BC001595, BC014390, BC019252, BC000051, BC001627, BC004119, BC003087, BC001658, BC007660, BC000694, BC001659, BC017365, BC003088, BC001613, BC001142, BC000025, BC001451, BC003548, BC002446, BC001866, BC001626, BC009763, BC006194, BC001398, BC017175, BC000635, BC001621, BC000566, BC001144, BC014636, BC003549, BC001145, BC000104, BC004349, BC002442, BC001044, BC003550, BC001109, BC001110, BC007672, BC001013, BC025414, BC017366, BC003551, BC004420, BC003552, BC002464, BC008767, BC001146, BC000567, BC001870, BC000101, BC000713, BC001689, BC012316, BC008740, BC002506, BC001147, BC008037, BC002444, BC001022, BC001871, BC001033, BC002448, BC000706, BC007656, BC000036, BC018823, BC017367, BC001387, BC000714, BC001690, BC000715, BC000601, BC001440, BC001734, BC003608, BC001148, BC001625, BC009175, BC022845, BC002520, BC008747, BC012530, BC008764, BC002609, BC002443, BC021965, BC019253, BC001873, BC001874, BC007317, BC007318, BC007319, BC001765, BC008039, BC001376, BC000716, BC000717, BC021192, BC003553, BC001046, BC003090, BC006493, BC007320, BC002447, BC001482, BC002453, BC004129, BC001355, BC003060, BC002955, BC003554, BC002487, BC002439, BC017369, BC001047, BC001766, BC014392, BC002586, BC007655, BC001444, BC002532, BC000091, BC000718, BC003609, BC008763, BC007321, BC001152, BC003091, BC001435, BC001767, BC000719, BC011599, BC017188, BC002488, BC004963, BC001660, BC000649, BC001441, BC006534, BC000632, BC001472, BC009806, BC003610, BC001484, BC001878, BC009808, BC001731, BC002956, BC000670, BC002489, BC001485, BC000062, BC001025, BC016155, BC003092, BC001423, BC003061, BC017178, BC004130, BC004132, BC001880, BC021967, BC014638, BC002578, BC008770, BC000102, BC000682, BC001438, BC001111, BC000590, BC001154, BC002959, BC005136, BC001881, BC001486, BC001661, BC010878, BC007323, BC001036, BC001768, BC006494, BC016758, BC001691, BC006535, BC002429, BC005137, BC009177, BC003613, BC000639, BC003555, BC004421, BC003556, BC021193, BC001155, BC000544, BC005115, BC000019, BC000720, BC001113, BC001156, BC001769, BC001386, BC002436, BC001466, BC001380, BC003093, BC001157, BC013348, BC001114, BC001454, BC001158, BC001883, BC001050, BC001884, BC001662, BC000235, BC001588, BC001771, BC000127, BC002960, BC008734, BC006495, BC007333, BC014301, BC002513, BC008906, BC000028, BC016027, BC000283, BC002479, BC000187, BC007340, BC005138, BC002461, BC001596, BC003062, BC001159, BC004136, BC003094, BC003615, BC001031, BC000069, BC000721, BC002573, BC001886, BC000722, BC002544, BC004137, BC002527, BC000674, BC000093, BC000629, BC000723, BC015557, BC003616, BC002515, BC000724, BC003557, BC008726, BC004352, BC014397, BC002549, BC001161, BC001887, BC001359, BC000010, BC013903, BC001634, BC005139, BC007348, BC001738, BC002567, BC001633, BC002485, BC000725, BC000097, BC020170, BC007349, BC002962, BC000627, BC005116, BC000652, BC021085, BC000070, BC002505, BC000038, BC001692, BC000581, BC008765, BC001163, BC001744, BC002503, BC001708, BC001392, BC000023, BC004138, BC001019, BC000603, BC001773, BC003096, BC017452, BC001052, BC001720, BC004892, BC008727, BC002585, BC003617, BC008929, BC000576, BC000661, BC023975, BC001164, BC017198, BC002577, BC005140, BC001888, BC000057, BC001664, BC023503, BC017199, BC014561, BC013968, BC001756, BC001422, BC002426, BC006496, BC000249, BC004964, BC000250, BC008750, BC001889, BC001165, BC000011, BC001115, BC002591, BC000114, BC000594, BC002965, BC000654, BC002539, BC000726, BC008777, BC004139, BC000660, BC003097, BC003098, BC000727, BC000578, BC001491, BC008730, BC017378, BC004140, BC017564, BC009235, BC005827, BC009503, BC000251, BC000631, BC002454, BC000117, BC001890, BC002579, BC004141, BC005141, BC001166, BC011601, BC021981, BC008930, BC017194, BC000054, BC003064, BC003558, BC000252, BC002599, BC002445, BC000185, BC001891, BC001492, BC025372, BC002555, BC000671, BC006458, BC006459, BC000644, BC003065, BC014563, BC007401, BC000728, BC000278, BC016759, BC004143, BC003100, BC000006, BC002530, BC000043, BC017565, BC007402, BC000260, BC002967, BC000729, BC001116, BC015961, BC006195, BC003559, BC000588, BC000213, BC001732, BC003560, BC001693, BC019256, BC001777, BC000039, BC001493, BC021190, BC001892, BC000574, BC001167, BC001712, BC019257, BC001665, BC000012, BC002968, BC017197, BC000657, BC001752, BC000253, BC000094, BC002970, BC001666, BC000550, BC016760, BC001778, BC008768, BC001055, BC001894, BC002432, BC002441, BC001465, BC003101, BC001169, BC003102, BC001056, BC000642, BC006460, BC000658, BC001416, BC000730, BC003103, BC001391, BC001779, BC011604, BC001808, BC000609, BC013382, BC003104, BC001709, BC001117, BC004965, BC017193, BC001599, BC007491, BC001494, BC017187, BC003619, BC009894, BC005143, BC002511, BC000029, BC000692, BC002468, BC005830, BC008745, BC000645, BC003561, BC001057, BC023976, BC006537, BC001600, BC001118, BC000650, BC000731, BC001639, BC000098, BC002525, BC003105, BC000732, BC002618, BC000617, BC001119, BC001364, BC003106, BC002604, BC001898, BC003563, BC024043, BC002558, BC002564, BC000044, BC009470, BC001496, BC001395, BC001172, BC003067, BC001430, BC000598, BC003107, BC003620, BC002592, BC002434, BC000055, BC000597, BC000548, BC000075, BC000543, BC009895, BC013383, BC013910, BC012798, BC008741, BC023504, BC006538, BC002973, BC001780, BC008775, BC005145, BC002541, BC001453, BC000595, BC000045, BC009896, BC007662, BC001058, BC001449, BC014431, BC000614, BC001641, BC001358, BC001602, BC003565, BC001383, BC006539, BC001173, BC000733, BC000053, BC003566, BC001120, BC000734, BC003108, BC014564, BC000266, BC000571, BC001497, BC002600, BC009898, BC007659, BC003567, BC007403, BC000060, BC001900, BC002517, BC012333, BC003568, BC002582, BC001450, BC003569, BC000183, BC004893, BC006462, BC001901, BC001782, BC006463, BC000046, BC001439, BC001642, BC000736, BC018648, BC003622, BC014565, BC000234, BC014566, BC000737, BC000738, BC001174, BC002538, BC001462, BC002430, BC018649, BC000739, BC001060, BC001123, BC001632, BC001620, BC003110, BC007404, BC001410, BC008061, BC007405, BC003623, BC036703, BC014433, BC006497, BC003068, BC005147, BC001414, BC005831, BC006196, BC002975, BC000254, BC002614, BC001394, BC002427,

BC002455, BC000092, BC001431, BC000033, BC002976, BC002594, BC002536, BC003624, BC002977, BC001432, BC000740, BC001500, BC008062, BC001903, BC002433, BC002476, BC002546, BC001176, BC002978, BC001721, BC001177, BC001125, BC000741, BC001630, BC001619, BC000233, BC001501, BC000665, BC001502, BC000690, BC003112, BC001178, BC001459, BC000086, BC008063, BC005832, BC000545, BC001670, BC000562, BC001503, BC015558, BC001605, BC001388, BC001746, BC001741, BC000232, BC000080, BC001904, BC012799, BC000182, BC001504, BC001372, BC001606, BC007407, BC006465, BC000095, BC017455, BC002584, BC001023, BC001754, BC008749, BC008064, BC002601, BC001785, BC001786, BC015969, BC003113, BC001179, BC003575, BC000013, BC002557, BC001061, BC003625, BC016320, BC002576, BC007493, BC001643, BC001505, BC011606, BC001180, BC001034, BC006498, BC006540, BC001360, BC002523, BC002979, BC000742, BC004966, BC004147, BC006541, BC019260, BC000211, BC017174, BC004967, BC000261, BC001506, BC008933, BC002466, BC000693, BC003576, BC006499, BC001644, BC008065, BC006197, BC021985, BC018652, BC006542, BC008934, BC001062, BC006501, BC023985, BC004151, BC000745, BC019236, BC004153, BC004154, BC006543, BC001063, BC006544, BC006504, BC008937, BC000747, BC006505, BC004155, BC001064, BC013923, BC000214, BC004156, BC018654, BC008938, BC007408

SUPPLEMENTARY FILE 4: PATTERN OF INSERTION AND/OR DELETIONS TO OBTAIN THE COMPLETE DATASET WITH INDELS

Manipulation of the 981 sequences qualified as `sure Complete` from CGD-MGC (Table 1) to create 16 new datasets of 981 sequences each by introducing artificial insertions and/or deletions as depicted in the following figure:

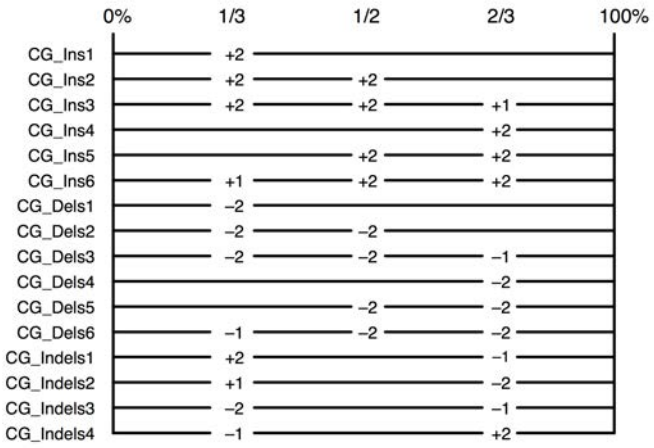


Fig. 3. Pattern of artificially-introduced insertions and deletions in every sequence of CGD-MGC dataset, generating the 16 datasets of 981 sequences each indicated at the left. Note that in no case is the initial reading frame recovered after the indel combination.

The obtained sequence datasets were the following:

- CG_Ins1 to CG_Ins6: datasets including insertions of 1-2 nt in the first third, middle and/or last third of the sequence, shifting the reading frame one or more times. In some cases, there are up to three insertion events per sequence.
- CG_Dels1 to CG_Dels6: datasets equivalent to the preceding ones, but with deletions of 1-2 nt instead of insertions.
- CG_Indels1 to CG_Indels4: datasets including combinations of insertions and deletions of 1-2 nt in the first third and last third of the sequence for shifting twice the reading frame.

SUPPLEMENTARY FILE 5: DETAILED DESCRIPTION OF OF MISCLASSIFICATIONS OF FULL-LENGTHERNEXT

In CGD-GB:

A total of 27 sequences in CGD-GB were misclassified (Table 1), 20 of them being `C-terminal`, 4 `N-Terminal`, 2 were simply `Coding` and 1 was `Unknown`. Four (BC111790, BC103905, BC080189, BC015371) of the 20 `C-terminal` sequences were misclassified because the top-BLASTx subject (the subject with the highest score and *E*-value) corresponds to a confounding reference of the protein (Suppl. Fig. 4A) because it has an *E*-value and/or score slightly higher than the right reference sequence. For example, BC111790 gives an *E*-value = 0.0 for Q5SW24 (long reference) and Q5SW24-2 (short reference), but their scores differ (1068 for Q5SW24 and 1061 for Q5SW24-2). BC111790 should be analysed using the shorter reference Q5SW24-2 because identity spans the complete length of both sequences. However, BC111790 is also identical to the long reference Q5SW24, but only downstream of the M corresponding to its initial M, while the sequence upstream of this M has no significant similarity (only 7 similarities in 21 positions). The lack of similarity among those 21 amino acids while the rest are identical is a remarkable behaviour that can be explained only if the long reference Q5SW24 is considered to contain a 21 N-terminal amino acid-long extra sequence. The spurious similarity in the extra region accounts for the higher score in Q5SW24 with respect to Q5SW24-2 (Suppl. Fig. 4A.1). Moreover, this dual behaviour of similarity also prompts to select reference Q5SW24-2, and not the reference Q5SW24, as this is the right orthologue of BC111790. Similarly, 2 (BC104884, BC063037) of the misclassified 4 `N-terminal` sequences did have an equivalent explanation because the top-BLASTx subject was a confounding reference longer than the queries at the 3' region (Suppl. Fig. 4A.2). The 6 sequences treated as cases of Suppl. Fig. 4A were mispredicted because the query sequence seems to be incomplete since it is shorter than a top-BLASTx confounding reference although, in fact, it contains the correct reference and should be classified as `Complete`.

Other cases of apparent misprediction correspond to the other 2 sequences as `N-terminal` (Table 1), where the query sequence contained a frame-shift that misled to a premature stop codon (Suppl. Fig. 4B.1); therefore, the query sequence did not code for a complete protein since the C-terminal part is absent. As a result, the FULL-LENGTHERNEXT status was right while the query sequence metadata of `complete cds` were wrong.

In 13 of the 20 cases where FULL-LENGTHERNEXT status was `C-terminal` (Table 1), the status was right because the query sequence does not contain the beginning of the gene: a frameshift mistakenly rises a false ATG as the start codon (Suppl. Fig. 4.B.2). Again, the FULL-LENGTHERNEXT was right while the sequence metadata were wrong.

Finally, 2 of the 3 sequences that did not retrieve any reliable similarity in UniProtKB protein databases, were identified as coding sequences, suggesting that this part of the algorithm is reliable. For the remaining mispredicted sequences, it is difficult to assert whether the status or the metadata were right, since they contain too many sequencing errors and several subjects reveal different classes of status.

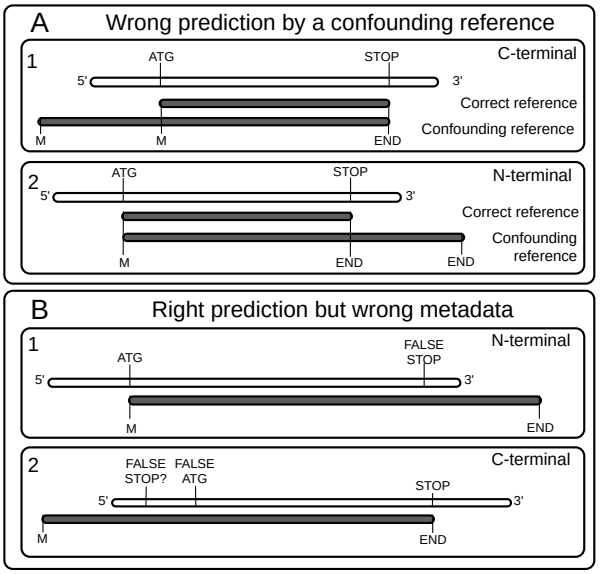


Fig. 4. Cases of complete sequences from CGD-GB classified as incomplete. The query (testing) sequence is in white and the reference sequence in database is in dark grey. A: cases where a true complete sequence was classified as (1) `C-terminal` because the query cannot contain an ATG located where required by the first M of the confounding reference, or (2) `N-terminal` because the query does not contain an stop codon at the position indicated by the confounding reference. B: cases where metadata of the query sequence indicated that it was complete but FULL-LENGTHERNEXT predicts that it is an incomplete protein. In some cases (1), the query sequence finishes in a premature stop codon due to a frame-shift. In other cases (2), the query sequence starts at an internal ATG, lacking the 5'-end of the gene; since it is also due to a frame-shift, it usually has an in-frame stop codon upstream of the false ATG.

In summary, the 27 seemingly misclassified sequences from CGD-GB cannot be considered FULL-LENGTHERNEXT failures since 15 of them were misannotated and the prediction of FULL-LENGTHERNEXT was the real status. Another 6 cases were analysed using a confounding reference and 2 out of the 3 sequences without UniProt orthologue were correctly classified as `Coding`.

In CGD-MGC:

In this dataset, only 9 sequences were apparently misclassified, all of them as `C-terminal` (Table 1). Two (BC008063 and BC023985) were truncated at the 5' end, their metadata therefore being wrong and the FULL-LENGTHERNEXT status being right (as in Fig. 4B.2). Another one (BC001863) seems to be a chimeric sequence because it shows similarity with two different subjects, one of which is the sequence BC020999, which has the description *WARNING: chimeric clone*. The remaining 6 sequences receive a wrong status due to the use of a confounding reference (as in Fig. 4A). It is interesting to note that of the 10 sequences having a `Putative complete` status (Table 1), 7 (BC008039, BC003106, BC008367, BC017174, BC001223, BC002429, BC002461) did not start at a methionine, suggesting that the 5'-end of the sequence may not contain the true ATG.

Capítulo 12

Prototipo para anotar secuencias genómicas incompletas

Mi contribución: Apoyo en la organización de la orientación a objetos del código, paralelización del análisis y desarrollo de la interfaz web.

Título del artículo: GENote v.β: A Web Tool Prototype for Annotation of Unfinished Sequences in Non-model Eukaryotes
Autores: Noé Fernández-Pozo, Darío Guerrero-Fernández, Rocío Bautista, Josefa Gómez-Maldonado, Concepción Avila, Francisco M. Cánovas, M. Gonzalo Claros
Publicación: Bioinformatics for Personalized Medicine - Lecture Notes in Computer Science. Springer Berlin Heidelberg
Volumen:6620
Año: 2012
DOI: http://dx.doi.org/10.1007/978-3-642-28062-7_7

Abstract De novo identification of genes in newly-sequenced eukaryotic genomes is based on sensors, which are not available in non-model organisms. Many annotation tools have been developed and most of them require sequence training, computer skills and accessibility to sufficient computational power. The main need of non-model organisms is finding genes, transposable elements, repetitions, etc., in reliable assemblies. GENote v.β is intended to cope with these aspects as a web tool for researchers without bioinformatics skills. It facilitates the annotation of new, unfinished sequences with descriptions, GO terms, EC numbers and KEEG pathways. It currently localises genes and transposons, which enable the sorting of contigs or scaffolds from a BAC clone, and reveals some putative assembly inconsistencies. Results are provided in GFF3 format and in tab-delimited text readable in viewers; a summary of findings is provided also as a PNG file.

Parte V

DIFUSIÓN DE RESULTADOS MEDIANTE BASES DE DATOS

Capítulo 13

Sistema de máquinas virtuales para bases de datos de transcriptómica

13.1. Una base de datos simple para transcriptómica

La presentación de una base de datos con los datos relevantes sobre un proyecto de secuenciación es una necesidad hoy en día en los proyectos de investigación. Afortunadamente, la tecnología para conectar las páginas web con las bases de datos y ofrecer la información a través de Internet se está convirtiendo casi en un estándar, lo que está permitiendo la proliferación de bases de datos públicas que apoyan la investigación sobre organismos modelo [215], farmacogenómica [241], humanos [242], estudios genómicos sobre el cáncer [237] o metagenómica de los microorganismos [211], entre otros.

Por eso hemos desarrollado un sistema de presentación de datos que permite el almacenamiento genérico de experimentos de secuenciación de transcriptomas.

En la primera base de datos que desarrollamos utilizamos Ruby-on-Rails [74], que en el año 2007 ya llevaba un tiempo establecido como una herramienta de alta productividad para el desarrollo de aplicaciones web. Al ser la primera, siempre ha ido marcando el camino al resto de alternativas que comenzaban a aparecer, como Django[78] para Python, o CakePHP [26] basado en PHP. Una de sus ventajas es que permiten una organización de código excelente, además de proporcionar una capa de abstracción de la base de datos. De esta forma pudimos empezar el desarrollo conectando directamente con una base de datos Oracle que ya teníamos en funcionamiento, y posteriormente realizar una migración a MySQL con mínimas modificaciones de código fuente. En ella se almacenaron, además de las secuencias anotadas del transcriptoma de pino, información de gestión del material biológico, como las placas de cultivo en las que están los clones, dónde se sitúa en el congelador, si ese clon aparece o no en la primera micromatriz de pino que se diseñó en el grupo de investigación del catedrático Francisco Cánovas, etc.

Mi contribución se centró en el diseño de las tablas que contendrían los datos (figura 13.1) , en la elección del lenguaje más apropiado, y el portal web para que se integre correctamente con la base de datos y con la infraestructura del Centro de Supercomputación y Bioinformática (Centro de Supercomputación y Bioinformática de la UMA (SCBI)) de la Universidad de Málaga. La base de datos resultante se describe en el artículo que viene a continuación.

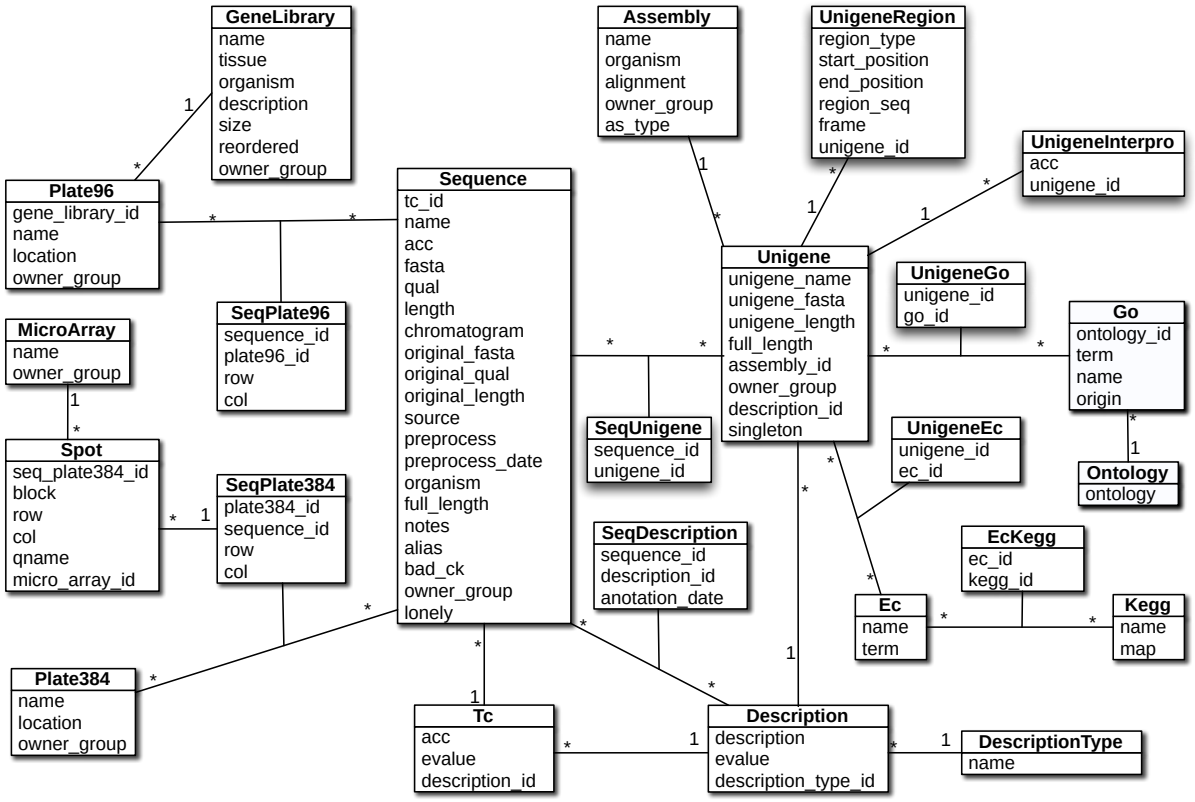


Figura 13.1: *Esquema inicial de la base de datos de transcriptómica usado en EuroPineDB.*

Título del artículo: EuroPineDB: a high-coverage web database for maritime pine transcriptome
Autores: Fernandez Pozo, Noe; Canales, Javier; Guerrero Fernandez, Daro; Villalobos, David P.; Diaz Moreno, Sara M.; Bautista, Rocio; Flores Monterroso, Arantxa; Guevara Morato, Ángeles; Perdiguero Jimenez, Pedro; Collada Collada, Maria Carmen; Cervera Goy, María Teresa; Soto de Viana, Álvaro; Ordas, Ricardo; Canton, Francisco R.; Avila, Concepcion; Canovas, Francisco M. y Gonzalo Claros, M.
Publicación: BMC Genomics
Volumen:12 p 366
Año: 2011
DOI: `http://dx.doi.org/10.1186/1471-2164-12-366`

Abstract

Background Pinus pinaster is an economically and ecologically important species that is becoming a woody gymnosperm model. Its enormous genome size makes whole-genome sequencing approaches are hard to apply. Therefore, the expressed portion of the genome has to be characterised and the results and annotations have to be stored in dedicated databases.

Description EuroPineDB is the largest sequence collection available for a single pine species, Pinus pinaster (maritime pine), since it comprises 951 641 raw sequence reads obtained from non-normalised cDNA libraries and high-throughput sequencing from adult (xylem, phloem, roots, stem, needles, cones, strobili) and embryonic (germinated embryos, buds, callus) maritime pine tissues. Using open-source tools, sequences were optimally pre-processed, assembled, and extensively annotated (GO, EC and KEGG terms, descriptions, SNPs, SSRs, ORFs and InterPro codes). As a result, a 10.5x P. pinaster genome was covered and assembled in 55 322 UniGenes. A total of 32 919 (59.5 %) of P. pinaster UniGenes were annotated with at least one description, revealing at least 18 466 different genes. The complete database, which is designed to be scalable, maintainable, and expandable, is freely available at: `www.scbi.uma.es/pindb/`. It can be retrieved by gene libraries, pine species, annotations, UniGenes and microarrays (i.e., the sequences are distributed in two-colour microarrays; this is the only conifer database that provides this information) and will be periodically updated. Small assemblies can be viewed using a dedicated visualisation tool that connects them with SNPs. Any sequence or annotation set shown on-screen can be downloaded. Retrieval mechanisms for sequences and gene annotations are provided.

Conclusions The EuroPineDB with its integrated information can be used to reveal new knowledge, offers an easy-to-use collection of information to directly support experimental work (including microarray hybridisation), and provides deeper knowledge on the maritime pine transcriptome.

13.2. División de la base de datos en máquinas virtuales

La base de datos EuroPineDB estaba diseñada más para navegar por ella que para interrogarla, aunque llevaba una búsqueda de secuencias muy básica con texto y con BLAST+ (para las secuencias de nucleótidos). En cuanto se empezó a usar de forma intensiva se observó que su arquitectura, que agrupaba el servidor web y las búsquedas con BLAST+ en una misma máquina, no era eficiente y podía dar problemas de saturación con facilidad y desesperar de esta forma al usuario. Esto nos llevó cambiar completamente la forma de construir la base de datos y a pensar en que podíamos sacar provecho de los servidores de máquinas virtuales del SCBI. El sistema ideado está basado en tres máquinas virtuales que cualquier laboratorio puede alojar para poner a disposición del gran público o de los investigadores del propio grupo los resultados que han obtenido.

La división en máquinas virtuales diferentes proporciona diversas mejoras respecto al clásico modelo todo en uno, sobre todo en cuanto al rendimiento y en la contención de los posibles problemas de saturación de los recursos. Cada componente del sistema que exige un uso intensivo de la CPU está aislado en una máquina virtual independiente, de modo que no puede interferir en el funcionamiento normal del resto del sistema. Por ejemplo, el servidor web que atiende las peticiones de los usuarios no se verá ralentizado cuando otro usuario esté realizando una consulta a la base de datos, o realice una búsqueda que lance un programa de procesamiento de fondo como puede ser una búsqueda BLAST+ que tarda bastante en completarse.

La arquitectura básica constará de tres máquinas virtuales (figura 13.2): una para el servidor web, otra para alojar y manejar la base de datos, y otra para realizar todo tipo de cálculos (principalmente los BLAST+). La comunicación entre las máquinas se realiza mediante una red privada que, como tal, se considerará muy segura al no poderse ver desde el exterior. De hecho, la única máquina visible desde el exterior es el servidor web, que dispondrá de dos interfaces ethernet virtuales, uno para la red privada interna, y otro para la red exterior por la que da servicios a los usuarios. En las máquinas virtuales se ha implementado un sistema de colas para ejecutar con eficacia las peticiones de búsqueda con BLAST+ de los usuarios. Así, si la máquina destinada a cálculo dispone de 2 CPU, se le permite ejecutar a la vez 2 trabajos de 1 CPU cada uno, pero nunca más de dos. Cuando termina uno, se le notifica al usuario y arranca el siguiente trabajo que podría ser de ese mismo usuario o de otro distinto, pero siempre manteniendo una cola First In First Out (FIFO).

Veamos con más detalle cada una de las máquinas.

13.2.1. Servidor web

Esta máquina virtual se encargará de proporcionar el acceso desde el exterior a través de un portal web realizado con Ruby-on-Rails (RoR). El servidor web accede a los datos almacenados en el servidor de bases de datos a través de la red privada para satisfacer las consultas de los usuarios, que van inspeccionando utilizando cualquier navegador web los diferentes ensamblajes y sus respectivas anotaciones.

Los usuarios pueden además solicitar la ejecución de búsquedas por comparación de secuencias de entrada contra diversos ficheros de datos existentes basados en los ensamblajes almacenados en la base de datos. Estos trabajos

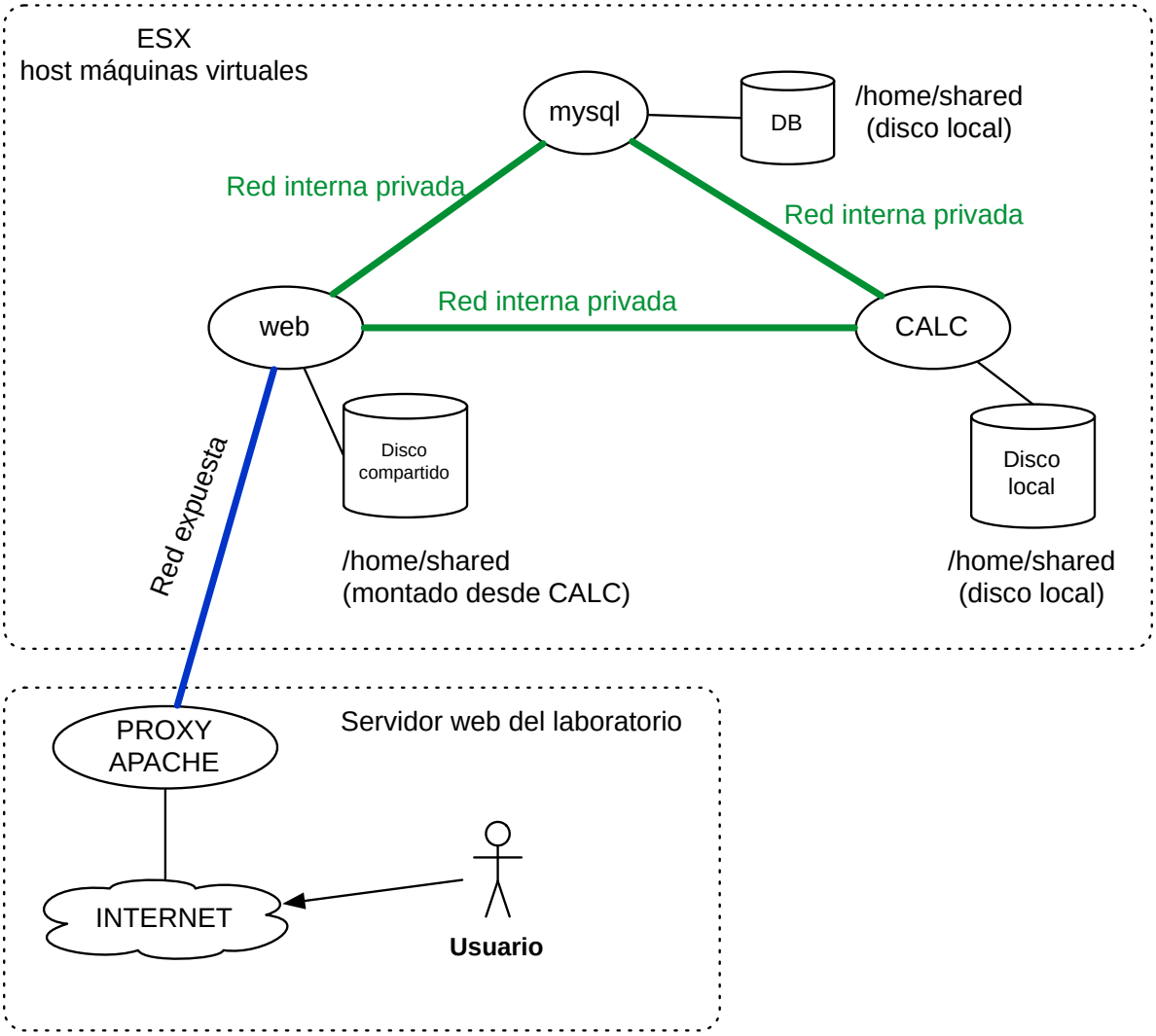


Figura 13.2: *Esquema genérico de las 3 máquinas virtuales necesarias para ofrecer una base de datos de transcriptómica con un rendimiento eficiente*

de comparaciones se realizarán con BLAST+ y se enviarán a ejecutar en el servidor de cálculo con la finalidad de liberar al servidor web de la carga de trabajo que esto supondría, y que en otro caso podría provocar interrupciones en el servicio. En caso de que se disponga de recursos de computación adicionales, esta carga podría derivarse al sistema de colas que lo gestione.

El servidor web lleva instalados los siguientes paquetes/servicios:

- Sistema operativo Linux (OpenSuse 11.4)
- Apache 2.2.17
- Ruby on Rails v2
- Passenger 3.0.7 (conecta RoR con Apa-

che)

- Aplicación para RoR que proporciona la interfaz web para la base de datos de transcriptómica.

13.2.2. Servidor de base de datos

Esta máquina virtual aloja una base de datos en MySQL con los datos que se hayan decidido publicar sobre el proyecto. Se pueden incluir diferentes versiones de un mismo ensamblaje, ensamblajes parciales de un ensamblaje global, o ensamblajes diferentes con datos diferentes, todo ellos con sus respectivas anotaciones. De esta forma se disminuye al mínimo la carga de trabajo del servidor web.

Llevará instalados los siguientes paquetes/servicios:

- Sistema operativo Linux (OpenSuse 11.4)
- MySQL Comunity 5.1.53

13.2.3. Servidor de cálculo

Esta máquina virtual se encargará de realizar las ejecuciones de BLAST+ (o cualquier otro algoritmo pesado que se quiera conectar a la base de datos) contra los ficheros de datos del proyecto. Para evitar el colapso del servidor web, todos los cálculos son enviados a este servidor adicional que ejecuta los trabajos por orden de entrada. Para ello cuenta hemos desarrollado un sistema de colas muy rudimentario sqs_queue_system. Una vez ejecutado el trabajo, los resultados se almacena en una parte del disco compartido que está disponible para el servidor web, y el usuario podrá acceder a los mismos a través del mismo portal.

El servidor de bases de datos lleva instalados los siguientes paquetes/servicios:

Servidor	CPUS/ cores	RAM	Almacenamiento
Web	2 cores	4 GB	30 GB
Base de datos	2 cores	4 GB	60 GB
Cálculo	4 cores	8 GB	60 GB
TOTAL	**8 cores**	**16 GB**	**150 GB**

Figura 13.3: *Recursos estimados para una base de datos de transcriptómica de tamaño medio*

- Sistema operativo Linux (OpenSuse 11.4)
- Paquete de software BLAST+ 2.2.25 [37]
- Software propio para gestionar/encolar la carga de trabajo
- Software para compartir los trabajos con el servidor web (NFS 2.19.3)

13.2.4. Infraestructura de soporte

Las máquinas virtuales se han desarrollado sobre un servidor de virtualización ESXi (gratuito) o ESX (con licencia de pago) de VMware, aunque en caso necesario se pueden traducir a otros hipervisores distintos.

En la figura 13.3 se resumen los recursos exigidos para cada máquina virtual. Estos recursos están asignados con cierto margen, es decir, son recursos suficientes para alojar bases de datos de transcriptómica de un organismo complejo con varias versiones de ensamblajes y anotaciones. No obstante, si la cantidad de usuarios sube de tal modo que un sólo servidor no pueda gestionarla, al tratarse de máquinas virtuales se puede incrementar los recursos asignados sin complejidad, tanto por parte de la web, como las bases de datos y el procesamiento.

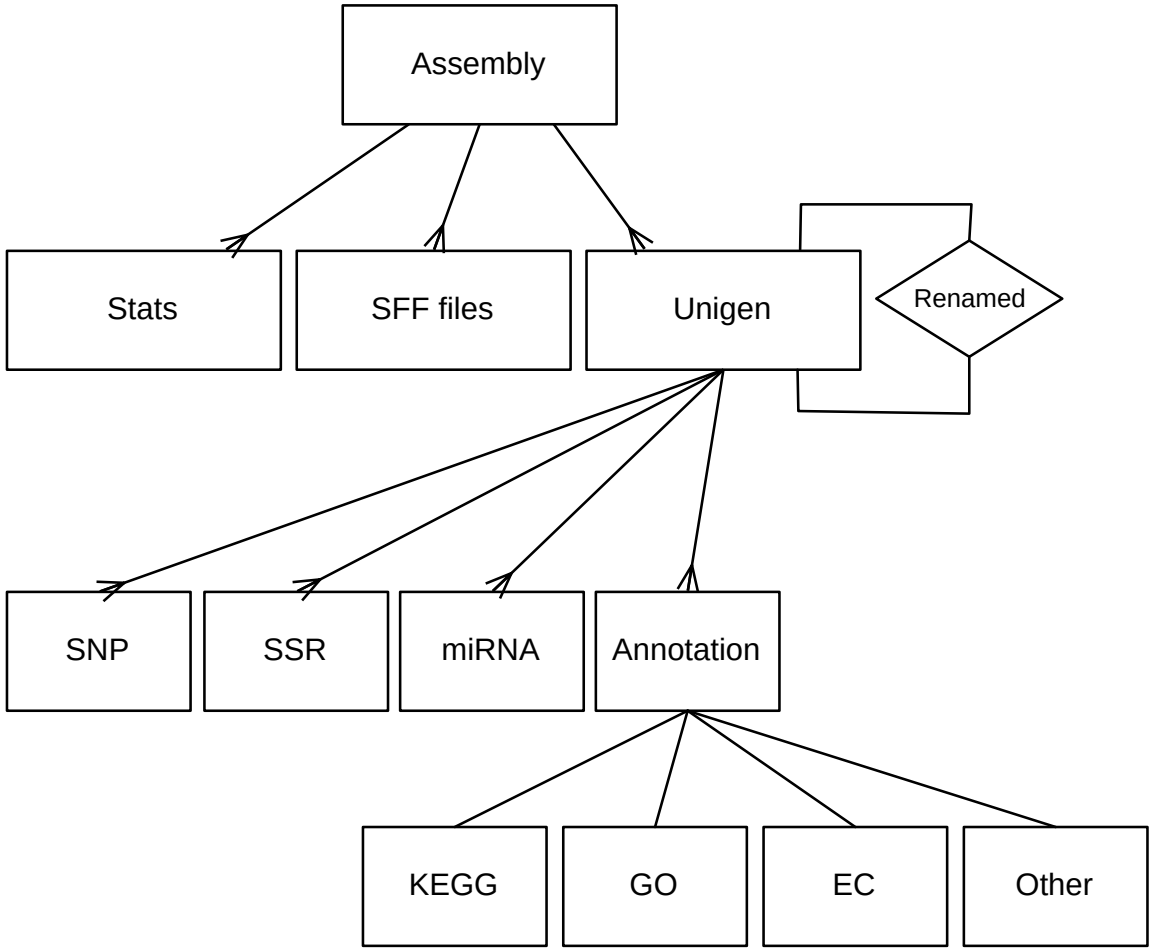


Figura 13.4: *Modelo de datos simplificado para las siguientes aplicaciones de la base de datos de transcriptómica*

13.3. Mejora del modelo de datos

En la figura 13.4 podemos observar el modelo de datos simplificado que se utiliza para la base de datos de transcriptómica. Según este modelo de datos, para cada ensamblaje podemos almacenar los archivos de las lecturas originales, así como los unigenes pertenecientes al mismo. Los archivos de las secuencias originales, tanto en FASTQ como en SFF se almacenan fuera de la base de datos en un fichero que sigue una técnica de indexación similar a la utilizada en FQBin (capítulo 8) ajeno a la base de datos, pero son accesibles desde la interfaz web. Las estadísticas sobre el contenido del ensamblaje se han calculado al principio, al ser muy costosas desde el punto de vista computacional, y ser siempre igual para cada ensamblaje; por tanto, guardarlas «desperdicia» espacio pero «ahorra mucho» tiempo.

Para cada transcrito (denominado *unigen* en las figuras 13.1 y 13.4) podremos almacenar conjuntos de SNP, repeticiones de secuencias simples (SSR) o cadenas de miRNA (microRNA, o sea, cadenas pequeñas de ARN que influyen en la expresión de los genes), junto con otras anotaciones como mapas KEGG, códigos EC (referentes a su actividad enzimática), términos GO, claves de InterPro, o una descripción textual. Para cada una de las características que se almacenen del transcrito se mantendrá determinada información adicional, como son la zona del transcrito al que se refieren, niveles de calidad de la anotación o programa que la ha generado.

Se observó que las tablas que definían la base de datos EuroPineDB contenían información redundante que se podía simplificar. Además, se consideró que la información de infraestructura de laboratorio, como las placas y posiciones en micromatrices, ya no merecía la pena mantenerla, y además había que dar soporte a las nuevas tecnologías de ultrasecuenciación que se estaban imponiendo para estudiar transcriptomas. Todo ello nos hizo replantear el esquema de la base de datos, y proponer uno mucho más flexible donde la principal mejora se centra en simplificar el manejo de los distintos tipos de anotaciones. Vimos que había que preparar la base de datos para facilitar la adición de nuevos tipos de anotaciones, por lo que todas las anotaciones que antes se almacenaban en tablas diferentes (`EC`, `GO`, `SNPs`, `KEGG`, etc.), ahora se almacenan en una tabla genérica de anotaciones, donde cada anotación está etiquetada con su tipo (`EC`, `GO`, `SNPs`, `KEGG`). Este cambio permite su búsqueda por separado y añadir nuevos tipos de anotaciones simplemente con añadir un nuevo tipo de etiqueta, sin necesidad de modificar el esquema de la base de datos, ni las relaciones entre sus entidades, ni la interfaz web. EuroPineDB se empezó a desarrollar aprovechando un *cluster* de servidores Oracle RAC 10g [224] en alta disponibilidad que se estaba usando para otros productos. En ese momento pareció la estrategia más adecuada, porque tener una base de datos tan potente al alcance empujaba a aprovecharla para alojar los datos de la base de datos de transcriptómica. La estrategia se mostró errónea cuando empezamos a usar esta base de datos para que se expusieran en servidores de otros centros de investigación, que muy probablemente no tuvieran intención de adquirir una licencia de Oracle. Por tanto, decidimos cambiar a usar una base de datos de uso público. MySQL [160] cumplía todos los requisitos y fue la elegida, aunque igualmente podríamos haber elegido PostgreSQL [207]. Gracias a que la aplicación web esta realizada en RoR que utiliza una abstracción de persistencia de objetos en bases de datos llamada ActiveRecord [105], la migración de Oracle a MySQL fue transparente, y sin necesidad de tocar código fuente más que para cambiar el conector de la base de datos.

La siguiente mejora consistió en un ajuste de la configuración de MySQL (`/etc/my.cnf`) para optimizar el rendimiento de las búsquedas y el acceso en lectura, al ser el que más se realiza en este tipo de aplicaciones mediante las siguientes modificaciones:

```
# Crea un archivo temporal por cada
    tabla y permite su purgado. Si no se
     utiliza esta opción, todas las
    tablas usan el mismo archivo
    temporal y se impide el purgado del
    mismo, por lo que éste no deja de
    crecer.
innodb_file_per_table

# Anota las consultas que son muy
    lentas para permitirnos optimizar el
     código SQL que realiza la consulta,
     añadir índices o modificar la
    estrategia de búsqueda
log-slow-queries
```

```
# Se amplía el buffer dedicado de 16M a
    128M a los identificadores de los
    registros, esto implica menos
    accesos a disco.

key_buffer_size = 128M

# Amplía el buffer para la lista de
    tablas abiertas
table_open_cache = 256

# Amplía el buffer de ordenación de 512
    K a 4MB, esto acelera las consultas
    en las que se pide que se ordenen
    los resultados
sort_buffer_size = 4M

# Amplía el buffer de lectura de 256K a
    16M, esto acelera las consultas
    evitando accesos a disco al tener más
    cantidad de datos en RAM
read_buffer_size = 16M

# Permitir al proceso MySQL utilizar
    hasta un 80\% de la memoria
    disponible en la máquina para
    acelerar su funcionamiento. Esto
    podemos permitirnoslo porque la
    máquina MySQL se encuentra instalada
    en una máquina virtual independiente
    donde no convive con otros procesos.
    En caso de estar en un servidor
    compartido debemos ser más
    conservadores con estos dos parámetros.
innodb_buffer_pool_size = 1024M
innodb_additional_mem_pool_size = 256M
```

13.4. Importación de los datos

Se ha propuesto un sistema desacoplado, es decir, los ensamblajes y anotaciones se realizan con los recursos de computación o supercomputación disponibles, y los ficheros resultantes se suben a la máquina virtual del servidor web donde se importan con un *script* que lee los archivos, los interpreta y va insertando los datos en sus respectivas tablas. Una vez finalizada la importación, los archivos originales de los ensamblajes también se quedan almacenados con el objetivo de poder realizar comparaciones con BLAST+ desde el portal web.

13.5. Transcriptoma de pino marítimo y de lenguado

La nueva estructura de la base de datos se ha utilizado para mostrar y difundir el transcriptoma de dos organismos: el pino marítimo (*Pinus pinaster*) en la evolución de EuroPineDB denominada SustainPineDB, y dos especies de lenguado, el común (*Solea solea*) y el senegalés (*Solea senegalensis*), en la base de datos SoleaDB. Mi contribución consistió en (1) facilitar la ejecución de los flujos de trabajo largos que se usaron (figuras 1 y 2 en el artículo de SustainPineDB, y figura 2 en el artículo de SoleaDB); (2) el diseño y construcción de la base de datos, (3) permitir que la importación de datos a las tablas sea lo más simple posible, y finalmente (4) su administración y gestión de las migraciones, puesto que SustainPine está alojada en el SCBI, pero las máquinas virtuales de SoleaDB se generan en el SCBI y se envían al IFAPA El Toruño (Cádiz) donde le dan visibilidad en la Uniform Resource Locator (URL) `http://www.juntadeandalucia.es/agriculturaypesca/ifapa/soleadb_ifapa/`. Además, quiero señalar que en ambas bases de datos se han utilizado las herramientas SeqTrimNext (Capítulo 10) y Full-LengtherNext (Capítulo 11) descritas en esta memoria.

Título del artículo: De novo assembly of maritime pine transcriptome: implications for forest breeding and biotechnology
Autores: Canales, Javier and Bautista, Rocio and Label, Philippe and Gómez-Maldonado, Josefa and Lesur, Isabelle and Fernández-Pozo, Noe and Rueda-López, Marina and Guerrero-Fernández, Dario and Castro-Rodríguez, Vanessa and Benzekri, Hicham and Cañas, Rafael A. and Guevara, María-Angeles and Rodrigues, Andreia and Seoane, Pedro and Teyssier, Caroline and Morel, Alexandre and Ehrenmann, François and Le Provost, Grégoire and Lalanne, Céline and Noirot, Céline and Klopp, Christophe and Reymond, Isabelle and García-Gutiérrez, Angel and Trontin, Jean-François and Lelu-Walter, Marie-Anne and Miguel, Celia and Cervera, María Teresa and Cantón, Francisco R. and Plomion, Christophe and Harvengt, Luc and Avila, Concepción and Gonzalo Claros, M. and Cánovas, Francisco M.
Publicación: Plant Biotechnology Journal
Volumen:12:3 - p286-p299
Año: 2014
DOI: http://doi.org/10.1111/pbi.12136

Abstract

Summary Maritime pine (Pinus pinasterAit.) is a widely distributed conifer species in Southwestern Europe and one of the most advanced models for conifer research. In the current work, comprehensive characterization of the maritime pine transcriptome was performed using a combination of two different next-generation sequencing platforms, 454 and Illumina. De novo assembly of the transcriptome provided a catalogue of 26 020 unique transcripts in maritime pine trees and a collection of 9641 full-length cDNAs. Quality of the transcriptome assembly was validated by RT-PCR amplification of selected transcripts for structural and regulatory genes. Transcription factors and enzyme-encoding transcripts were annotated. Furthermore, the available sequencing data permitted the identification of polymorphisms and the establishment of robust single nucleotide polymorphism (SNP) and simple-sequence repeat (SSR) databases for genotyping applications and integration of translational genomics in maritime pine breeding programmes. All our data are freely available at SustainpineDB, the P. pinaster expressional database. Results reported here on the maritime pine transcriptome represent a valuable resource for future basic and applied studies on this ecological and economically important pine species.

Título del artículo: De novo assembly, characterization and functional annotation of Senegalese sole (Solea senegalensis) and common sole (Solea solea) transcriptomes: integration in a database and design of a microarray
Autores: Benzekri, H., Armesto, P., Cousin, X., Rovira, M., Crespo, D., Merlo, M., Merlo ,D., Mazurais, R., Bautista, D., Guerrero-Fernández,... Manchado, M.
Publicación: BMC Genomics
Volumen:15:952
Año: 2014
DOI: http://doi.org/10.1186/1471-2164-15-952

Abstract

Background Senegalese sole (Solea senegalensis) and common sole (S. solea) are two economically and evolutionary important flatfish species both in fisheries and aquaculture. Although some genomic resources and tools were recently described in these species, further sequencing efforts are required to establish a complete transcriptome, and to identify new molecular markers. Moreover, the comparative analysis of transcriptomes will be useful to understand flatfish evolution.

Results A comprehensive characterization of the transcriptome for each species was carried out using a large set of Illumina data (more than 1,800 millions reads for each sole species) and 454 reads (more than 5 millions reads only in S. senegalensis), providing coverages ranging from 1,384x to 2,543x. After a de novo assembly, 45,063 and 38,402 different transcripts were obtained, comprising 18,738 and 22,683 full-length cDNAs in S. senegalensis and S. solea, respectively. A reference transcriptome with the longest unique transcripts and putative non-redundant new transcripts was established for each species. A subset of 11,953 reference transcripts was qualified as highly reliable orthologs (¿97

Conclusions Transcriptomes and molecular markers identified in this study represent a valuable source for future genomic studies in these economically important species. Orthology analysis provided new clues regarding sole genome evolution indicating a divergent evolution of crystallins in flatfish. The design of a microarray and establishment of a reference transcriptome will be useful for large-scale gene expression studies. Moreover, the integration of transcriptomic data in the SoleaDB will facilitate the management of genomic information in these important species.

13.6. Extensión semántica y preparación para datos de expresión

Las nuevas demandas de la transcriptómica ha hecho que hayamos tenido que hacer evolucionar la base de datos para:

- Incluir una cuarta máquina virtual para permitir búsquedas semánticas de la web basadas en Linked Data [28] gracias a una colaboración con el grupo de investigación del catedrático José Aldana Montes y el profesor Ismael Navas Delgado [182, 150].
- Añadir tablas para almacenar datos de expresión génica con RNA-Seq [228].

Con estas nuevas características se ha enviado a publicar un artículo donde describimos la base de datos ReprOlive que contiene los transcriptomas del tejido reproductivo (polen y pistilo) de olivo y próximamente incluirá también el transcriptoma semillero. Se adjunta el manuscrito de dicho artículo.

Frontiers in Journal

Original Research
2015-04-21

ReprOlive: a Database with Linked-Data for the Olive Tree (*Olea europaea* L.) Reproductive Transcriptome

ReprOlive: an olive tree reproductive transcriptome database

Rosario Carmona[1,2,§], A. Zafra[1,§], Pedro Seoane[3], A. Castro[1], Darío Guerrero-Fernández[2], Trinidad Castillo-Castillo[4], Ana Medina-García[4], Francisco M. Cánovas[3], José F. Aldana-Montes[4], Ismael Navas-Delgado[4], Juan D. Alché[1], M. Gonzalo Claros[2,3,*]

[1] *Department of Biochemistry, Cell and Molecular Biology of Plants. Estación Experimental del Zaidín. CSIC. Granada. Spain.*
[2] *Plataforma Andaluza de Bioinformática, Edificio de Bioinnovación, Universidad de Málaga. Spain*
[3] *Departamento de Biología Molecular y Bioquímica, Universidad de Málaga. Málaga. Spain*
[4] *Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga. Spain.*
[§] *These authors contributed equally to this work*

*** Correspondence:** *M. Gonzalo Claros, Departamento de Biología Molecular y Bioquímica, Facultad de Ciencias, Universidad de Málaga. 29071 Málaga. Spain. E-mail:* claros@uma.es

Keywords: Olive, transcriptome, reproduction, database, pollen, stigma, linked-data, annotation, assembly.

Abstract

Plant reproductive transcriptomes have been analysed in different species due to the presence of specific transcripts and the agronomical and biotechnological importance of plant reproduction. Here we presented an olive tree reproductive transcriptome database with samples from pollen and stigma at different developmental stages, and leaf and root as control vegetative tissues (http://reprolive.eez.csic.es). It was developed from 2,077,309 raw reads and 1,549 Sanger sequences. Using a pre-defined workflow based on open-source tools, sequences were pre-processed, assembled, mapped and annotated with expression data, descriptions, GO terms, InterPro signatures, EC numbers, KEGG pathways, ORFs, and SSRs. Tentative transcripts were also annotated with the corresponding orthologues in *Arabidopsis thaliana* from TAIR and RefSeq databases to enable Linked-Data integration. It results in a reproductive transcriptome comprising 72,846 contigs with average length of 686 bp, of which 63,965 (87.8%) included at least one functional annotation, and 55,356 (75.9%) had an orthologue. A minimum of 23,568 different tentative transcripts was identified and 5,835 of them contain a complete ORF. The representative reproductive transcriptome can be reduced to 28,972 tentative transcripts for further gene expression studies. Partial transcriptomes from pollen, stigma and vegetative tissues as control were also constructed. ReprOlive provides free access and download capability to these results. Retrieval mechanisms for sequences and transcript annotations are provided. Graphical localisation of annotated enzymes into KEGG pathways is also possible. Finally, ReprOlive has included a semantic conceptualisation by means of a Resource Description Framework (RDF) allowing Linked Data search for extracting the most updated information related to enzymes, interactions, allergens, structures and reactive oxygen species.

1. Introduction

Research in plant reproduction is accelerating rapidly as a direct consequence of the technological progresses (Dickinson and Franklin-Tong, 2011). Differential screening was initially used to identify abundant or specific transcripts of very specialised cells (Engel et al., 2003) and have been progressively replaced by the use of commercial microarrays and RNA-sequencing platforms. The use of transcriptomic approaches is unravelling the particular functionality of the key subsets of cells in charge of male and female gamete formation, and the complex interactions and signalling networks involved in the pollen-pistil interaction [reviewed in (Dukowic-Schulze and Chen, 2014)]. In good agreement with the degree of difficulty in isolating reproductive cells and extracting their RNAs, most studies up to date have reported gene expression in whole anther tissues, followed by male meiocytes, and female structures and meiocytes, being *Arabidopsis thaliana* the most widely studied, followed by lily, tobacco, brassica, petunia, maize, cotton and rice (Dukowic-Schulze and Chen, 2014). Those studies had shown that reproductive transcriptomes are substantially different from their vegetative counterparts, in concordance with the high proportion of specific transcripts present in these tissues, sometimes over 1,000 genes (Kamalay and Goldberg, 1980). Moreover, huge differences in terms of temporal expression are present among developmental stages (e.g. meiosis initiation, mature pollen, pollen germination) and spatial/tissue localisation (e.g. sporogenous tissue, tapetum, isolated pollen grains, pollen tube…), the most striking changes occurring in the mature pollen upon hydration and germination (Wang et al., 2008;Wei et al., 2010). The peculiarity of reproductive tissues in terms of gene expression also deserves a dedicated study not only for agronomical, biological and biotechnological reasons but also in seek of putative new allergens in pollen (El Kelish et al., 2014;Villalba et al., 2014).

Olive tree (*Olea europaea* L.) is one of the most important oil-producing plant species all over the world. While waiting for the genome sequence (Muleo et al., 2012), transcriptomic approaches have been exploited. For example, subtractive libraries from olive fruits sampled at three different stages shed light on metabolic pathways and transcriptional aspects related to carbohydrates, fatty acids, secondary metabolites, transcription factors and hormones as well as response to biotic and abiotic stresses throughout olive drupe development (Galla et al., 2009). Comparative 454 pyrosequencing from two olive genotypes during fruit development provided information about the structure and putative function of gene transcripts accumulated during fruit development, reporting differentially expressed genes with potential relevance in regulating the fruit metabolism and phenolic content during ripening (Alagna et al., 2009). ESTs were generated from two cDNA libraries from young olive leaves and immature olive fruits (Ozgenturk et al., 2010), which serve as a valuable source for further functional studies. Sanger sequencing and further microarray analysis identified differentially expressed transcripts in salt–tolerant and salt–sensitive olive cultivars (Bazakos et al., 2012). The olive abscission zone during cell separation in order to understand mature fruit abscission control was also studied by high-throughput sequencing (Gil-Amado and Gomez-Jimenez, 2013) to help in current olive breeding programmes. More recently, 12 cDNA libraries from olive fruit, seeds, young stems, leaves, buds and roots were sequenced, assembled and annotated (Muñoz-Merida et al., 2013).

A number of questions involving olive reproductive biology are still open. They include the search of explanations and the definition of criteria for potential improvements of the plant as regard to the selection of genotypes, the culture conditions to prevent alternate bearing, the extended juvenility of the plant (particularly in some cultivars), and the presence of self-incompatible genotypes. Knowledge about the pollen-pistil interactions in this plant is still scarce, and molecular evidence of the presence of self-incompatibility mechanisms (although largely suspected of the gametophytic type), is also limited in spite of the most recent transcriptomic analyses (Barcaccia et al., 2012;Collani et al., 2012;Turktas et al., 2013). Hence, this study extracted RNAs from pollen and

stigma in different maturing and developing stages to provide a reproductive transcriptome of olive tree and a user-friendly database containing the resulting information. Database queries may help scientists to develop further research and to design strategies to improve both yield and quality in these agronomic fields. Moreover, new clinical approaches are also expected to derive from the increased knowledge about the putative allergens present in the olive pollen.

2. Materials and methods

2.1. Sequence Processing

2.1.1. RNA Sources and Sequencing

With the aim of gathering most representative sequences from olive reproductive tissues, the 12 gene libraries described in Table 1 were constructed. RNAs and mRNAs from mature pollen grains, *in vitro* germinated pollen at 2 different times (1 h and 5 h), and stigmas at developmental stages 2, 3 and 4 (as defined by (Zafra et al., 2010)) were isolated using RNeasy Plant and Oligotex PoliA+ kits (Quiagen), respectively. cDNA libraries to be sequenced with a Roche GS-FLX Titanium+ were generated using the cDNA Synthesis System Kit (Roche). As a representation of olive vegetative transcriptome for control purposes, four gene libraries from olive leaves, roots and radicles were constructed. The three subtractive libraries (named with «Subs» in Table 1) resulting from the comparison of mature pollen, stigmas at developmental stage 4 and leaves (unpublished) were sequenced by the classical Sanger method.

2.1.2. Sequence Pre-processing and Assembling

Raw reads were pre-processed and assembled following the same pipeline as previously described by our laboratory (Benzekri et al., 2014;Canales et al., 2014) and illustrated as a flow diagram in Supplementary Data file 1. Briefly, pre-processing was based on SeqTrimNext [(Falgueras et al., 2010); http://www.scbi.uma.es/seqtrimnext] to remove low quality, ambiguous and low complexity stretches, linkers, adaptors, vector fragments, organelle DNA, polyA/polyT tails, and contaminated sequences while keeping the longest informative part of the read. Pyrosequences below 40 bp and Sanger sequences below 100 bp were also discarded. Useful reads (Tables 1 and 2) were assembled with an overlap-layout-consensus algorithm such as MIRA3, and a strict de Bruijn graph analysed by a Eulerian path such as Euler-SR. The contigs obtained (Table 2) were reconciled with CAP3 at 85% similarity to provide a final set or tentative transcripts (TTs) having consensus sequences closer to real transcripts (Liang et al., 2000;Fernandez-Pozo et al., 2011). The fact that reproductive transcriptome consists of 72,846 TT and the stigma transcriptome 60,400 (Table 2), was clearly overestimating the number of genes of these tissues. This overestimation may come from the presence of alleles, paralogues, fragmented sequences, alternative splicing, and even a combination of them. Therefore, further refinement will be required based on annotation before their inclusion in a database (see below).

2.1.3. Annotation

Functional classification of a list of interesting genes is absolutely required for future comparative studies. Reliable annotations were generated by combining separate information sources. Therefore, gene descriptions, GO terms, EC numbers, and InterPro signatures were provided by Sma3s (Muñoz-Mérida et al., 2014) using the non-redundant plant division of UniProtKB in order to remove spurious annotations. KEGG maps were retrieved directly from the KEGG site using the obtained ECs. Another gene description, putative start and stop codons, predicted amino-acid sequence, ORF status (full-length or incomplete coded proteins), putative ncRNAs, *Arabidopsis thaliana* orthologue from TAIR10 (Lamesch et al., 2012) and RefSeq (Pruitt et al., 2012) (as is in Nov 2012), protein coding status based on TransDecoder (http://transdecoder.github.io), and the reference TT were provided by Full-LengtherNext (P. Seoane et al., in preparation; http://www.scbi.uma.es/fulllengthernext). Microsatellites, as a source of genetic markers, were obtained by screening for the presence of SSR motifs using MREPS (http://bioinfo.lifl.fr/mreps/; (Kolpakov et al., 2003) with default parameters counting repeats whose period was at least 2 and size at least 12 and a coverage of up to 1000 reads. A total of 5,835 reproductive TT (1,976 in pollen and 4,822 in stigma transcriptomes, Table 2) are having microsatellites on their sequences. In our previous experience (Fernandez-Pozo et al., 2011;Benzekri et al., 2014;Canales et al., 2014), detection of SNPs in natural populations provides a huge amount of data of very difficult interpretation (Benzekri et al., 2014); therefore, SNPs have not been predicted.

The flow template based on AutoFlow (Seoane et al., submitted) that automates the complete process from pre-processing to annotation is detailed in Supplementary Data file 2.

2.1.4. Expression data

Since the Roche FLX platform produces a limited number of reads in contrast to Illumina ultrasequences, libraries were combined to obtain a pool of reads from pollen (libraries PM-Subs, PM, PG1 and PG2, 373,268 reads), stigma (libraries S2, S3, S4, S4-Subs, 430,165 reads) and vegetative tissues (libraries L, L-Subs, R1 and R2, 89,403 reads). These reads were mapped to all reference TT included in ReproOlive (Table 2) using Bowtie2 (Langmead and Salzberg, 2012) and allowing each read to map in every complementary TT. Mapped reads were counted with Bio-samtools from BioRuby (Goto et al., 2010) and included in the database as row counts (available for download to be analysed with other software) or as RPKM values (for comparing purposes in order to clearly identify TTs specific or not from pollen and/or stigma).

2.2. Database Construction

2.2.1. Implementation and Architecture

ReprOlive runs with the Apache HTTP Server 2 and MySQL 5 database management system in Linux OS. Ruby On Rails 2.3.11 (http://rubyonrails.org/) scripts were used to create the user interface on HTML 5 coupled with MySQL to use of a model-view-controller pattern to maintain strict separation between the web interface (views) code, database contents (models), and all methods

that handle interactions between views and database (controllers). This allowed to divide the database in four different virtual machines (Fig. 1): one for the web interface, one for the database content, one for calculus methods (e.g., blast queries) and the fourth for linked data (semantic) search. BioRuby (Goto et al., 2010) is required for some importation and managing tasks. The functionality of the Linked Data search was implemented using a SPARQL EndPoint (a service to send queries to the RDF database) provided by an instance of Virtuoso Open-Source Edition. RDF (Resource Description Framework) information has been produced using D2RQ (dump-rdf script), mapping the database schema with one application ontology (available at http://150.214.214.6/olivedb.owl). The use of independent virtual machines distributes tasks between machines, allowing for multiple, concomitant browsing and searching capabilities.

2.2.2. Availability and Updates

ReprOlive is freely available at http://reprolive.eez.csic.es. Bulk imports, updates, and database managements were automated: when source data are saved in *import_new_projects* folder, the database automatically launches the necessary Ruby gems that import sequences, annotations and expression data into a new assembly version of the database. Therefore, updates of ReprOlive transcriptomes with re-assembled and re-annotated TTs, and new expression data, will be automatically incorporated, making the database easily scalable, maintainable and expandable. Implementation based on independent virtual machines makes ReprOlive easily clonable and adaptable to any computer environment without complicated installations.

3. Results and Discussion

3.1. Web Interface and Navigation

Since molecular sequence databases are fundamental resources for modern bioscientists, ReprOlive currently houses annotated sequences of olive tree pollen, stigma, a small set of vegetative tissues, and a tentative transcriptome combining reproductive tissues (Table 2 and Fig. 2A). It has been designed with a user-friendly interface that can be browsed anonymously to facilitate researchers to access to this information. There is a navigation bar containing buttons for database mining from different entry points and based on different criteria, including three different possibilities of search.

3.1.1. Home Page

This is the default entry page where general information is offered in the three panels. The left panel contains links to the version history of the database, the scheme of the pipeline that produced the last version as automatically provided with AutoFlow, and the history of visits. On the right panel, information about tool versions used in the assembly pipeline, the current database release and the funding credits. This page can be recalled by means of the «Home» button in the navigation bar.

3.1.2. Assemblies

This button opens the main start point for database navigation by means of tab panels; note that the

word 'assemblies' is used here as synonym of 'transcriptomes'. The «All assemblies» tab is shown by default and presents all available transcriptomes (Fig. 2A). The reproductive transcriptome is selected by default, but the user can change the selection, taking into account that only one transcriptome can be browsed at a time. The next tab, «Assembly info,» is divided into two sides, the left side containing detailed information about the chosen assembly. The right side gives the possibility of (i) downloading the whole set of TTs in Fasta format, (ii) downloading only the reference TT with annotations, (iii) downloading sequence and annotations for a custom set of TT identifiers, and (iv) downloading raw expression data. These capabilities have been designed to provide data for further use in external tools instead of embedding cumbersome, bioinformatic tools in the own database.

3.1.3. Tentative transcripts' tab

This panel allows for the navigation through all TTs in the assembly (transcriptome) in a paginated way (Fig. 2B). Each TT code is illustrated with relevant information, such as its length, descriptions, ORF status, and if it is a reference TT (column 'Where'). Since consistency of descriptions is a sign of reliable annotation, common words in the description for one TT are marked in green. There are three different ways of filtering TTs (identifiable in different rows in Fig. 2B):

1) The default view corresponds to the «Annotated transcripts» button, but all TTs can be shown using the «All transcripts» button (Fig. 2B, first row of grey buttons).

2) Selection of TTs that comply any of the ORF statuses shown in buttons highlighted in grey. For example, «Complete» button (Fig. 2B, second row of grey buttons) filters out TTs that do not code for a complete protein; «Coding» button retains TTs without predicted ORF that should code for a protein based on the TransDecoder test. Also, TTs corresponding to putative ncRNA precursors can be selected.

3) Selection of TTs by their name or by words in the description (pop-up menu and text field in Fig. 2B). This allows finding a particular TT (e.g. the cysteine protease coded by 'rp11_olive_018645'), or a family of sequences (e.g., for the reproductive transcriptome, the 115 cysteine proteinases in ReprOlive, or the 16 complete ORFs coding for cysteine proteinases).

Clicking on one TT identifier, the complete information about it is shown as a pop-up text (Fig. 3). The first one shows the sequence and a button to download it in Fasta form. The «Annotations» block contains the following tables: 1) the assigned descriptions by Sma3s and Full-LengtherNext as a user-friendly way to offer information about tentative functions; 2) tables for gene names, GOs, ECs, KEGG pathways, each one accompanied by the associated *E* value to enable another empirical assessment of annotation quality. 3) the InterPro signatures, which add high-valued annotations, such as functional sites, protein families or conserved domains, with a single search (Hunter et al., 2012); and 4) the TAIR and RefSeq orthologues of *Arabidopsis thaliana* (Table 2), permitting gene-enrichment and functional analyses with a non-model species such as olive tree. The «ORF prediction» block (Fig. 3) comes from the Full-LengtherNext predictions, providing the putative ORF, if this ORF is complete, the position of start and/or stop codons, and the alignment that allows such predictions. The ORF prediction is an extremely useful information that will find direct use in laboratories, for example in designing primers to clone ORFs, or designing 3'-probes that discern between closely related TTs. Finally, if the TT shown is part of the reference transcriptome, the «Expression data» block will show the raw counts and RPKM of this TT in the three types of tissues

included in ReprOlive (pollen, stigma and vegetative). These data clearly display if the expression of this TT is specific, up-regulated or down-regulated in any of the samples. For example, the TT shown in Fig. 3 is more expressed in stigma than in pollen, and is not expressed in vegetative (leaf and root) tissues, making it a good candidate for a specific reproductive TT.

3.1.4. SSR tab

Plant cDNAs from natural orchards should be heterozygous and contain a high frequency of polymorphisms. Since microsatellites (SSRs) occur frequently in most eukaryote genomes and can be very informative, multi-allelic and reproducible, this panel shows the list of TTs having at least one SSR available and the TT where it is found. Each SSR motif is shown as a tetrad containing its sequence, the TT that contains it, and the start and end positions. SSRs can be filtered by the number of nucleotides in the motif and by their length, revealing that reproductive transcriptome has hexanucleotide motifs in 501 TTs, tetranucleotide motifs in 575 TTs, and that 493 SSRs have more than 20 nt in length. SSRs have direct applications as molecular markers since they are easily converted in primers (Guerrero et al., 2010) that provide co-dominant and stable results (Abdellatif and Khidr, 2010) that overcome the limitations of other types of molecular markers (Garcia et al., 2004). Moreover, ORF-based SSRs are more advantageous since they will reduce the mapping efforts required for the development of high-density maps and association studies, and will facilitate comparative genomics.

3.1.5. Descriptions, GO, EC and InterPro Tabs

The TT annotations in Fig. 3 can also be browsed by means of their respective tab panels. Unfortunately, since descriptions were written by humans, it is frequent to find different descriptions for the same sequence (Fig. 4A), as can be deduced by the fact that there are 82,334 descriptions for 63,965 TTs with functional annotations. Clicking on one description, the collection of TTs sharing it is displayed. Tab panels for browsing through the 45,781 GOs, the 10,003 ECs, and the 187,899 InterPros behave as the Description tab panel.

3.1.6. KEGG Pathways' Tab

The KEGG Pathways tab panel shows in a paginated way the 146 reproductive tissue pathways (143 in pollen and 146 in stigma) and 145 vegetative pathways sorted by the number of ECs ("Present ECs") identified in ReprOlive (Fig 4B). The pathway codes, the total number of pathway enzymes ("Total ECs") as well pathway coverages ("Coverage (%)"), are also displayed. Among the most covered pathways in reproductive transcriptome are 91% coverage of glucosinolate biosynthesis (map 00966), 80% coverage of betalain biosynthesis (map 00965), 75% coverage of brassinosteroid biosynthesis (map 00905) and DDT degradation (map 00351), 71% coverage of carbon fixation (map00710), 68% coverage of α-linolenic acid metabolism (map 00592), and 56% coverage of flavonoid biosynthesis (map 00941) and phenylpropanoid biosynthesis (map 00940).

When clicking on the pathway name, the panel contents change to show the EC list present in ReprOlive for that pathway, and its image displaying in green the enzymes found in the database (Fig. 4C). The EC names can be deployed to show which ReprOlive TTs have this function assigned. This allows the selection for specific TTs coding the complete protein, such as 'rp11_olive_007935'

for 4.2.1.92 (hydroperoxide dehydratase). The coloured pathway image is interactive and allows the navigation to the KEGG database to obtain more information.

3.1.7. Expression Tab

This panel shows the list of reference TTs with the raw number of reads mapped in this TT or the RPKM as a comparable value (Fig. 4D). The first line makes reference to the total number of reads used to map and the total length of the transcriptome to obtain a mean RPKM (total number of reads by the total length of the reference transcriptome) to colour RPKM values in green to indicate that it is over the transcriptome mean, and in red when it is below this mean. These values must only be considered orientative, since RPKM is not the best normalisation measure (Wagner et al., 2012), and since RNA-seq values correspond to pyrosequence mapping (and not short read mapping). However, this value can help in determining the tissue where the TT is preferentially expressed. For example, the rp11_olive_005693 (a POZ/BTB domain containing protein) is expressed in the three tissues, mainly in stigma; the pollen seems to specifically express a significant proportion of uncharacterised proteins, such as rp11_olive_028897, and a probable hydrolase (rp11_olive_026947) seems to be expressed only in reproductive tissues, mainly in pollen. Raw expression data are downloadable from the Assembly Info tab panel.

The utility of ReprOlive annotations together with the expression values expressed as RPKM values is shown in Supplementary file 3, where the 1655 TTs that are expressed only in pollen (as per their RPKM) and have a valid RefSeq orthologue have been analysed using GOrilla (Eden et al., 2009) using the default parameters. The enriched GO terms those pollen-specific TTs correspond to pollen tube growth, actin filament organisation, plant cell wall organisation and modification, polysaccharide catabolic process, and carbohydrate transport, to cite the most representative biological processes. Considering the cell component, they are mainly expressed in the extracellular region, followed by the pollen tube and the plant cell wall. These results support the experimental procedure of the sequenced libraries, the assembly, the orthologue annotation and even the RPKM values in spite of they are calculated from Roche/454 reads.

3.1.8. Blast Search

TTs can be retrieved by sequence similarity using Blast+ (Camacho et al., 2009). A blast-based search engine with customisable *E*-value for nucleotide (blastn) or amino acids (blastx) has been implemented. The type of sequence (amino acid or nucleotides) is automatically detected. Blast searches are conducted against the transcriptomes selected by the corresponding checkbox. Blast executions are queued and the URL where the final result will be stored for a month is shown when the task is finished. The user can download the results as an HTML file (the same information shown on screen) or as the direct blast output.

3.1.9. Annotation Search

This page it is complementary to the filtrations by words in descriptions, by the size of SSR, by the ORF status, etc., included within some «Assemblies» panels, since it searches the database by combining GOs, ECs, InterPros, descriptions, orthologues and gene names. The search can combine using AND or OR with all those fields. The search can be restricted to only one of the transcriptomes

in ReprOlive by means of checkboxes. As a result, a paginated list of TT fulfilling the requirements is shown and TT sequences can be downloaded in Fasta format.

3.1.10. Linked-Data Search

A novelty of ReprOlive with respect to other plant databases is that its information has also been published as structured in RDF format. This allows its interlinking with other semantic databases, such as UniProtKB. Since most TTs in ReprOlive (mainly the reference transcriptome) have an *Arabidopsis* orthologue with TAIR10/RefSeq IDs, the Linked-Data search can retrieve information about 3D-structures (PDB database), allergens (Allergome database), interactions (STRING and IntAct databases) and enzyme data (BRENDA, BioCyc, KEGG and Reactome databases). The advantages of this semantic search are gaining access to updated information of external databases, and complementing and extending the stored information in ReprOlive.

Semantic capability of ReprOlive starts with the automatic selection of which TTs will be the semantic seed. The first line of the "Linked-data search" tab enables to collect TTs sharing IDs, GOs, ECs or InterPro codes. Clicking on the button "Get local data", related information on ReprOlive is provided. But clicking on "Search", the *Arabidopsis* orthologue of every TT is extracted and used to recover external information concerning structures, interactions, allergens or enzyme data. Retrieved information can be saved using the "Download results" button.

An example of use can start from the rp11_olive_029403 TT corresponding to transcription factor similar to UNE12 in *Arabidopsis*. This TT is only expressed in both reproductive tissues and not in vegetative tissues. This TT is annotated with GO:0080147 corresponding to root hair cell development. Using this GO as seed, its interaction network in STRING database was recovered (Fig. 5), showing that rp11_olive_029403 TT codes for a protein of unfertilized embryo sac involved in double fertilization forming a zygote and endosperm. In addition, it is co-expressed with other transcription factors of known (AT1G03040) and unknown (AT5G03500, AT5G03495) function, a protein involved in phosphate starvation (SPX4), a protein of phytochrome response (PAR1), as well as proteins of unknown functions (AT3G27420, AT5G11980, AT4G24840). Its function as transcription factor and the interactions with many unknown proteins makes rp11_olive_029403 TT a good regulator candidate for reproductive tissues.

3.2. The Reproductive Transcriptome According to ReprOlive

3.2.1. ReprOlive is a complementary source of information for olive tree transcriptome

The ReprOlive reproductive transcriptome includes a significantly higher number of final TTs than those provided by most studies (Alagna et al., 2009;Galla et al., 2009;Ozgenturk et al., 2010), although lower than the reported by Muñoz-Mérida et al. (Muñoz-Merida et al., 2013) resulting from the screening of numerous vegetative tissues and stages. The TT mean length is significantly higher than previous studies likely due to the use of Titanium+ technology. As a result, ReprOlive TTs are highly complementary to previous studies, maybe representing the only publicly available annotated database fully dedicated to olive tree transcriptome, including tools for different types of search and functional and structural annotations. Some other publicly available databases including olive

transcriptome sequences like NCBI (e.g. SRX193576 accession), Oleaestdb (http://140.164.45.140/oleaestdb/index.php) (Alagna et al., 2009), and the European Nucleotide Archive (http://www.ebi.ac.uk/ena/home) (e.g. SRR592583 accession), either include raw sequences only, or a lower degree of operative resources. In conclusion, ReprOlive may help researches devoted to either plant-reproduction or other disciplines to retrieve relevant information on olive transcriptome.

3.2.2. Reproductive Transcriptome Expresses Approximately Half of Olive Tree Genes, Mainly in Stigma

As expected, pollen and stigma have TTs in common since (1) the number of TTs in reproductive transcriptome is below the sum of pollen and stigma transcriptomes (Table 2); (2) stigma and reproductive transcriptomes contain 146 pathways while pollen contains only 143; and (3) there are TTs whose expression data indicate that both are expressed in pollen and stigma. Row «Number of TTs with annotation» of Table 2 shows that 87.8% TTs were functionally annotated, although not all annotated TTs contain an orthologue. The number of unique orthologues (Table 2, «Unique IDs» rows) indicates that stigma transcriptome seems to be more complex than the pollen transcriptome Considering the number of orthologues with *A. thaliana* in TAIR10 and RefSeq (15,503 and 17,612, respectively), it can be suggested that the reproductive transcriptome is a subset ranging from 55% to 62% of the complete transcriptome if it is assumed that the olive tree genome, like *A. thaliana* (Lamesch et al., 2012), contains ~27,200 protein-coding genes.

3.2.3. The High Proportion of Full-Length ORFs Reveals a Reliable Transcriptome

Sequencing and assembling where highly successful since the number of chimeras is very low and the number of complete ORFs is 12,6% of the transcriptome, and 24,8% of the unique IDs, which is clearly above other transcriptomes builded with the same strategy (Benzekri et al., 2014;Canales et al., 2014). Having a large collection of full-length protein sequences is crucial for accurate annotation, comparative analysis between transcriptomes, and also for obtaining accurate gene expression profiles related to growth, development and environmental changes. In fact, an important collection of those full-length ORFs (such as the numerous forms of the Ole e 1 and Ole e 2 [profilins] allergens, Cu,Zn-superoxide dismutases, catalases, peroxidases, NADPH oxidase, enzymes of the glutathion-ascorbate cycle and thioredoxins) have been cloned by the authors based on the sequence in the database (results not shown), which confirms their reliability and utility.

3.2.4. Tentative Transcripts without Orthologue as a Source of Putative New Olive Specific Transcripts

The Full-LengtherNext analysis shown in Table 2 (column “Reproductive”) is quite strict in assigning an orthologue, since 17,445 TTs do not have one. The TransDecoder analysis, contained in Full-LengtherNext, revealed that 7,114 TTs could code for a protein, suggesting that 10,331 TTs can be discarded from the reproductive transcriptome. From the likely coding TTs, 628 of them code for a complete protein and 6,486 for an incomplete one. After the functional annotation using Sma3, 80 (12.7%) of likely complete TTs and 1,750 (27%) of likely incomplete TTs remain with no functional annotation at all, and other 134 (21.3%) and 1,612 (24.8%), respectively, are annotated as “uncharacterized” protein. These subsets of coding TTs should include some kind of olive-specific

TTs, opening new research opportunities in olive tree for deciphering their function.

3.2.5. Reference Transcriptomes Seem to Gather the Olive Tree Heterozygosity

As stated above, the size of transcriptomes is over-representing the putative number of olive tree genes, representing the maximal number of TTs expressed in reproductive tissues. Therefore, a subset of the transcriptome including the longest TTs with unique, different orthologous ID, and the longest, >500 bp, non-redundant unknown TT with coding or putative coding status are provided as a kind of "Reference transncriptome" (Table 2. It is useful for expression studies, such as expression analyses included in the «Expression» panel of the database. Sequences belonging to a reference transcriptome be easily identified by a «REF» tag (Figs. 2 and 3) on their description. Since the numbers of unique RefSeq and TAIR10 IDs in complete (17,612 and 15,503, respectively) and reference (14,706 and 13,584, respectively) reproductive transcriptomes are quite close, it is suggested that, even if some genes could be lost, the Reference Transcriptome is representative of genes expressed in reproductive tissues. Moreover, two alleles for every locus seem to be included in the Reference Transcriptomes since the number of unique TAIR10/RefSeq IDs is ~65% of the total TAIR10/RefSeq IDs (Table 2, rows below "Reference transcriptome").

4. Conclusions and Future Prospects

ReprOlive offers transcriptomic information related to olive tree reproductive tissues (with leaf and root as vegetative control) with unrestricted public access. It contains sufficient information on TTs that can be used for genomics, molecular studies, genetic maps, expression analyses, new allergen detection, and even future breeding purposes. It offers a comprehensive on-line system for information retrieval and management, and has help in the mining of reproductive transcriptome. The availability of a reference transcriptome with preliminary expression data and putative olive-specific genes give chances to new research areas. ReprOlive has also been published as a standard semantic conceptualization in RDF, enabling its integration with other RDF-based databases to provide distributed, updated annotation as well as data integration. Thus, ReprOlive joins the most novel approach to publish Open Data as previously done by relevant databases as UniProtKB (http://beta.sparql.uniprot.org/sparql/). In a near future, more olive sequences from public resources and our own research studies will be collected and archived, including future RNA-seq data sets, in order to provide the most complete information about the overall olive tree transcriptome. This may include other reproductive cells, tissues and organs of interest (such as isolated meiocytes, tapetum, endosperm, mesocarp, ovary and embryo sac), as well as additional developmental stages and olive cultivars of interest. Therefore, ReprOlive (both in its present form and thorough future developments) may help researches devoted to either plant-reproduction or other disciplines to retrieve relevant information on olive transcriptome.

5. Acknowledgement

The authors gratefully acknowledge Rocío Bautista for helpful discussions and bioinformatic counselling, and Josefa Gómez Maldonado and Concepción Ávila for highly valuable pyrosequencing at the sequencing facilities of the University of Málaga. The authors also thankfully

458 acknowledge the sequencing facilities, the computer resources and the technical support provided by
459 the Plataforma Andaluza de Bioinformática of the University of Málaga. This work was supported by
460 co-funding from the ERDF (European Regional Development Fund) and (i) Ministerio de Ciencia e
461 Innovación [grant numbers BFU2011-22779 to JDA, and TIN2011-25840 and TIN2014-58304-R to
462 JFAM], and (ii) Plan Andaluz de Investigación, Desarrollo e Innovación [grant numbers P10-CVI-
463 6075 to MGC, P10-AGR-6274 and P11-CVI-7487 to JDA, and P11-TIC-7529 and P12-TIC-1519 to
464 JFAM].

465

466 6. References[1]

467 Abdellatif, K.F., and Khidr, Y.A. (2010). Genetic diversity of new maize hybrids based on SSR markers as
468 compared with other molecular and biochemical markers. *J. Crop. Sci. Biotechnol.* 13**,** 139-145.
469 Alagna, F., D'agostino, N., Torchia, L., Servili, M., Rao, R., Pietrella, M., Giuliano, G., Chiusano, M.L., Baldoni,
470 L., and Perrotta, G. (2009). Comparative 454 pyrosequencing of transcripts from two olive genotypes
471 during fruit development. *BMC Genomics* 10**,** 399. doi: 10.1186/1471-2164-10-399.
472 Barcaccia, G., Botton, A., Galla, G., Ramina, A., Muleo, R., Baldoni, L., and Perrotta, G. (2012). Comparative
473 genomics for identifying flower organ identity genes in peach and olive. *Acta Hort* 967**,** 43-53.
474 Bazakos, C., Manioudaki, M.E., Therios, I., Voyiatzis, D., Kafetzopoulos, D., Awada, T., and Kalaitzis, P. (2012).
475 Comparative transcriptome analysis of two olive cultivars in response to NaCl-stress. *PLoS One* 7**,**
476 e42931. doi: 10.1371/journal.pone.0042931.
477 Benzekri, H., Armesto, P., Cousin, X., Rovira, M., Crespo, D., Merlo, M.A., Mazurais, D., Bautista, R., Guerrero-
478 Fernandez, D., Fernandez-Pozo, N., Ponce, M., Infante, C., Zambonino, J.L., Nidelet, S., Gut, M.,
479 Rebordinos, L., Planas, J.V., Begout, M.L., Claros, M.G., and Manchado, M. (2014). De novo assembly,
480 characterization and functional annotation of Senegalese sole (Solea senegalensis) and common sole
481 (Solea solea) transcriptomes: integration in a database and design of a microarray. *BMC Genomics*
482 15**,** 952. doi: 10.1186/1471-2164-15-952.
483 Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., and Madden, T.L. (2009).
484 BLAST+: architecture and applications. *BMC Bioinformatics* 10**,** 421. doi: 10.1186/1471-2105-10-421.
485 Canales, J., Bautista, R., Label, P., Gomez-Maldonado, J., Lesur, I., Fernandez-Pozo, N., Rueda-Lopez, M.,
486 Guerrero-Fernandez, D., Castro-Rodriguez, V., Benzekri, H., Canas, R.A., Guevara, M.A., Rodrigues, A.,
487 Seoane, P., Teyssier, C., Morel, A., Ehrenmann, F., Le Provost, G., Lalanne, C., Noirot, C., Klopp, C.,
488 Reymond, I., Garcia-Gutierrez, A., Trontin, J.F., Lelu-Walter, M.A., Miguel, C., Cervera, M.T., Canton,
489 F.R., Plomion, C., Harvengt, L., Avila, C., Gonzalo Claros, M., and Canovas, F.M. (2014). De novo
490 assembly of maritime pine transcriptome: implications for forest breeding and biotechnology. *Plant*
491 *Biotechnol J* 12**,** 286-299. doi: 10.1111/pbi.12136.
492 Collani, S., Galla, G., Ramina, A., Barcaccia, G., Baldoni, L., Alagna, F., Caceres, E.M., Muleo, R., and Perrotta,
493 G. (2012). Self-Incompatibility in olive: a new hypothesis on the S-locus genes controlling pollen-pistil
494 interaction. *Acta Hort* 967**,** 133-140.
495 Dickinson, H.G., and Franklin-Tong, N. (2011). Preface to special issue on plant reproductive development:
496 from recombination to seeds. *J. Exp. Bot.* 62**,** 1531-1532.
497 Dukowic-Schulze, S., and Chen, C. (2014). The meiotic transcriptome architecture of plants. *Front Plant Sci* 5**,**
498 220. doi: 10.3389/fpls.2014.00220.

[1] Provide the doi when available, and ALL complete author names.

499 Eden, E., Navon, R., Steinfeld, I., Lipson, D., and Yakhini, Z. (2009). GOrilla: a tool for discovery and
500 visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics* 10**,** 48. doi:
501 10.1186/1471-2105-10-48.
502 El Kelish, A., Zhao, F., Heller, W., Durner, J., Winkler, J.B., Behrendt, H., Traidl-Hoffmann, C., Horres, R.,
503 Pfeifer, M., Frank, U., and Ernst, D. (2014). Ragweed (*Ambrosia artemisiifolia*) pollen allergenicity:
504 SuperSAGE transcriptomic analysis upon elevated CO2 and drought stress. *BMC Plant Biol* 14**,** 176.
505 doi: 10.1186/1471-2229-14-176.
506 Engel, M.L., Chaboud, A., Dumas, C., and Mccormick, S. (2003). Sperm cells of Zea mays have a complex
507 complement of mRNAs. *Plant J* 34**,** 697-707.
508 Falgueras, J., Lara, A.J., Fernandez-Pozo, N., Canton, F.R., Perez-Trabado, G., and Claros, M.G. (2010).
509 SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC*
510 *Bioinformatics* 11**,** 38. doi: 10.1186/1471-2105-11-38.
511 Fernandez-Pozo, N., Canales, J., Guerrero-Fernandez, D., Villalobos, D.P., Diaz-Moreno, S.M., Bautista, R.,
512 Flores-Monterroso, A., Guevara, M.A., Perdiguero, P., Collada, C., Cervera, M.T., Soto, A., Ordas, R.,
513 Canton, F.R., Avila, C., Canovas, F.M., and Claros, M.G. (2011). EuroPineDB: a high-coverage web
514 database for maritime pine transcriptome. *BMC Genomics* 12**,** 366. doi: 10.1186/1471-2164-12-366.
515 Galla, G., Barcaccia, G., Ramina, A., Collani, S., Alagna, F., Baldoni, L., Cultrera, N.G., Martinelli, F., Sebastiani,
516 L., and Tonutti, P. (2009). Computational annotation of genes differentially expressed along olive
517 fruit development. *BMC Plant Biol* 9**,** 128. doi: 10.1186/1471-2229-9-128.
518 Garcia, A.a.F., Benchimol, L.L., Barbosa, A.M.M., Geraldi, I.O., Souza Jr., C.L., and De Souza, A.P. (2004).
519 Comparison of RAPD, RFLP, AFLP and SSR markers for diversity studies in tropical maize inbred lines.
520 *Genet. Mol. Biol.* 27**,** 579-588.
521 Gil-Amado, J.A., and Gomez-Jimenez, M.C. (2013). Transcriptome analysis of mature fruit abscission control
522 in olive. *Plant Cell Physiol* 54**,** 244-269. doi: 10.1093/pcp/pcs179.
523 Goto, N., Prins, P., Nakao, M., Bonnal, R., Aerts, J., and Katayama, T. (2010). BioRuby: bioinformatics software
524 for the Ruby programming language. *Bioinformatics* 26**,** 2617-2619. doi:
525 10.1093/bioinformatics/btq475.
526 Guerrero, D., Bautista, R., Villalobos, D.P., Canton, F.R., and Claros, M.G. (2010). AlignMiner: a Web-based
527 tool for detection of divergent regions in multiple sequence alignments of conserved sequences.
528 *Algorithms Mol Biol* 5**,** 24. doi: 10.1186/1748-7188-5-24.
529 Hunter, S., Jones, P., Mitchell, A., Apweiler, R., Attwood, T.K., Bateman, A., Bernard, T., Binns, D., Bork, P.,
530 Burge, S., De Castro, E., Coggill, P., Corbett, M., Das, U., Daugherty, L., Duquenne, L., Finn, R.D.,
531 Fraser, M., Gough, J., Haft, D., Hulo, N., Kahn, D., Kelly, E., Letunic, I., Lonsdale, D., Lopez, R., Madera,
532 M., Maslen, J., Mcanulla, C., Mcdowall, J., Mcmenamin, C., Mi, H., Mutowo-Muellenet, P., Mulder, N.,
533 Natale, D., Orengo, C., Pesseat, S., Punta, M., Quinn, A.F., Rivoire, C., Sangrador-Vegas, A., Selengut,
534 J.D., Sigrist, C.J., Scheremetjew, M., Tate, J., Thimmajanarthanan, M., Thomas, P.D., Wu, C.H., Yeats,
535 C., and Yong, S.Y. (2012). InterPro in 2011: new developments in the family and domain prediction
536 database. *Nucleic Acids Res* 40**,** D306-312. doi: 10.1093/nar/gkr948.
537 Kamalay, J.C., and Goldberg, R.B. (1980). Regulation of structural gene expression in tobacco. *Cell* 19**,** 935-
538 946.
539 Kolpakov, R., Bana, G., and Kucherov, G. (2003). mreps: Efficient and flexible detection of tandem repeats in
540 DNA. *Nucleic Acids Res* 31**,** 3672-3678.
541 Lamesch, P., Berardini, T.Z., Li, D., Swarbreck, D., Wilks, C., Sasidharan, R., Muller, R., Dreher, K., Alexander,
542 D.L., Garcia-Hernandez, M., Karthikeyan, A.S., Lee, C.H., Nelson, W.D., Ploetz, L., Singh, S., Wensel, A.,
543 and Huala, E. (2012). The Arabidopsis Information Resource (TAIR): improved gene annotation and
544 new tools. *Nucleic Acids Res* 40**,** D1202-1210. doi: 10.1093/nar/gkr1090.
545 Langmead, B., and Salzberg, S.L. (2012). Fast gapped-read alignment with Bowtie 2. *Nat Methods* 9**,** 357-359.
546 doi: 10.1038/nmeth.1923.
547 Liang, F., Holt, I., Pertea, G., Karamycheva, S., Salzberg, S.L., and Quackenbush, J. (2000). An optimized

protocol for analysis of EST sequences. *Nucleic Acids Res* 28**,** 3657-3665.

Muleo, R., Morgante, M., Velasco, R., Cavallini, A., Perrotta, G., and Baldoni, L. (2012). "Olive Tree Genomic," in *Olive Germplasm – The Olive Cultivation, Table Olive and Olive Oil Industry in Italy,* ed. I. Muzzalupo. (Rijeka, Croatia: InTech), 133–148.

Muñoz-Merida, A., Gonzalez-Plaza, J.J., Canada, A., Blanco, A.M., García-López, M.D.C., Rodríguez, J.M., Pedrola, L., Sicardo, M.D., Hernández, M.L., De La Rosa, R., Belaj, A., Gil-Borja, M., Luque, F., Martínez-Rivas, J.M., Pisano, D.G., Trelles, O., Valpuesta, V., and Beuzón, C.R. (2013). *De novo* assembly and functional annotation of the olive (*Olea europaea*) transcriptome. *DNA Res* 20**,** 93-108. doi: 10.1093/dnares/dss036.

Muñoz-Mérida, A., Viguera, E., Claros, M.G., Trelles, O., and Pérez-Pulido, A.J. (2014). Sma3s: A Three-Step Modular Annotator for Large Sequence Datasets. *DNA Res*. doi: 10.1093/dnares/dsu001.

Ozgenturk, N.O., Oruç, F., Sezerman, U., Kuçukural, A., Korkut, S.V., Toksoz, F., and Un, C. (2010). Generation and Analysis of Expressed Sequence Tags from *Olea europaea* L. *Compar Funct Genomics* 2010**,** 757512.

Pruitt, K.D., Tatusova, T., Brown, G.R., and Maglott, D.R. (2012). NCBI Reference Sequences (RefSeq): current status, new features and genome annotation policy. *Nucleic Acids Res* 40**,** D130-135. doi: 10.1093/nar/gkr1079.

Turktas, M., Inal, B., Okay, S., Erkilic, E.G., Dundar, E., Hernandez, P., Dorado, G., and Unver, T. (2013). Nutrition metabolism plays an important role in the alternate bearing of the olive tree (Olea europaea L.). *PLoS One* 8**,** e59876. doi: 10.1371/journal.pone.0059876.

Villalba, M., Rodriguez, R., and Batanero, E. (2014). The spectrum of olive pollen allergens. From structures to diagnosis and treatment. *Methods* 66**,** 44-54. doi: 10.1016/j.ymeth.2013.07.038.

Wagner, G.P., Kin, K., and Lynch, V.J. (2012). Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. *Theory Biosci* 131**,** 281-285. doi: 10.1007/s12064-012-0162-3.

Wang, Y., Zhang, W.Z., Song, L.F., Zou, J.J., Su, Z., and Wu, W.H. (2008). Transcriptome analyses show changes in gene expression to accompany pollen germination and tube growth in Arabidopsis. *Plant Physiol* 148**,** 1201-1211. doi: 10.1104/pp.108.126375.

Wei, L.Q., Xu, W.Y., Deng, Z.Y., Su, Z., Xue, Y., and Wang, T. (2010). Genome-scale analysis and comparison of gene expression profiles in developing and germinated pollen in Oryza sativa. *BMC Genomics* 11**,** 338. doi: 10.1186/1471-2164-11-338.

Zafra, A., Rodriguez-Garcia, M.I., and Alche Jde, D. (2010). Cellular localization of ROS and NO in olive reproductive tissues during flower development. *BMC Plant Biol* 10**,** 36. doi: 10.1186/1471-2229-10-36.

583 ## 7. Tables

584

585 **Table 1**: Gene libraries used in ReprOlive

Gene library	Tissue	Developmental stage	Sequencing method	Raw reads	Useful reads
PM-Subs	Pollen	Mature	Sanger	666	518
PM	Pollen	Mature	Pyrosequencing	216,497	111,242
PG1	Pollen	1 h germination	Pyrosequencing	258,167	141,232
PG5	Pollen	5 h germination	Pyrosequencing	233,921	120,276
S2	Stigma	Stage2	Pyrosequencing	257,813	138,077
S3	Stigma	Stage3	Pyrosequencing	247,401	141,903
S4	Stigma	Stage4	Pyrosequencing	262,269	149,929
S4-Subs	Stigma	Stage4	Sanger	480	256
L	Leaf	Mature	Pyrosequencing	223,399	41,178
L-Subs	Leaf	Mature	Sanger	403	251
R1	Root	Mature	Pyrosequencing	231,237	25,899
R2	Root	Radicle	Pyrosequencing	145,204	22,075

586

587

588 **Table 2**: Main features of transcriptomes in ReprOlive based on Full-LengtherNext analyses

Feature	Pollen	Stigma	Vegetative	Reproductive
Assembling statistics				
Number of useful reads	373,268	430,165	89,403	803,433
Mean length (nt)	383	385	545	384
Number of MIRA3 contigs	54,754	73,823	42,310	116,298
Number of Euler-SR contigs	4,807	15,216	490	16,211
Number of Contigs after CAP3 reconciliation	28,094	60,964	39,425	73,589
Number of TTs without chimeras and artefacts*	27,823	60,400	38,919	72,846
Mean length (nt)	608	678	664	686
N50 (nt)	661	780	683	798
Annotation statistics				
Longest TT (nt)	7,016	7,757	2,865	7,950
Number of ncRNAs	31	17	265	45
Number of TTs with annotation	24,861	54,129	36,700	63,965
Number of TTs with orthologue	21,607	46,910	32,076	55,356
with unique orthologue IDs	11,672	21,326	15,003	23,568
with orthologue from *A. thaliana* RefSeq	21,233	46,924	31,945	54,890
unique RefSeq IDs	9,769	16,565	12,489	17,612
with orthologue from *A. thaliana* TAIR10	21,312	47,038	31,980	55,067
unique TAIR10 IDs	8,922	14,656	11,247	15,503
Number of TTs coding a complete protein	2,809	7,137	3,559	9,157
unique, complete proteins	1,976	4,822	2,220	5,835
Number of TTs without orthologue	6,185	13,473	6,578	17,445
likely coding for a complete protein	170	446	242	628
likely coding for an incomplete protein	2610	5,312	2,523	6,486
Reference transcriptome				
Number of representative TTs	13,589	25,720	17,340	28,972
A. thaliana RefSeq orthologues	10,878	20,612	14,576	22,565
unique RefSeq IDs	8,281	13,901	10,349	14,706
A. thaliana TAIR10 orthologues	10,900	20,658	14,581	22,638
unique TAIR10 IDs	7,842	12,883	9,756	13,584

589 * Artefacts are internal, direct or inverse, repetitions

8. Figure legends

Figure 1: Database organisation in virtual machines. Using Web browsers, users will retrieve data from ReprOlive by querying or browsing through the Web interface. Web pages are generated on demand by the virtual machine that manages the Web interface. When queries require intensive calculations (for example, Blast comparisons), they are performed by the virtual machine for calculus. Queries to the RDF of ReprOlive are handled by the virtual machine for semantics. The use of different virtual machines is transparent for the user that can only see inputs and outputs in user's Web browser.

Figure 2: Screen captures providing a general overview of ReprOlive. **(A)** «All assemblies» panel showing the four transcriptomes included in the database. **(B)** First page of «Tentative transcripts» panel, where filtering criteria are accessible and information about each TT is displayed (see text for details). TTs are shown in pages of 20 sequences.

Figure 3: Information shown for one single TT, i.e. 'rp11_olive_022501' corresponding to an adenylate kinase where the «Annotation» and «ORF prediction» pop-up texts are deployed. See text for details.

Figure 4: Reproductive transcriptome illustrating the views of Description, KEGG Pathways and Expression tab panels. **(A)** List of TT description that illustrates the lack of consistence between descriptions provided by humans. **(B)** First page of KEGG Pathways showing the most populated ones in ReprOlive. **(C)** Example of α-linolenic acid metabolism pathway where enzymes in ReprOlive are highlighted in green; the TTs corresponding to 4.2.1.92 are deployed to show that there is at least one coding for the complete protein. **(D)** The seventh page illustrating different types of TT expression based on RPKMs.

Figure 5: Screen capture of the evidence view of interactions of UNE12 *Arabidopsis* orthologue to rp11_olive_029403 TT. This information can be shown in page http://string-db.org/newstring_cgi/show_network_section.pl?identifier=3702.AT4G02590.1-P at the STRING 9.1 database. The magenta colour of edges indicate that interactions were reported experimentally. The meaning of each node is explained at the figure bottom.

9. Supplementary Material

File 1: Flow diagram of the strategy for pre-processing, assembling and annotation of the transcriptomes described in this manuscript, where yellow, double-lined boxes are the inputs, and the black boxes provide the output results.

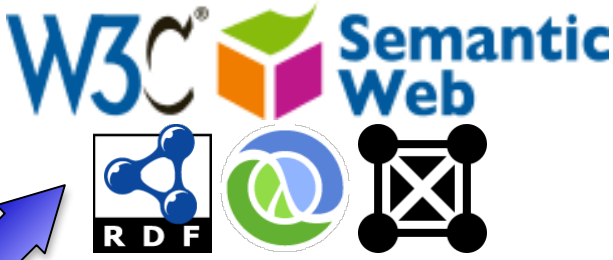
File 2: The flow template based on AutoFlow that automates the complete process from pre-processing to annotation. Execution AutoFlow with the parameter --graphic with this flow template produces its semantic representation as in Supplementary File 1.

File 3: GO enrichment using GOrilla of the 1655 TTs that are pollen-specific, sorted by their RPKM, that have an *Arabidopsis* orthologue in RefSeq. The first page contains the significant biological processes and the second page reveals the cellular component where the processes occur.

Virtual machine for database
(MySQL)

Virtual machine for calculus methods
(Blast, queue system)

Virtual machine for semantics
(Virtuoso)



Virtual machine for Web interface
(Apache, Ruby on Rails)

Query
Internet

Personal computers
(Web browser)

Assembly name	Organism	Tissue	Transcripts number	Description
pollen_transcriptome_v1.0	*Olea europaea*	pollen	27949	Complete transcriptome assembly of pollen.
stigma_transcriptome_v1.0	*Olea europaea*	stigma	60711	Complete transcriptome assembly of stigma.
reproductive_tissues_transcriptome_v1.0 (default)	*Olea europaea*	pollen and stigma	73247	Complete transcriptome assembly of pollen and stigma together.
vegetative_tissues_transcriptome_v1.0	*Olea europaea*	leaf, radicle and root	39017	Transcriptome assembly of vegetative tissues.

Name	Length	Software	Description	E-value	ORF Status	Where
olive_transcript_005400	140	• FL-Next	tr=Uncharacterized protein; Solanum tuberosum (Potato).	0.0008	Internal	REF
olive_transcript_005401	1868	• FL-Next	tr=Putative uncharacterized protein; Vitis vinifera (Grape).	0.0	Complete	REF
		• sma3	RING-type E3 ubiquitin ligase	3.188e-09		
olive_transcript_005402	140	• sma3	Glycosyltransferase	1.985e-33	Internal	REF
		• FL-Next	sp=Probable beta-1,4-xylosyltransferase IRX14; Arabidopsis thaliana (Mouse-ear cress).	1.0e-19		
olive_transcript_005403	1868	• sma3	Uncharacterized protein	0.0	Putative Complete	
		• FL-Next	sp=Cystathionine gamma-synthase, chloroplastic; Arabidopsis thaliana (Mouse-ear cress).	1.0e-18		
olive_transcript_005404	140	• FL-Next	tr=Uncharacterized protein; Oryza brachyantha.	0.071	Internal	REF
olive_transcript_005406	140	• FL-Next	sp=**probable exocyst complex component** 6; Arabidopsis thaliana (Mouse-ear cress).	7.0e-17	Internal	REF
		• sma3	**probable exocyst complex component** 6	4.151e-06		
olive_transcript_005407	1868	• FL-Next	sp=MATE efflux family protein 4, chloroplastic; Arabidopsis thaliana (Mouse-ear cress).	0.0	Putative C-terminus	
		• sma3	Enhanced disease susceptibility	8.433e-24		
olive_transcript_005408	140	• sma3	T-**complex** protein 1 **subunit gamma**	1.001e-20	Internal	
		• FL-Next	sp=T-**complex** protein 1 **subunit gamma**; Thalassiosira weissflogii (Marine diatom).	8.0e-20		

Assembly Name: reproductive_tissues_transcriptome_v1.0

Transcript Name: olive_transcript_022501
Length: 1060 nt
Where: REF

Transcript Fasta

Download Fasta

Annotations

Source	Descriptions	Term	Type	E-value	Identity %
sma3	Adenylate kinase	-	-	3.827e-35	-
FL-Next	sp=Adenylate kinase 2, chloroplastic; Arabidopsis thaliana (Mouse-ear cress).	-	-	0.0	75

Source	ECs	Term	Type	E-value	Identity %
sma3	Adenylate kinase.	2.7.4.3	-	0.0	-

Source	KEGGs	Term	Type	e value	Identity
sma3	Purine metabolism	00230		0.0	

Source	Gene names
sma3	ADK; PGSC0003DMG400013933; adk; adk3;

Source	GOs	Term	Type	E-value	Identity %
sma3	adenylate kinase activity	GO:0004017	Molecular Function	0.0	-
sma3	ATP binding	GO:0005524	Molecular Function	0.0	-
sma3		GO:0016776	Molecular Function	5.216e-07	-
sma3	nucleotide kinase activity	GO:0019201	Molecular Function	3.943e-07	-
sma3		GO:0019205	Molecular Function	9.482e-13	-
sma3		GO:0006139	Biological Process	1.971e-10	-

Source	InterPros	Term	Type	E-value	Identity %
sma3	Adenylate kinase/UMP-CMP kinase	IPR000850	-	0.0	-
sma3	Adenylate kinase subfamily	IPR006259	-	0.0	-
sma3	Adenylate kinase, active site lid domain	IPR007862	-	0.0	-
sma3	Domain of unknown function DUF1995	IPR018962	-	2.10195e-44	-
sma3	P-loop containing nucleoside triphosphate hydrolase	IPR027417	-	0.0	-

Source	DB - Species	ID	Description	E-value	Identity %
FL-Next	*Tair_Athaliana_database*		| Symbols: AMK2 | adenosine monophosphate kinase | chr5:19375488-19378058 FORWARD LENGTH=283	0.0	75
FL-Next	*RefSeq_Athaliana_database*		adenosine monophosphate kinase [Arabidopsis thaliana]	0.0	75

ORF Prediction

Status: Complete
Subject seq: Q9FIJ7
Warning info: Selected protein is the only available option, Selected protein is the only available option,
Testcode index: 0

Test Code was used to find complete genes when there was not found a reliable orthologue. The best ORF (Open Reading Frame) is shown, and only ORFs > 200pb were analyzed. Your ORF will be more reliable if a stop codon was found before the start codon. A Test Code value > 0.95 means the ORF is probably coding. A Test Code value < 0.74 means the ORF is probably non-coding. Test Code values in between 0.74 and 0.95 mean it is uncertain whether the ORF is coding or not.

Predicted protein: MACSGCYSVNFMAVSSNPNKPLTSSTLLTSKLSQSYSQIPFSSLKFNSKLSPIQLRISKSAPNSFVVVSGIKAEPLKVMISGAPASGKGTQCELITKKYDLVHIAAGDLLRAEVAAGTENGRRAKEYMEKGQLVPDEIVVTMVKERLSQPDSKEKGWLMDGYPRSSSQATALKGFGFEPHIFILLEVPEEILVDRVVGRRLDPVTGKIYHLKYSPPETEEIAARLTQRFDDTEEKVKLRLLTHNKNVEAVLSMYEDVTVRVDGSLSKEDVFTQIDTALRNVLGQRQAPIGSVAM*

Length: 295

Coding sequence: GCGTAGATGCAAA_-_ATGGCTTGTAGCGGTTGCTATTCAGTGAATTTCATGGCAGTATCCTCCAATCCAAACAAGCCCTTAACTTCATCTACTCTCCTCACTTCAAAATTATCTCAATCCTA
CTCGCAAATCCCCTTTTCCTCTTTGAAATTCAATTCGAAATTATCCCCAATCCAGTTGCGTATATCAAAGTCTGCTCCAAATTCATTCGTTGTGGTATCTGGGATCAAGGCAGAGCCGTTGAAGGTAATGATATCGG
GAGCTCCTGCTTCTGGTAAAGGAACCCAATGCGAGCTCATCACTAAGAAGTACGATTTGGTGCACATTGCAGCTGGAGATCTTCTCAGGGCTGAAGTTGCTGCTGGGACAGAAAATGGGAGACGAGCGAAGGA
ATACATGGAGAAGGGGCAGTTGGTCCCAGATGAAATAGTTGTCACGATGGTCAAGGAGCGTTTATCACAACCAGATTCTAAAGAAAAAGGCTGGCTTATGGACGGATATCCAAGGAGCTCGTCACAAGCGACTG
CTCTTAAAGGATTTGGCTTCGAGCCACACATTTTCATTCTTCTTGAAGTCCCTGAAGAGATCCTTGTTGACAGAGTTGTTGGCCGCAGACTAGATCCTGTTACGGGGAAAATATATCATCTAAAGTATTCTCCCCCT
GAAACTGAAGAGATTGCTGCTAGGCTTACTCAACGCTTTGATGACACTGAAGAAAAGGTGAAACTGCGTTTGCTCACACATAACAAAAATGTGGAGGCCGTGCTTTCAATGTATGAGGATGTAACAGTCAGGGTG
GATGGAAGTCTTTCCAAGGAGGACGTATTTACCCAAATCGACACAGCGTTGAGAAATGTACTTGGACAAAGGCAGGCTCCCATAGGATCAGTGGCAATGTGA___AAACATCAAAGAAATCAATAAGAAAAAGGC
ATATCGACTGCCTAGACCGACTGCCACCTCTATTCGTGCTGATCCCACACCCTGAGTTGATCCGCAAATGGGCTCATCGGTGTTCCAAGGCCGACACGGACACTGCTGGCTGGTTTGAAGCTGCTCTTTCCCAC

Alignment:

```
olive_transcript_022501 RISKS----APNSFVVVSGIKAEPLKVMISGAPASGKGTQCELITKKYDLVHIAAGDLLRAEVAAGTENGRRAKEYMEKG(
Q9FIJ7                  RVSRSPSIIAPK-FQIVAAEKSEPLKIMISGAPASGKGTQCELITHKYGLVHISAGDLLRAEIASGSENGRRAKEHMEKG(
```

Expression

Class value	Stigma	Pollen	Vegetative
raw	11	3	-
rpkm	0.024	0.008	-

Back

Listing 100624 Descriptions A

« Previous 1 **2** 3 4 5 6 7 8 9 ... 5031 5032 Next »

Name	Source
1-O-acylglucose:anthocyanin-O-acyltransferase-like protein	sma3
1-phosphatidylinositol phosphodiesterase-related protein	sma3
1-phosphatidylinositol-3-phosphate 5-kinase	sma3
1-phosphatidylinositol-3-phosphate 5-kinase FAB1	sma3
1-phosphatidylinositol-3-phosphate 5-kinase fab1	sma3
1-pyrroline-5-carboxylate dehydrogenase	sma3
1,3-alpha-D-xylosyltransferase	sma3
1,3-beta-glucan synthase	sma3
1,4-alpha-glucan-branching enzyme, chloroplastic/amyloplastic	sma3
1,4-dihydroxy-2-naphthoate polyprenyltransferase, chloroplastic	sma3
1.2.1.n2	sma3
10 kDa chaperonin	sma3
101 kDa heat shock protein	sma3
117M18_19	sma3
117M18_7	sma3
12-oxophytodienoate reductase	sma3
12-oxophytodienoate reductase 1	sma3
12-oxophytodienoate reductase 2	sma3
12-oxophytodienoate reductase 3	sma3
12-oxophytodienoate reductase opr, putative	sma3

« Previous 1 **2** 3 4 5 6 7 8 9 ... 5031 5032 Next »

Listing 150 KEGGs B

« Previous **1** 2 3 4 5 6 7 8 Next »

Code	Name	Present ECs	Total ECs	Coverage (%)
00230	Purine metabolism	47	162	29.01
00500	Starch and sucrose metabolism	38	77	49.35
00520	Amino sugar and nucleotide sugar metabolism	37	110	33.64
00330	Arginine and proline metabolism	33	104	31.73
00270	Cysteine and methionine metabolism	31	68	45.59
00260	Glycine, serine and threonine metabolism	30	76	39.47
00860	Porphyrin and chlorophyll metabolism	29	89	32.58
00380	Tryptophan metabolism	29	63	46.03
00240	Pyrimidine metabolism	28	123	22.76
00360	Phenylalanine metabolism	25	75	33.33
00620	Pyruvate metabolism	25	67	37.31
00051	Fructose and mannose metabolism	24	68	35.29
00564	Glycerophospholipid metabolism	24	68	35.29
00630	Glyoxylate and dicarboxylate metabolism	23	70	32.86
00900	Terpenoid backbone biosynthesis	23	48	47.92
00400	Phenylalanine, tyrosine and tryptophan biosynthesis	23	45	51.11
00250	Alanine, aspartate and glutamate metabolism	22	57	38.60
00350	Tyrosine metabolism	22	58	37.93
00010	Glycolysis / Gluconeogenesis	22	64	34.38
00940	Phenylpropanoid biosynthesis	21	31	67.74

« Previous **1** 2 3 4 5 6 7 8 Next »

Map Number: Map 00592
Map Name: alpha-Linolenic acid metabolism

Enzymes in alpha-Linolenic acid metabolism:

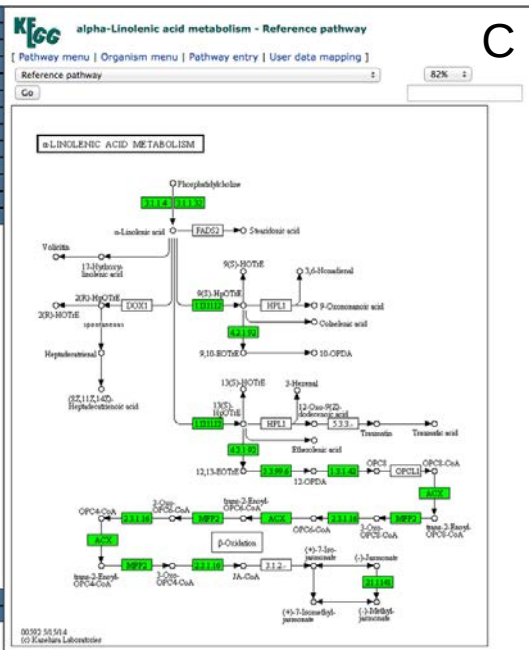
- 1.-.-.-,
- 1.13.11.12, Linoleate 13S-lipoxygenase.
- 1.14.19.-,
- 1.3.1.42, 12-oxophytodienoate reductase.
- 1.3.3.6, Acyl-CoA oxidase.
- 2.1.1.141, Jasmonate O-methyltransferase.
- 2.3.1.16, Acetyl-CoA C-acyltransferase.
- 3.1.1.32, Phospholipase A(1).
- 3.1.1.4, Phospholipase A(2).
- 3.1.2.-,
- 4.1.2.-,
- 4.2.1.17, Enoyl-CoA hydratase.
- 4.2.1.92, Hydroperoxide dehydratase.

EC Link: 4.2.1.92

Unigene	Assembly	Length	Full-Length
olive_transcript_018639	pollen-stigma-v1.0	1163 nt	Internal
olive_transcript_011195	pollen-stigma-v1.0	1450 nt	Putative C-terminus
olive_transcript_007935	pollen-stigma-v1.0	1645 nt	Complete
olive_transcript_046873	pollen-stigma-v1.0	700 nt	Internal
olive_transcript_051493	pollen-stigma-v1.0	662 nt	Internal
olive_transcript_059971	pollen-stigma-v1.0	612 nt	C_terminal Sure

- 5.3.99.6, Allene-oxide cyclase.
- 6.2.1.-,

C



Listing expression values D

raw
RPKM

« Previous **1** 2 3 4 5 6 7 8 9 ... 1539 1540 Next »

Transcript	Length (nt)	Stigma	Pollen	Vegetative
reproductive_tissues_transcriptome_v1.0	50337155	430165	373268	89403
olive_transcript_005400	140	1	-	-
olive_transcript_005401	1868	17	24	-
olive_transcript_005402	140	1	-	-
olive_transcript_005404	140	1	-	-
olive_transcript_005405	1868	15	-	2
olive_transcript_005406	140	1	-	-
olive_transcript_005415	1867	45	11	2
olive_transcript_005417	1867	11	26	4
olive_transcript_005418	142	1	-	-
olive_transcript_005423	1867	11	1	1
olive_transcript_005425	1866	8	31	1
olive_transcript_005426	140	1	-	-
olive_transcript_005427	1866	56	11	8
olive_transcript_005432	140	1	-	-
olive_transcript_005433	1865	-	96	-
olive_transcript_005434	140	1	-	-
olive_transcript_005436	140	-	1	-
olive_transcript_005437	1865	34	121	5
olive_transcript_005438	140	-	1	-
olive_transcript_005440	140	-	2	-

« Previous **1** 2 3 4 5 6 7 8 9 ... 1539 1540 Next »

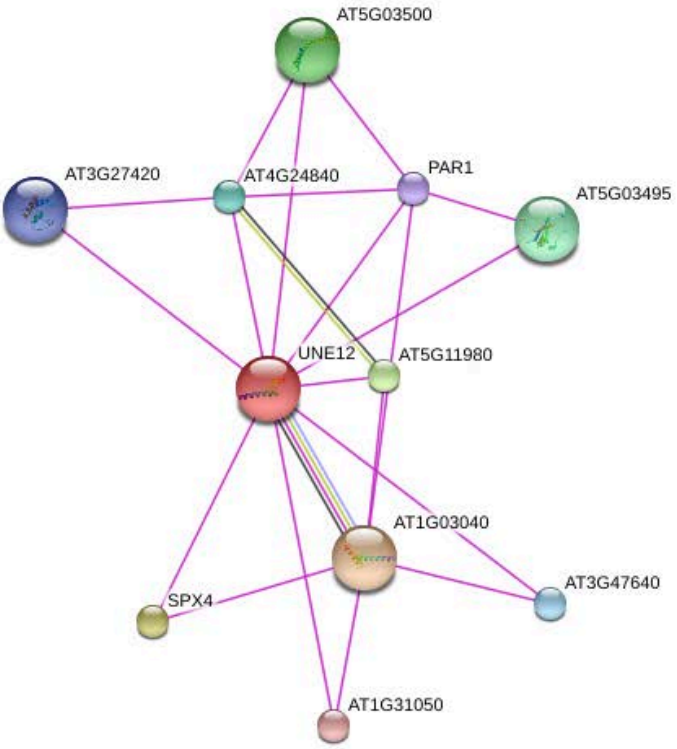
Listing expression values E

raw
RPKM

« Previous 1 2 3 4 5 6 7 **8** 9 10 11 12 ... 1539 1540 Next »

Transcript	Length (nt)	Stigma	Pollen	Vegetative
reproductive_tissues_transcriptome_v1.0	50337155	0.009	0.007	0.002
olive_transcript_005688	144	-	0.037	-
olive_transcript_005691	1839	0.037	0.006	-
olive_transcript_005693	1839	0.032	0.026	0.024
olive_transcript_005695	1839	0.039	0.001	0.018
olive_transcript_005697	1839	0.011	0.004	0.006
olive_transcript_005699	1839	0.014	0.016	0.006
olive_transcript_005101	1902	0.031	0.028	0.024
olive_transcript_005103	1902	0.017	0.006	0.006
olive_transcript_005104	135	-	0.02	-
olive_transcript_005105	1902	0.023	0.011	-
olive_transcript_005107	1901	0.018	0.225	0.012
olive_transcript_005108	135	0.017	-	-
olive_transcript_005110	135	0.017	-	-
olive_transcript_005111	1901	0.024	-	0.024
olive_transcript_005112	135	0.017	-	-
olive_transcript_005113	1901	0.097	-	0.012
olive_transcript_005114	135	0.017	-	-
olive_transcript_005117	1900	0.022	-	0.012
olive_transcript_005119	1900	0.009	0.055	-
olive_transcript_005122	135	0.017	-	-

« Previous 1 2 3 4 5 6 7 **8** 9 10 11 12 ... 1539 1540 Next »



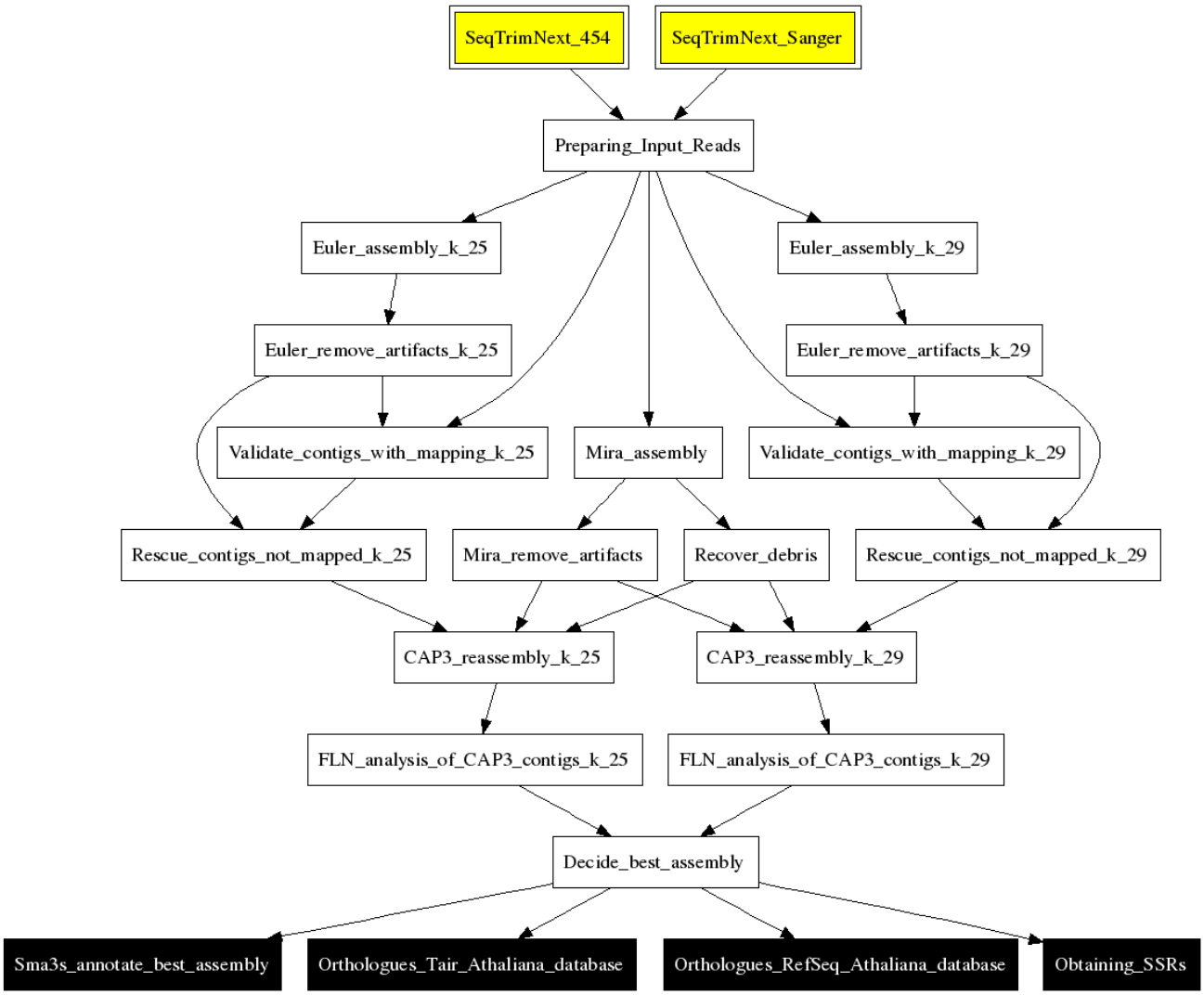
This is the **evidence view**. Different line colors represent the types of evidence for the association.

Your Input:

		Neighborhood	Gene Fusion	Cooccurrence	Coexpression	Experiments	Databases	Textmining	[Homology]	Score
UNE12	UNE12 (unfertilized embryo sac 12); DNA binding / transcription factor; unfertilized embryo sac 12 (UNE12); FUNCTIONS IN- transcription factor activity, DNA binding; INVOLVED IN- double fertilization forming a zygote and endosperm, regulation of transcription; LOCATED IN- nucleus; EXPRESSED IN- 22 plant structures; EXPRESSED DURING- 13 growth stages; CONTAINS InterPro DOMAIN/s- Basic helix-loop-helix dimerisation region bHLH (InterPro-IPR001092), Helix-loop-helix DNA-binding (InterPro-IPR011598); BEST Arabidopsis thaliana protein match is- basic helix-loop-helix (bHLH) family protein ([...] (310 aa) *(Arabidopsis thaliana)*									

Predicted Functional Partners:

AT1G03040	basic helix-loop-helix (bHLH) family protein; basic helix-loop-helix (bHLH) family protein; FUN [...] (302 aa)				•	•		•	•	0.946
SPX4	SPX4 (SPX DOMAIN GENE 4); SPX DOMAIN GENE 4 (SPX4); FUNCTIONS IN- molecular_function unknown; I [...] (318 aa)					•				0.923
AT5G11980	conserved oligomeric Golgi complex component-related / COG complex component-related; conserved [...] (569 aa)					•				0.923
AT5G03500	transcription coactivator; transcription coactivator; FUNCTIONS IN- transcription coactivator a [...] (168 aa)					•				0.923
AT5G03495	nucleotide binding; nucleotide binding; FUNCTIONS IN- nucleotide binding; INVOLVED IN- biologic [...] (226 aa)					•				0.923
AT4G24840	unknown protein; FUNCTIONS IN- molecular_function unknown; INVOLVED IN- protein transport, Golg [...] (756 aa)					•				0.923
AT3G47640	basic helix-loop-helix (bHLH) family protein; basic helix-loop-helix (bHLH) family protein; FUN [...] (240 aa)					•				0.923
AT3G27420	unknown protein; unknown protein; FUNCTIONS IN- molecular_function unknown; INVOLVED IN- biolog [...] (244 aa)					•				0.923
PAR1	PAR1 (PHY RAPIDLY REGULATED 1); Encodes PHYTOCHROME RAPIDLY REGULATED1 (PAR1), an atypical basi [...] (118 aa)					•				0.923
AT1G31050	transcription factor; transcription factor; FUNCTIONS IN- transcription factor activity; INVOLV [...] (268 aa)					•				0.923



Supplementary File 1: Flow diagram of the strategy for pre-processing, assembling and annotation of the transcriptomes described in this manuscript where yellow, double-lined boxes are the inputs, and the black boxes are the output results

```
#########################################################################################
########################################
# Declaring variables
#########################################################################################
########################################

# static variables
$original_454_reads=../../454_reads.fastq
$original_sanger_reads=../../Sanger_reads.fastq
$assembly_name=reproductive
$transcript_name=rp11_olive_
$db_orthologues_path=../../
$db_orthologues=[Tair_Athaliana_database;RefSeq_Athaliana_database]

# dynamic variable where the best assembly path will be stored
@best_assembly=NULL

#########################################################################################
########################################
# Pre-processing 454/Roche reads using SeqTrimNext
#########################################################################################
########################################
SeqTrimNext_454){
    module load seqtrimnext/last
    ?
    seqtrimnext -t transcriptomics_454_plants.txt -Q $original_454_reads -w [lcpu]
-s 10.243 > stn_454_reads.txt
    cat output_files/RL*/sequences*.fastq > input_mira_in.454.fastq
}

#########################################################################################
########################################
# Pre-processing Sanger reads using SeqTrimNext
#########################################################################################
########################################
SeqTrimNext_Sanger){
    module load seqtrimnext/last
    ?
    seqtrimnext -t sanger.txt -Q $original_sanger_reads -w [lcpu] -s 10.243 >
stn_sanger_reads.txt
    mv output_files/sequences_.fastq input_mira_in.sanger.fastq
}

#########################################################################################
########################################
# Preparing reads for assembling
#########################################################################################
########################################
```

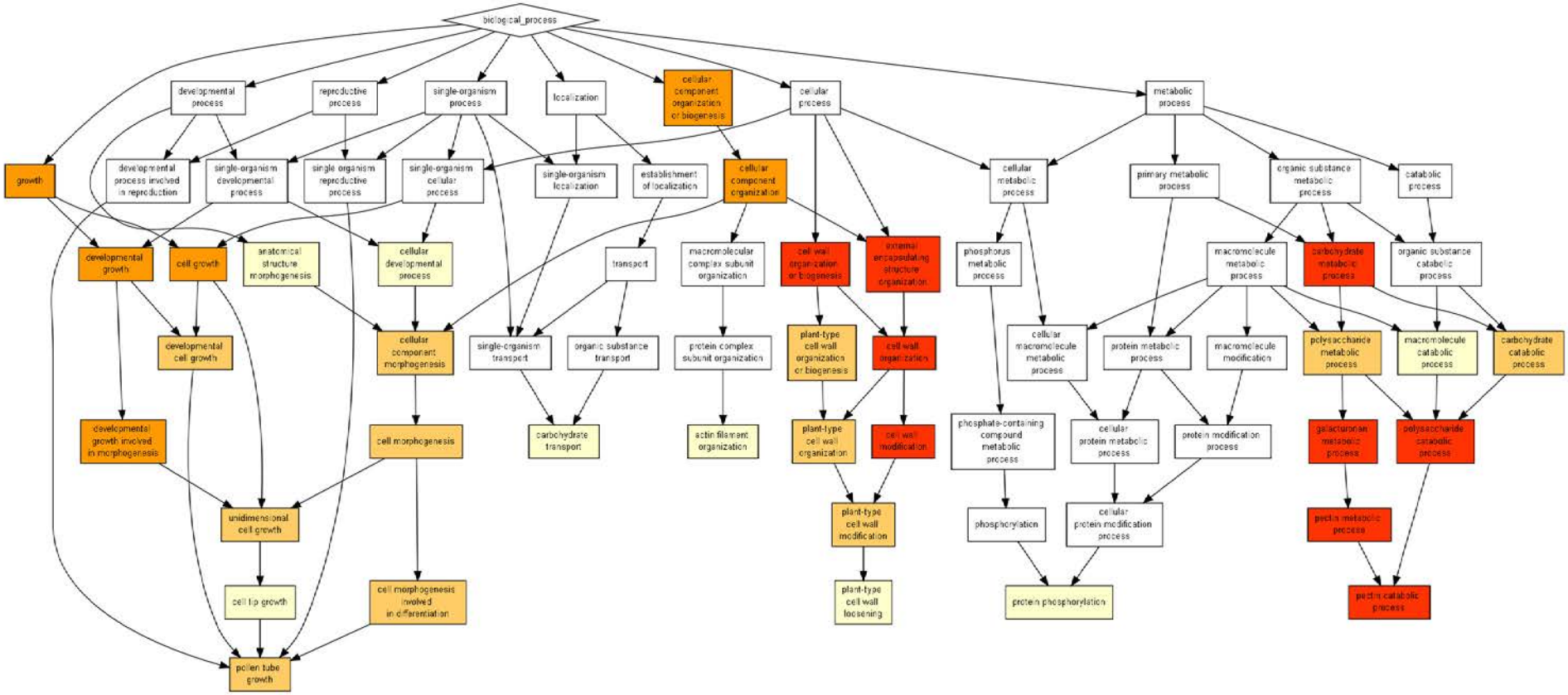
```
Preparing_Input_Reads){
    module load seqtrimnext/last
    ln -s SeqTrimNext_454)/input_mira_in.454.fastq
    ln -s SeqTrimNext_Sanger)/input_mira_in.sanger.fastq
    ?
    fastq2fasta.rb SeqTrimNext_454)/input_mira_in.454.fastq input_euler_454
    fastq2fasta.rb SeqTrimNext_Sanger)/input_mira_in.sanger.fastq input_euler_sanger
    cat input_euler_454.fasta input_euler_sanger.fasta > input_euler.fasta
}

#########################################################################################
#######################################
#Assembling with MIRA3 (OLC assembler)
#########################################################################################
#######################################

Mira_assembly){
    module load mira/3.2.0
    ln -s Preparing_Input_Reads)/input_mira_in.454.fastq
    ln -s Preparing_Input_Reads)/input_mira_in.sanger.fastq
    ln -s Preparing_Input_Reads)/input_euler.fasta
    mkdir -p $SCRATCH/$assembly_name
    ?
    mira -fastq -project=input_mira --job=denovo,est,normal,454,sanger -CL:ascdc
SANGER_SETTINGS -CO:fnicpst=yes  454_SETTINGS  -CO:fnicpst=yes -notraceinfo
COMMON_SETTINGS -GE:not=4 -DI:lrt=$SCRATCH/$assembly
    rm -rf $SCRATCH/$assembly_name

}

 # Remove artifacts from MIRA3
Mira_remove_artifacts){
    source ~soft_cvi_114/initializes/init_fln
    gem list full_lengther_next
    sort_fasta_list.rb
Mira_assembly)/input_mira_assembly/input_mira_d_results/input_mira_out.unpadded.fas
a
    lista_to_fasta.rb
Mira_assembly)/input_mira_assembly/input_mira_d_results/input_mira_out.unpadded.fas
a list > mira_contigs_dispersed.fasta
    ?
    full_lengther_next -f mira_contigs_dispersed.fasta -g plants -c 500 -z -w [lcpu]
}

 # Recover sequences from debris of MIRA3
Recover_debris){
    source ~soft_cvi_114/initializes/init_fln
    lista_to_fasta.rb Mira_assembly)/input_euler.fasta
```

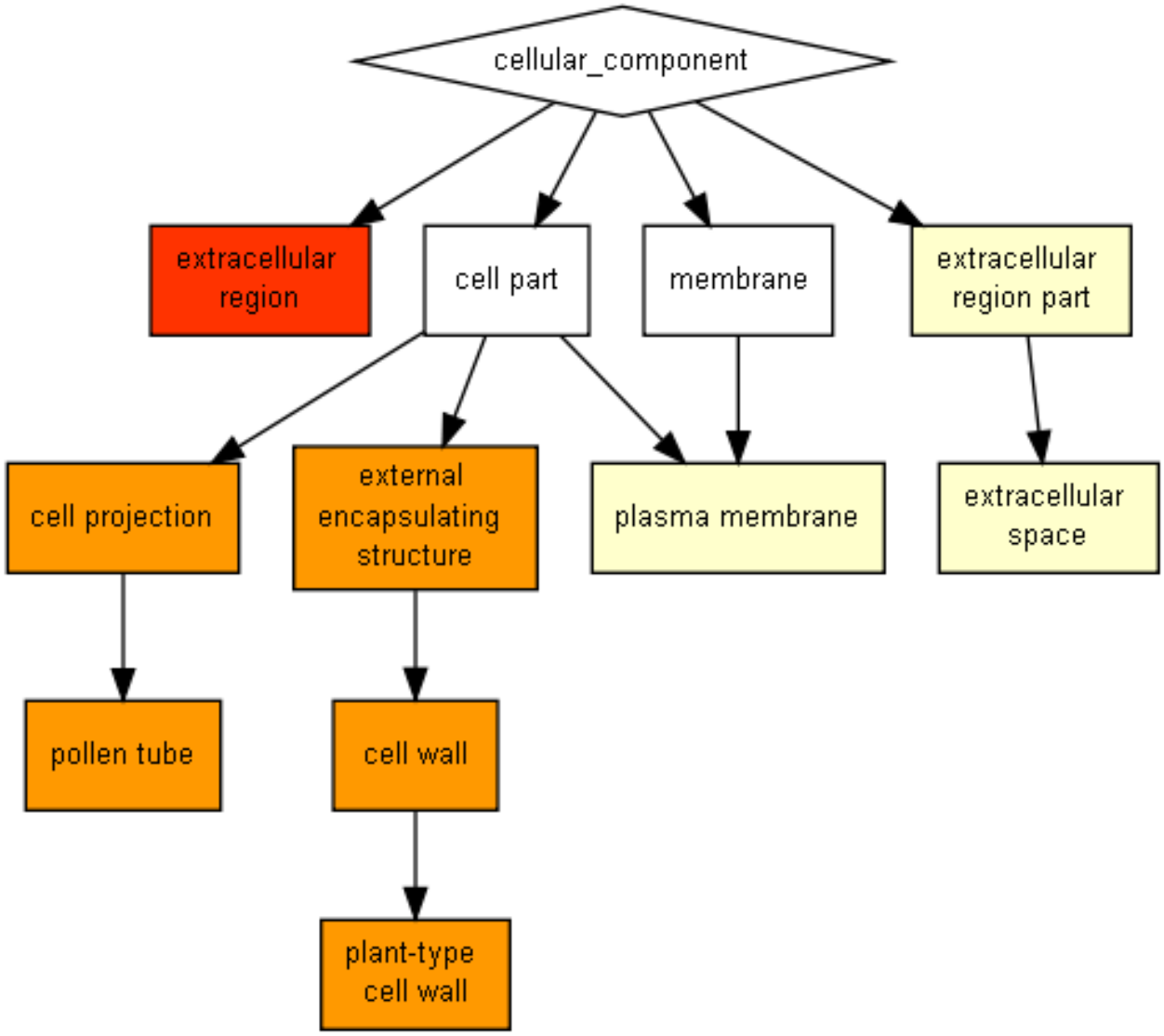
```
Mira_assembly)/input_mira_assembly/input_mira_d_info/input_mira_info_debrislist.txt
> mira_debris.fasta
    sort_fasta_list.rb mira_debris.fasta
    lista_to_fasta.rb mira_debris.fasta list > mira_debris_dispersed.fasta
    ?
    full_lengther_next -f mira_debris_dispersed.fasta -g plants -c 500 -z -w [lcpu]
-q d
    table_header.rb -t fln_results/pt_seqs n > coding_debris_mira.list
    table_header.rb -t fln_results/new_coding.txt >> coding_debris_mira.list
    lista_to_fasta.rb mira_debris_dispersed.fasta coding_debris_mira.list >
coding_debris_mira.fasta
}

##########################################################################################
########################################
# Assembling with EULER (de Bruijn assembling using 25 & 29 k-mers)
##########################################################################################
########################################
Euler_assembly_k_[25;29]){
    module load euler/120408
    ln -s Preparing_Input_Reads)/input_euler.fasta
    ?
    Assemble.pl input_euler.fasta (*)
}

# Remove artifacts from EULER
Euler_remove_artifacts_k_[25;29]){
    source ~soft_cvi_114/initializes/init_fln
    sort_fasta_list.rb !Euler_assembly_k_*!/input_euler.fasta.contig
    lista_to_fasta.rb !Euler_assembly_k_*!/input_euler.fasta.contig list >
dispersed_input_euler.fasta
    ?
    full_lengther_next -f dispersed_input_euler.fasta -g plants -c 500 -z -w [lcpu]
}


Validate_contigs_with_mapping_k_[25;29]){
    module load bowtie/v2_2.0.0-beta7
    bowtie2-build -f !Euler_remove_artifacts_k_*!/fln_results/unigenes.fasta ref
    ?
    bowtie2 ref -f -U Preparing_Input_Reads)/input_euler.fasta -p [cpu] --very-fast
-S mapeo.sam
}


Rescue_contigs_not_mapped_k_[25;29]){
    module load samtools/0.1.16
    ?
```

```
    mapping_tool.rb -i !Validate_contigs_with_mapping_k_*!/mapeo.sam -f -o
euler_mapped_contigs.list
    mapping_tool.rb -i !Validate_contigs_with_mapping_k_*!/mapeo.sam -f -r -o
euler_unmapped_contigs.list
    lista_to_fasta.rb !Euler_remove_artifacts_k_*!/fln_results/unigenes.fasta
euler_mapped_contigs.list > euler_mapped_contigs.fasta
    lista_to_fasta.rb !Euler_remove_artifacts_k_*!/fln_results/unigenes.fasta
euler_unmapped_contigs.list > temp.fasta
    table_header.rb -t !Euler_remove_artifacts_k_*!/fln_results/pt_seqs >
annot_coding_euler_unmapped.list
    table_header.rb -t !Euler_remove_artifacts_k_*!/fln_results/new_coding.txt >>
annot_coding_euler_unmapped.list
    lista_to_fasta.rb temp.fasta annot_coding_euler_unmapped.list >
euler_coding_unmapped_contigs.fasta
}

#############################################################################
########################################
## CAP3 (final reconciliation of MIRA3 contigs with each Euler-25 and Euler-29
contigs)
#############################################################################
########################################

CAP3_reassembly_k_[25;29]){
    module load cap3/101507
    ?
    cat Recover_debris)/coding_debris_mira.fasta
Mira_remove_artifacts)/fln_results/unigenes.fasta
!Rescue_contigs_not_mapped_k_*!/euler_mapped_contigs.fasta
!Rescue_contigs_not_mapped_k_*!/euler_coding_unmapped_contigs.fasta  >
reassembly.fasta
    cap3 reassembly.fasta -p 95 -o 40
}

FLN_analysis_of_CAP3_contigs_k_[25;29]){
   module load ruby
   source ~soft_cvi_114/initializes/init_fln
   cat !CAP3_reassembly_k_*!/reassembly.fasta.cap.contigs
!CAP3_reassembly_k_*!/reassembly.fasta.cap.singlets > unigenes.fasta
   sort_fasta_list.rb unigenes.fasta
   lista_to_fasta.rb unigenes.fasta list > dispersed_unigenes.fasta
   fasta_standard_renamer.rb dispersed_unigenes.fasta $transcript_name 6
   ?
   full_lengther_next -f dispersed_unigenes.fasta_new -g plants -c 300 -z -r -w
[]cpu]
}

#############################################################################
```

```
########################################
## Deciding the best assembly
################################################################################
########################################

Decide_best_assembly){
    module load ruby
    assembly=`fln_assembly_evaluator !FLN_analysis_of_CAP3_contigs_k_! `
    ?
    echo $assembly
    env_manager "best_assembly=$assembly;"
}

################################################################################
########################################
## Annotating the best assembly with sma3s
################################################################################
########################################

Sma3s_annotate_best_assembly){
    . ~soft_cvi_114/initializes/init_distributed_sma3s
    ?
    distributed_sma3s -c 'sma3s_v2.pl -a 123 -p F -v 2' -i
$best_assembly/fln_results/unigenes.fasta -o olive.annot -d
/mnt/home/soft/blast_db/current/fmt/sma3/uniprot_plants.dat -g 250 -w [lcpu]
}

################################################################################
########################################
# Obtaining ORTHOLOGUES in others Arabidopsis thaliana databases
################################################################################
########################################
Orthologues_$db_orthologues){
    source ~soft_cvi_114/initializes/init_fln
    echo Decide_best_assembly)
    mkdir db
    makeblastdb -in $db_orthologues_path/(*) -dbtype prot -parse_seqids -out db/(*)
    ?
    full_lengther_next -f $best_assembly/fln_results/unigenes.fasta -a '' -u db/(*)
-c 500 -z -w [lcpu] -q d -g plants
}

################################################################################
########################################
# Obtaining SSRS (microsatellites)
################################################################################
########################################
Obtaining_SSRs){
```

```
    module load ruby
    module load mreps/2.5
    echo Decide_best_assembly)
    ln -s $best_assembly/fln_results/unigenes.fasta
    ?
    replace_n.rb unigenes.fasta
    rm_codigo_degenerado.rb replaced_n_unigenes.fasta
    mreps -minsize 12 -minperiod 2 -exp 3.0 -fasta new_replaced_n_unigenes.fasta >
$assembly_name_ssr.txt
}

##############################################################################
########################################
```


biological_process
developmental process
reproductive process
single-organism process
localization
cellular component organization or biogenesis
cellular process
metabolic process
growth
developmental process involved in reproduction
single-organism developmental process
single organism reproductive process
single-organism cellular process
single-organism localization
establishment of localization
cellular component organization
cellular metabolic process
primary metabolic process
organic substance metabolic process
catabolic process
developmental growth
cell growth
anatomical structure morphogenesis
cellular developmental process
transport
macromolecular complex subunit organization
cell wall organization or biogenesis
external encapsulating structure organization
phosphorus metabolic process
macromolecule metabolic process
carbohydrate metabolic process
organic substance catabolic process
developmental cell growth
cellular component morphogenesis
single-organism transport
organic substance transport
protein complex subunit organization
plant-type cell wall organization or biogenesis
cell wall organization
cellular macromolecule metabolic process
protein metabolic process
macromolecule modification
polysaccharide metabolic process
macromolecule catabolic process
carbohydrate catabolic process
developmental growth involved in morphogenesis
cell morphogenesis
carbohydrate transport
actin filament organization
plant-type cell wall organization
cell wall modification
phosphate-containing compound metabolic process
cellular protein metabolic process
protein modification process
galacturonan metabolic process
polysaccharide catabolic process
unidimensional cell growth
plant-type cell wall modification
phosphorylation
cellular protein modification process
pectin metabolic process
cell tip growth
cell morphogenesis involved in differentiation
plant-type cell wall loosening
protein phosphorylation
pectin catabolic process
pollen tube growth



Parte VI

DISCUSIÓN

Capítulo 14

Discusión final

Con el desarrollo de esta tesis hemos procurado disminuir un poco más la distancia existente entre la biología de laboratorio y la bioinformática. Para ello hemos creado nuevas herramientas, *frameworks* y protocolos de uso que reducen los conocimientos necesarios para que usuarios noveles en el mundo de la bioinformática puedan aprovechar los recursos de supercomputación que se están haciendo tan necesarios en las investigaciones genómicas de hoy día. A continuación vamos a debatir el grado de cumplimiento de los objetivos establecidos al comienzo de este trabajo (capítulo 4).

14.1. Facilitar acceso a los recursos de supercomputación

Para cumplir este objetivo hemos desarrollado infraestructuras informáticas y herramientas que se describen en la parte II. Así, en el capítulo 5 describimos un sistema de escritorios virtuales para acceder de forma remota a una serie de recursos «listos para usar». De esta forma se ha puesto la supercomputación al servicio de los usuarios de bioinformática sin que estos sean conscientes de la complejidad que subyace al simple uso de una ventana de Windows® para ejecutar y gestionar los mismos datos que se pueden usar mediante la terminal de Picasso (el supercomputador de la UMA). Esto se ha conseguido mediante la integración de servidores de almacenamiento compartido, servidores de autentificación y un *cluster* de hipervisores. Además, hemos desarrollado un gestor que analiza la disponibilidad de las máquinas virtuales y optimiza el uso de los recursos sin intervención del usuario ni de los administradores del sistema. Este entorno se está utilizando en producción en la Plataforma Andaluza de Bioinformática de manera ininterrumpida desde el año 2008.

La facilidad de uso de un software complejo es difícil de alcanzar sin una interfaz de usuario visual y agradable. Dado que muchos programas disponibles para los análisis bioinformáticos sólo se puede ejecutar por línea de comandos —muy útiles y prácticos para la supercomputación, pero inabordables para muchos usuarios que se enfrentan por primera vez a este tipo de entornos—, en el capítulo 6 explicamos el desarrollo de un generador automático de interfaces de usuario para programas de líneas de comandos que oculta las complejidades de uso de un terminal de texto y del sistema de colas de un supercomputador detrás de una sencilla interfaz web. Este generador ha servido para proporcionar un portal web con programas desarrollados por nuestro grupo de investigación y por terceros (`http://www.scbi.uma.es/ingebiol`). El grupo del doctor Horacio Pérez Sánchez de la

UCAM ha mostrado un gran interés por él para distribuir máquinas virtuales con los algoritmos que desarrollan en su grupo de investigación, y esperamos que otros entornos también le encuentren uso.

Con la aportación de las dos herramientas que acabamos de discutir, podemos concluir que este objetivo ha sido cubierto, ya que hemos montado un autoservicio de bioinformática que acerca la supercomputación a los despachos y laboratorios de los usuarios mediante el uso de escritorios remotos y entornos web.

14.2. Aprovechamiento de recursos informáticos

Para cumplir este objetivo hemos desarrollado dos algoritmos en la parte III para paralelizar herramientas y para comprimir los datos de ultrasecuenciación, puesto que el aprovechamiento de los recursos de un centro de supercomputación exige la posibilidad de realizar ejecuciones en paralelo de los programas de análisis que necesitemos, tanto nuevos como ya existentes y no malgastar el espacio de almacenamiento. El *framework* de ejecución paralela y distribuida SCBI_MapReduce basado en granjas de tareas descrito en el capítulo 7 no requiere muchos conocimientos de paralelización, ya que permite crear los programas paralelos como varios programas lineales y el propio *framework* se encarga de distribuir el trabajo entre diferentes máquinas para obtener el paralelismo. Este *framework* además ha sido utilizado para crear diferentes herramientas como SeqTrimNEXT, Full-Lengthernext o GENote β.1 (descritas en la parte IV de esta tesis), y también se ha utilizado para crear *wrappers* para programas compilados que solo usan un núcleo de procesador, como la gema scbi_distributed_blast, que permite la ejecución distribuida de Blast+. Hemos comprobado que SCBI_MapReduce ha sido descargado más de 7.000 veces entre 2011 y 2015.

Las nuevas técnicas de ultrasecuenciación producen tal volumen de datos que es necesario pensar en el mejor aprovechamiento de los recursos de almacenamiento disponibles. Para ello hemos creado un nuevo formato de almacenamiento de secuencias comprimidas llamado FQBin que se describe en el capítulo 8. Se trata de un formato binario compatible con los formatos de texto más habituales (fasta y fastq) que ofrece un acceso aleatorio a las secuencias de manera muy eficaz. A la vez que se obtiene una mejora en el uso del espacio de almacenamiento, también se acelera la transferencia de ficheros por la red, dado que la cantidad de datos a transferir para los trabajos es menor. Esta gema de Ruby sido descargada 1500 veces desde el año 2013 al 2015.

Por lo tanto, podemos concluir que hemos cumplido este objetivo al haber diseñado herramientas que optimizan el uso de los recursos de supercomputación y almacenamiento de la UMA.

14.3. Nuevos algoritmos para biología

A pesar de la gran cantidad de programas que existen para resolver problemas biológicos, siempre podemos encontrar problemas que no cuentan con una herramienta adecuada que ayude a resolverlos. Siguiendo esta motivación, en la parte IV de esta tesis se reúnen todas las herramientas en cuyo desarrollo he intervenido en mayor o menor grado. En primer lugar nos encontramos con la necesidad de desarrollar una herramienta para la búsqueda de regiones más divergentes en los alineamientos de secuencias, lo que desembocó en el desarrollo

de AlignMiner (capítulo 9) que además permite diseñar cebadores con los que distinguir por PCR secuencias muy parecidas. También se ha utilizado para obtener los posibles SNP de un alineamiento. Además de la propia utilidad de esta herramienta, nos sirvió como banco de pruebas para darnos cuenta de la necesidad de crear el generador automático de interfaces (InGeBIOL) que ya hemos tratado en el apartado anterior.

A medida que las técnicas de ultrasecuenciación se fueron extendiendo, nos encontramos con que la herramienta de preprocesamiento de secuencias que habíamos desarrollado para la secuenciación de tipo Sanger (Seqtrim) [58] no era capaz de solventar los nuevos artefactos que se encontraban en la ultrasecuenciación. Por eso desarrollamos desde cero SeqTrimNEXT (capítulo 10) que incluye el uso de SCBI_MapReduce y está basado en una arquitectura de plugins y plantillas de ejecución, que no solo permitiría ampliar sus capacidades y adaptarnos mejor a las nuevas tecnologías, sino que también facilitaban su inclusión en InGeBiol. SeqTrimNEXT se utiliza por sistema en la Plataforma Andaluza de Bioinformática y sabemos que también está instalada en otros centros de investigación, puesto que ha sido descargado más de 21.000 veces desde su aparición pública en el año 2011.

Los trabajos sobre transcriptómica de organismos no modelo, como olivo, pino, haba o lenguado, nos impedían utilizar las técnicas habituales de comprobación de ensamblajes con una referencia. Por eso desarrollamos FullLengtherNEXT (capítulo 11), que también se está usando en la PAB por norma para determinar el mejor ensamblaje de transcriptomas, darle una anotación preliminar que permita tener una visión general, obtener un transcriptoma de referencia e incluso anotarlo con los mejores ortólogos. Este programa acumula más de 6.000 descargas desde el año 2012 al 2015 y también hace uso de SCBI_MapReduce para realizar todo el trabajo de forma distribuida y obtener mejores tiempos de ejecución.

En la misma línea de trabajo sobre organismos no modelo, hemos aportado la herramienta GENote β.1 (capítulo 12), que permite encontrar que puede haber en una secuencia de un organismo sobre el que hay poca información disponible. Para ello se realizan una serie de anotaciones iterativas con diferentes bases de datos relacionadas con especies cercanas, que pueden contener genes "parecidos". De nuevo, también se ha utilizado SCBI_MapReduce para optimizar el tiempo de ejecución.

El desarrollo de las las herramientas anteriores nos permite concluir que hemos aportado nuevas soluciones bioinformáticas para resolver problemas biológicos relacionados principalmente con la ultrasecuenciación y la anotación de ensamblajes.

14.4. Difusión y almacenamiento ordenado de los resultados de investigación

Cualquier investigación carece de sentido si los resultados obtenidos no salen del laboratorio que los generó, bien por un exceso de celo, o bien porque no saben cómo compartirlos de forma ordenada y útil. La informática permite organizar la información y realizar búsquedas instantáneas en grandes cantidades de datos mediante la creación de bases de datos. En la parte V de esta tesis se describe cómo se creó y mejoró una estructura básica de una base de datos consultable a través de la web para almacenar y difundir los datos generados a

partir de ultrasecuenciación de transcriptomas (capítulo 13). Este sistema de presentación ha ido evolucionando a medida que lo hemos ido necesitando para diferentes proyectos:

- EuropineDB: primera versión de la base de datos del transcriptoma de *Pinus Pinaster*.
- SustainpineDB: la evolución de EuropineDB tras añadirle datos de ultrasecuenciación de más tipos de tejidos.
- SoleaDB: el transcriptoma de *Solea senegalensis* y *Solea solea*, dos especies de lenguado de gran interés económico en la piscicultura.
- ReprOlive: presenta el transcriptoma de tejido reproductivo de olivo, una pequeña muestra de vegetativo, y en la versión de producción también incluye el transcriptoma de la semilla.

También debemos considerar que la difusión incluye el que la comunidad científica tenga un acceso completo a los programas y *frameworks* desarrollados en esta tesis. Por eso el código fuente de todos los desarrollos están accesibles tanto desde la PAB (`http://www.scbi.uma.es/site/scbi/downloads`) como desde el repositorio de github (`https://github.com/dariogf`) como código abierto.

En conclusión, no solamente hemos ayudado a difundir los resultados obtenidos por varios grupos de investigación a través de bases de datos públicas para los transcriptomas de los organismos que estudian, sino que también difundimos los resultados de esta tesis a través de los repositorios públicos de programas, como GitHub, y el propio SCBI.

Parte VII

BIBLIOGRAFÍA

Bibliografía

[1] Mohamed Abouelhoda, Shadi Issa, and Moustafa Ghanem. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support, 2012.

[2] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel Virtualization Technology for Directed I/O. *Intel Technology Journal*, 10(03):16, 2006.

[3] Dennis Abts and John Kim. High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities, 2011.

[4] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization, 2006.

[5] Advanced Micro Devices Inc. AMD-V Nested Paging. 2008.

[6] Pankaj Agarwal and Kouros Owzar. Next Generation Distributed Computing for Cancer Research. *Cancer Informatics*, pages 97–109, 2015.

[7] Nishi Ahuja. Datacenter power savings through high ambient datacenter operation: CFD modeling study. In *Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, pages 104–107, 2012.

[8] Francis J. Alexander, Adolfy Hoisie, and Alexander Szalay. Big Data, 2011.

[9] William (Bill) Allcock, Evan Felix, Mike Lowe, Randal Rheinheimer, and Joshi Fullop. Challenges of HPC monitoring. *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–6, 2011.

[10] Rolf Apweiler, Amos Bairoch, Cathy H Wu, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, Maria J Martin, Darren A Natale, Claire O'Donovan, Nicole Redaschi, and Lai-Su L Yeh. UniProt: the Universal Protein knowledgebase. *Nucleic Acids Res*, 32(Database issue):D115–9, January 2004.

[11] A Asosheh and MH Danesh. Comparison of OS level and hypervisor server virtualization. In *Proceedings of the 8th conference on systems ...*, pages 241–246, 2008.

[12] Jean-Marc Aury, Corinne Cruaud, Valérie Barbe, Odile Rogier, Sophie Mangenot, Gaelle Samson, Julie Poulain, Véronique Anthouard, Claude Scarpelli, François Artiguenave, and Patrick Wincker. High quality draft sequences for prokaryotic genomes using a mix of new sequencing technologies. *BMC genomics*, 9:603, 2008.

[13] J. W. Backus and W. P. Heising. Fortran. *IEEE Transactions on Electronic Computers*, EC-13, 1964.

[14] A Bairoch and R Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic acids research*, 28:45–48, 2000.

[15] Henri E. Bal and Matthew Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, 6:74–84, 1998.

[16] Derek W. Barnett, Erik K. Garrison, Aaron R. Quinlan, Michael P. Stromberg, and Gabor T. Marth. Bamtools: A C++ API and toolkit for analyzing and managing BAM files. *Bioinformatics*, 27(12):1691–1692, 2011.

[17] L Barroso and Urs Hölzle. *The Datacenter as a Computer*, volume 24. 2013.

[18] Luiz André Barroso and Urs Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, 2009.

[19] Wolfgang Barth. *Nagios: System and Network Monitoring*. 2008.

[20] Sebastian Bassi and Virginia V C Gonzalez. DNALinux virtual desktop edition. 2007.

[21] Serafim Batzoglou, David B. Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P. Mesirov, and Eric S. Lander. ARACHNE: A whole-genome shotgun assembler. *Genome Research*, 12(1):177–189, 2002.

[22] Dennis A. Benson, Karen Clark, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and Eric W. Sayers. GenBank. *Nucleic Acids Research*, 42(D1), 2014.

[23] Hicham Benzekri, Paula Armesto, Xavier Cousin, Mireia Rovira, Diego Crespo, Manuel Merlo, David Mazurais, Rocío Bautista, Darío Guerrero-Fernández, Noe Fernandez-Pozo, Marian Ponce, Carlos Infante, Jose Zambonino, Sabine Nidelet, Marta Gut, Laureana Rebordinos, Josep V Planas, Marie-Laure Bégout, M Claros, and Manuel Manchado. De novo assembly, characterization and functional annotation of Senegalese sole (Solea senegalensis) and common sole (Solea solea) transcriptomes: integration in a database and design of a microarray. *BMC Genomics*, 15(1):952, 2014.

[24] Viktors Berstis. Fundamentals of grid computing, 2002.

[25] M. Bhandarkar. MapReduce programming with apache Hadoop. *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.

[26] Martin Biermann. A simple versatile solution for collecting multidimensional clinical data based on the CakePHP web application framework. *Computer Methods and Programs in Biomedicine*, 114(1):70–79, 2014.

[27] Bitvise. WinSSH, 2000.

[28] Christian Bizer, T Heath, and T Berners-Lee. Linked data-the story so far. *International journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[29] N Travinin Bliss, Robert Bond, Jeremy Kepner, Hahn Kim, and Albert Reuther. Interactive grid computing at Lincoln Laboratory. *Lincoln Laboratory Journal*, 16(1):165, 2006.

[30] Joseph F. Boland, Charles C. Chung, David Roberson, Jason Mitchell, Xijun Zhang, Kate M. Im, Ji He, Stephen J. Chanock, Meredith Yeager, and Michael Dean. The new sequencer on the block: Comparison of Life Technology's Proton sequencer to an Illumina HiSeq for whole-exome sequencing. *Human Genetics*, 132(10):1153–1163, 2013.

[31] Borys J. Bradel and Tarek S. Abdelrahman. Automatic trace-based parallelization of Java programs. In *Proceedings of the International Conference on Parallel Processing*, 2007.

[32] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod a Chapman, Guillaume Chapuis, Rayan Chikhi, Hamidreza Chitsaz, Wen-Chi Chou, Jacques Corbeil, Cristian Del Fabbro, T Roderick Docking, Richard Durbin, Dent Earl, Scott Emrich, Pavel Fedotov, Nuno a Fonseca, Ganeshkumar Ganapathy, Richard a Gibbs, Sante Gnerre, Elénie Godzaridis, Steve Goldstein, Matthias Haimel, Giles Hall, David Haussler, Joseph B Hiatt, Isaac Y Ho, Jason Howard, Martin Hunt, Shaun D Jackman, David B Jaffe, Erich D Jarvis, Huaiyang Jiang, Sergey Kazakov, Paul J Kersey, Jacob O Kitzman, James R Knight, Sergey Koren, Tak-Wah Lam, Dominique Lavenier, François Laviolette, Yingrui Li, Zhenyu Li, Binghang Liu, Yue Liu, Ruibang Luo, Iain Maccallum, Matthew D Macmanes, Nicolas Maillet, Sergey Melnikov, Delphine Naquin, Zemin Ning, Thomas D Otto, Benedict Paten, Octávio S Paulo, Adam M Phillippy, Francisco Pina-Martins, Michael Place, Dariusz Przybylski, Xiang Qin, Carson Qu, Filipe J Ribeiro, Stephen Richards, Daniel S Rokhsar, J Graham Ruby, Simone Scalabrin, Michael C Schatz, David C Schwartz, Alexey Sergushichev, Ted Sharpe, Timothy I Shaw, Jay Shendure, Yujian Shi, Jared T Simpson, Henry Song, Fedor Tsarev, Francesco Vezzi, Riccardo Vicedomini, Bruno M Vieira, Jun Wang, Kim C Worley, Shuangye Yin, Siu-Ming Yiu, Jianying Yuan, Guojie Zhang, Hao Zhang, Shiguo Zhou, and Ian F Korf. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):10, 2013.

[33] Kelli Bramlett, Jeff Schageman, Luming Qu, Kristi Lea, Joey Cienfuegos, Bob Setterquist, Life Technologies, and Woodward St. RNA-Seq Applications on the Ion Torrent PGM. In *Plant & Animal Genome XX*, number 512, 2012.

[34] P Bridges, N Doss, W Gropp, E Karrels, E Lusk, and A Skjellum. User's Guide to MPICH, a Portable Implementation of MPI. *Argonne National Laboratory*, 9700:60439–64801, 1995.

[35] Ken O Burtch. *Linux shell scripting with Bash*, volume 1. 2004.

[36] F Javier Cabañes, Walter Sanseverino, Gemma Castellá, M Rosa Bragulat, Riccardo Aiese Cigliano, and Armand Sánchez. Rapid genome resequencing of an atoxigenic strain of Aspergillus carbonarius, 2015.

[37] C Camacho, G Coulouris, V Avagyan, N Ma, J Papadopoulos, K Bealer, and T L Madden. BLAST+: architecture and applications. *BMC Bioinformatics*, 10:421, 2009.

[38] Davide Campagna, Alessandro Albiero, Alessandra Bilardi, Elisa Caniato, Claudio Forcato, Svetlin Manavski, Nicola Vitulo, and Giorgio Valle. PASS: A program to align short sequences. *Bioinformatics*, 25(7):967–968, 2009.

[39] Cinzia Cantacessi, Aaron R. Jex, Ross S. Hall, Neil D. Young, Bronwyn E. Campbell, Anja Joachim, Matthew J. Nolan, Sahar Abubucker, Paul W. Sternberg, Shoba Ranganathan, Makedonka Mitreva, and Robin B. Gasser. A practical, bioinformatic workflow system for large data sets generated by next-generation sequencing. *Nucleic Acids Research*, 38(17), 2010.

[40] Mark J. Chaisson and Pavel A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18(2):324–330, 2008.

[41] Jeffrey Chang. Core services: Reward bioinformaticians. *Nature*, 520(7546):151–152, 2015.

[42] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*, volume 10. 2008.

[43] Sanjay Chatterjee, Sagnak Tasirlar, Zoran Budimlić, Vincent Cavé, Milind Chabbi, Max Grossman, Vivek Sarkar, and Yonghong Yan. Integrating asynchronous task parallelism with MPI. In *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pages 712–725, 2013.

[44] Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.

[45] Manuel Gonzalo Claros, Rocío Bautista, Darío Guerrero-Fernández, Hicham Benzerki, Pedro Seoane, and Noé Fernández-Pozo. Why Assembling Plant Genome Sequences Is So Challenging, 2012.

[46] Phillip E C Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly, 2011.

[47] Adaptative Computing. Moab Workload Manager, 2006.

[48] A Conesa, S Gotz, J M Garcia-Gomez, J Terol, M Talon, and M Robles. Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research. *Bioinformatics*, 21(18):3674–3676, 2005.

[49] CyberDuck. CyberDuck, 2000.

[50] M. Danelutto. Adaptive task farm implementation strategies. In *Proceedings - Euromicro Conference on Parellel, Distribeted and Network-based Proceeding*, pages 416–423, 2004.

[51] Giovanna DÁngelo, Teresa Di Rienzo, and Veronica Ojetti. Microarray analysis in gastric cancer: a review. *World journal of gastroenterology : WJG*, 20(34):11972–11976, September 2014.

[52] Troy Dawson, Jarek Polok, Connie Sieh, and Others. Scientific Linux, 2005.

[53] John Deacon. Model-view-controller (mvc) architecture. *...de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*, pages 1–6, 2009.

[54] Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de

novo genomic sequencing. *Genome Research*, 17(11):1697–1706, 2007.

[55] Dent Earl, Keith Bradnam, John St. John, Aaron Darling, Dawei Lin, Joseph Fass, Hung On Ken Yu, Vince Buffalo, Daniel R. Zerbino, Mark Diekhans, Ngan Nguyen, Pramila Nuwantha Ariyaratne, Wing Kin Sung, Zemin Ning, Matthias Haimel, Jared T. Simpson, Nuno A. Fonseca, Inanç Birol, T. Roderick Docking, Isaac Y. Ho, Daniel S. Rokhsar, Rayan Chikhi, Dominique Lavenier, Guillaume Chapuis, Delphine Naquin, Nicolas Maillet, Michael C. Schatz, David R. Kelley, Adam M. Phillippy, Sergey Koren, Shiaw Pyng Yang, Wei Wu, Wen Chi Chou, Anuj Srivastava, Timothy I. Shaw, J. Graham Ruby, Peter Skewes-Cox, Miguel Betegon, Michelle T. Dimon, Victor Solovyev, Igor Seledtsov, Petr Kosarev, Denis Vorobyev, Ricardo Ramirez-Gonzalez, Richard Leggett, Dan MacLean, Fangfang Xia, Ruibang Luo, Zhenyu Li, Yinlong Xie, Binghang Liu, Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Ted Sharpe, Giles Hall, Paul J. Kersey, Richard Durbin, Shaun D. Jackman, Jarrod A. Chapman, Xiaoqiu Huang, Joseph L. DeRisi, Mario Caccamo, Yingrui Li, David B. Jaffe, Richard E. Green, David Haussler, Ian Korf, and Benedict Paten. Assemblathon 1: A competitive assessment of de novo short read assembly methods, 2011.

[56] Ron Edgar, Michael Domrachev, and Alex E Lash. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic acids research*, 30(1):207–210, 2002.

[57] Michael Eisenstein. Oxford Nanopore announcement sets sequencing sector abuzz, 2012.

[58] Juan Falgueras, Antonio J Lara, Noé Fernández-Pozo, Francisco R Cantón, Guillermo Pérez-Trabado, and M Gonzalo Claros. SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC bioinformatics*, 11:38, 2010.

[59] C S Ferreira, B S Vaz, G Velasco, R A Tavares, H Hellebrandt, and E H Albergone. Poseidon Linux 3. x-The Scientific GNU/Linux option. *Pan-Am. J. Aquat. Sci*, 4, 2009.

[60] Dawn Field, Bela Tiwari, Tim Booth, Stewart Houten, Dan Swan, Nicolas Bertrand, and Milo Thurston. Open software for biologists: from famine to feast. *Nature biotechnology*, 24(7):801–803, 2006.

[61] FileZilla. FileZilla, 2000.

[62] David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language*, volume 159. 2008.

[63] Egan Ford. *Building a Linux Hpc Cluster With Xcat*. IBM Press, 2002.

[64] Tammy Fox. Red Hat Enterprise Linux 5. *East*, 323:5–6, 2008.

[65] John L Furlani. Modules: Providing a flexible user environment. In *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pages 141–152, 1991.

[66] Roberto Gallopini. Open Source Monitoring: Icinga vs Nagios, 2010.

[67] David Garlan and Mary Shaw. An Introduction to Software Architecture. *Knowledge Creation Diffusion Utilization*, 1(January):1–40, 1994.

[68] Wolfgang Gentzsch. Sun Grid Engine: Towards creating a compute power grid. In *Proceedings - 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid 2001*, pages 35–36, 2001.

[69] Almas A. Gheyas and David W. Burt. Microarray resources for genetic and genomic studies in chicken: A review, 2013.

[70] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11:R86, 2010.

[71] Richard L. Graham, Galen M. Shipman, Brian W. Barrett, Ralph H. Castain, George Bosilca, and Andrew Lumsdaine. Open MPI: A high-performance, heterogeneous MPI. In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 2006.

[72] Feng Guangwu, Zhao Qingming, Xiao Qing, Wang Jun, and Others. Application of Team Viewer in Remote System Management. *Time Education*, 5:23, 2011.

[73] Longbiao Guo, Zhenyu Gao, and Qian Qian. Application of resequencing to rice genomics, functional genomics and evolutionary analysis. *Rice (New York, N.Y.)*, 7(1):4, 2014.

[74] David Heinemeier Hansson and Thomas Fuchs. *Agile Web Development with Rails*. 2005.

[75] M A Harris, J I Clark, A Ireland, J Lomax, M Ashburner, R Collins, K Eilbeck, S Lewis, C Mungall, J Richter, G M Rubin, S Q Shu, J A Blake, C J Bult, A D Diehl, M E Dolan, H J Drabkin, J T Eppig, D P Hill, L Ni, M Ringwald, R Balakrishnan, G Binkley, J M Cherry, K R Christie, M C Costanzo, Q Dong, S R Engel, D G Fisk, J E Hirschman, B C Hitz, E L Hong, C Lane, S Miyasato, R Nash, A Sethuraman, M Skrzypek, C L Theesfeld, S A Weng, D Botstein, K Dolinski, R Oughtred, T Berardini, S Mundodi, S Y Rhee, R Apweiler, D Barrell, E Camon, E Dimmer, N Mulder, R Chisholm, P Fey, P Gaudet, W Kibbe, K Pilcher, C A Bastiani, R Kishore, E M Schwarz, P Sternberg, K Van Auken, M Gwinn, L Hannick, J Wortman, M Aslett, M Berriman, V Wood, S Bromberg, C Foote, H Jacob, D Pasko, V Petri, D Reilly, K Seiler, M Shimoyama, J Smith, S Twigger, P Jaiswal, T Seigfried, C Collmer, D Howe, M Westerfield, and Gene Ontology Consortium. The Gene Ontology (GO) project in 2006. *Nucleic Acids Research*, 34:D322–D326, 2006.

[76] Carl Hewitt. Actor Model of Computation, 2010.

[77] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. 1999.

[78] Adrian Holovaty and Jacob Kaplan-Moss. *The Django Book*. 2007.

[79] Carson Holt and Mark Yandell. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. *BMC Bioinformatics*, 12(1):491, 2011.

[80] Homolog_us. Bioinformatics at the Core Facilities.

[81] Stefan Howorka and Zuzanna Siwy. Nanopore analytics: sensing of single

molecules. *Chemical Society reviews*, 38(8):2360–2384, 2009.

[82] Shih Cheng Hu and Chao Ching Chen. Locating the very early smoke detector apparatus (VESDA) in vertical laminar clean rooms according to the trajectories of smoke particles. *Building and Environment*, 42:366–371, 2007.

[83] W. Huang, G. Santhanaraman, H. W. Jin, Q. Gao, and D. K. Panda. Design of high performance MVAPICH2: MPI2 over InfiniBand. In *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06*, pages 43–46, 2006.

[84] Xiaojun Huang, Hanlin Lu, Jun-Wen Wang, Liqin Xu, Siyang Liu, Jihua Sun, and Fei Gao. High-throughput sequencing of methylated cytosine enriched by modification-dependent restriction endonuclease MspJI. *BMC genetics*, 14:56, 2013.

[85] Xiaoqiu Huang and Anup Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9(9):868–877, 1999.

[86] Xiaoqiu Huang, Jianmin Wang, Srinivas Aluru, Shiaw Pyng Yang, and LaDeana Hillier. PCAP: A whole-genome assembly program. *Genome Research*, 13(9):2164–2170, 2003.

[87] Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev. Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*, pages 563–573, 2011.

[88] Adam A Hunter, Andrew B Macgregor, Tamas O Szabo, Crispin A Wellington, and Matthew I Bellgard. Yabi: An online research environment for grid, high performance and cloud computing, 2012.

[89] Ross Ihaka and Robert Gentleman. R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

[90] David Jackson, Quinn Snell, and Mark Clement. Core Algorithms of the Maui Scheduler. *Job Scheduling Strategies for Parallel Processing*, 2221:87–102, 2001.

[91] William R. Jeck, Josephine A. Reinhardt, David A. Baltrus, Matthew T. Hickenbotham, Vincent Magrini, Elaine R. Mardis, Jeffery L. Dangl, and Corbin D. Jones. Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23(21):2942–2944, 2007.

[92] M Jette and M Grondona. SLURM: Simple Linux Utility for Resource Management. In *ClusterWorld Conference and Expo CWCE*, volume 2682, pages 44–60, 2003.

[93] K. Jiang, O. Thorsen, A. Peters, B. Smith, and C.P. Sosa. An Efficient Parallel Implementation of the Hidden Markov Methods for Genomic Sequence-Search on a Massively Parallel System. *IEEE Transactions on Parallel and Distributed Systems*, 19, 2008.

[94] V Kalusivalingam. Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6). *RFC*, 2004.

[95] Minoru Kanehisa, Susumu Goto, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Data, information, knowledge and principle: Back to

metabolism in KEGG. *Nucleic Acids Research*, 42(D1), 2014.

[96] Tarun Kant. Open source bioinformatics workbench options for life science researchers. *New York Science Journal*, 3(10):82–87, 2010.

[97] John Kelbley and Mike Sterling. *Windows server 2008 R2 Hyper-V : insiders guide to Microsoft's Hypervisor*. 2010.

[98] Brian W Kernighan and Dennis M Ritchie. *The C programming language*, volume 78. 1988.

[99] Daehwan Kim and Steven L Salzberg. TopHat-Fusion: an algorithm for discovery of novel fusion transcripts, 2011.

[100] Martin Kircher and Janet Kelso. High-throughput DNA sequencing–concepts and limitations. *BioEssays : news and reviews in molecular, cellular and developmental biology*, 32(6):524–536, 2010.

[101] Avi Kivity, Uri Lublin, Anthony Liguori, Yaniv Kamay, and Dor Laor. kvm: the Linux virtual machine monitor. *Proceedings of the Linux Symposium*, 1:225–230, 2007.

[102] Yuichi Kodama, Martin Shumway, and Rasko Leinonen. The sequence read archive: explosive growth of sequencing data. *Nucleic Acids Research*, 40(D1):D54–D56, 2012.

[103] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. SETI@home-massively distributed computing for SETI. *Computing in Science & Engineering*, 3, 2001.

[104] L B Koski, M W Gray, B F Lang, and G Burger. AutoFACT: an automatic functional annotation and classification tool. *BMC Bioinformatics*, 6:151, 2005.

[105] Kyong I. Ku, Sung Joo Kang, Moonyoung Chung, Won Young Kim, and Wan Choi. Evaluation and optimization of activeRecord to implement the personalized software service platform. In *International Conference on Advanced Communication Technology, ICACT*, volume 3, pages 1763–1766, 2008.

[106] Sujai Kumar and Mark L Blaxter. Comparing de novo assemblers for 454 transcriptome data. *BMC genomics*, 11:571, 2010.

[107] Ralf Lämmel. Google's MapReduce programming model - Revisited. *Science of Computer Programming*, 68:208–237, 2007.

[108] Ben Langmead. Aligning short sequencing reads with Bowtie. *Current Protocols in Bioinformatics*, (SUPP.32), 2010.

[109] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2, 2012.

[110] Rasko Leinonen, Hideaki Sugawara, and Martin Shumway. The sequence read archive. *Nucleic Acids Research*, 39(SUPPL. 1), 2011.

[111] Simone Leo and Gianluigi Zanetti. Pydoop : a Python MapReduce and HDFS API for Hadoop. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 819–825, 2010.

[112] Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre. *Programming PHP*, volume 37. 2006.

[113] N. Lévêque, F. Renois, and L. Andréoletti. The microarray technology: Facts and controversies, 2013.

[114] Fran Lewitter and Michael Rebhan. Establishing a successful bioinformatics Core Facility Team, 2009.

[115] Fran Lewitter, Michael Rebhan, Brent Richter, and David Sexton. The need for centralization of computational biology resources. *PLoS Computational Biology*, 5(6), 2009.

[116] Lexnederbragt. A hybrid model for a High-Performance Computing infrastructure for bioinformatics, 2015.

[117] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map (SAM) Format and SAMtools. *Bioinformatics*, 2009.

[118] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, 2008.

[119] Jiangtian Li, Xiaosong Ma, Srikanth Yoginath, Guruprasad Kora, and Nagiza F. Samatova. Transparent runtime parallelization of the R scripting language. *Journal of Parallel and Distributed Computing*, 71:157–168, 2011.

[120] Jianjiang Li, Jiwu Shu, Yongjian Chen, Dingxing Wang, and Weimin Zheng. Analysis of factors affecting execution performance of openMP programs. *Tsinghua Science and Technology*, 10:304–308, 2005.

[121] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. SOAP: Short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[122] Yang Li, Pegah Kamousi, Fangqiu Han, Shengqi Yang, Xifeng Yan, Subhash Suri, and Santa Barbara. Memory Efficient De Bruijn Graph Construction. *arXiv*, page 13, 2012.

[123] Yong Lin, Jian Li, Hui Shen, Lei Zhang, Christopher J Papasian, and Hong-Wen Deng. Comparative studies of de novo assembly tools for next-generation sequencing technologies. *Bioinformatics (Oxford, England)*, 27:2031–2037, 2011.

[124] Burkhard Linke, Robert Giegerich, and Alexander Goesmann. Conveyor: A workflow engine for bioinformatic analyses. *Bioinformatics*, 27(7):903–911, 2011.

[125] Kun Liu and Long-jiang Dong. Research on Cloud Data Storage Technology and Its Architecture Implementation, 2012.

[126] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems, 2012.

[127] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. Parallelized short read assembly of large genomes using de Bruijn graphs. *BMC bioinformatics*, 12:354, 2011.

[128] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward a. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18:1039–1065, 2006.

[129] Gerton Lunter and Martin Goodson. Stampy: A statistical algorithm for sensitive and fast mapping of Illumina sequence reads. *Genome Research*, 21(6):936–939, 2011.

[130] Ruibang Luo, Binghang Liu, Yinlong Xie, Zhenyu Li, Weihua Huang, Jianying Yuan, Guangzhu He, Yanxiang Chen, Qi Pan, Yunjie Liu, Jingbo Tang, Gengxiong Wu, Hao Zhang, Yujian Shi, Yong Liu, Chang Yu, Bo Wang, Yao Lu, Changlei Han, David W Cheung, Siu-Ming Yiu, Shaoliang Peng, Zhu Xiaoqian, Guangming Liu, Xiangke Liao, Yingrui Li, Huanming Yang, Jian Wang, Tak-Wah Lam, and Jun Wang. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1):18, 2012.

[131] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi Man Liu, Xiaoqian Zhu, Edward Wu, Lap Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W. Cheung, Hing Fung Ting, Siu Ming Yiu, Shaoliang Peng, Chang Yu, Yingrui Li, Ruiqiang Li, and Tak Wah Lam. SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner. *PLoS ONE*, 8(5), 2013.

[132] Carol M. Lushbough, Douglas M. Jennewein, and Volker P. Brendel. The BioExtract Server: A web-based bioinformatic workflow platform. *Nucleic Acids Research*, 39(SUPPL. 2), 2011.

[133] Iain Maccallum, Dariusz Przybylski, Sante Gnerre, Joshua Burton, Ilya Shlyakhter, Andreas Gnirke, Joel Malek, Kevin McKernan, Swati Ranade, Terrance P Shea, Louise Williams, Sarah Young, Chad Nusbaum, and David B Jaffe. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome biology*, 10(10):R103, 2009.

[134] Raj D. Maitra, Jungsuk Kim, and William B. Dunbar. Recent advances in nanopore sequencing, 2012.

[135] Lev Manovich. Trending: The Promises and the Challenges of Big Social Data. *Debates in the Digital Humanities*, pages 1–10, 2011.

[136] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30:817–840, 2004.

[137] Michael L Metzker. Sequencing technologies - the next generation. *Nature reviews. Genetics*, 11(1):31–46, 2010.

[138] J R Miller, S Koren, and G Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

[139] Jason R. Miller, Arthur L. Delcher, Sergey Koren, Eli Venter, Brian P. Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry, and Granger Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.

[140] Jason R. Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data, 2010.

[141] André E Minoche, Juliane C Dohm, and Heinz Himmelbauer. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and Genome Analyzer systems, 2011.

[142] Hiroyuki Mishima, Kensaku Sasaki, Masahiro Tanaka, Osamu Tatebe, and Kohichiro Yoshiura. Agile parallel bioinfor-

matics workflow management using Pwrake, 2011.

[143] Olena Morozova and Marco A. Marra. Applications of next-generation sequencing technologies in functional genomics, 2008.

[144] Antonio Muñoz Mérida, Enrique Viguera, M Gonzalo Claros, Oswaldo Trelles, and Antonio J Pérez-Pulido. Sma3s: A Three-Step Modular Annotator for Large Sequence Datasets. *DNA research : an international journal for rapid publication of reports on genes and genomes*, pages 1–13, 2014.

[145] Nicola J. Mulder and Rolf Apweiler. The InterPro database and tools for protein domain analysis, 2008.

[146] Stefan C Müller, Gustavo Alonso, Adam Amara, and André Csillaghy. Pydron: Semi-Automatic Parallelization for Multi-Core and the Cloud. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 645–659, Broomfield, CO, 2014. USENIX Association.

[147] K. Mullis, F. Faloona, S. Scharf, R. Saiki, G. Horn, and H. Erlich. Specific enzymatic amplification of DNA in vitro: The polymerase chain reaction. *Cold Spring Harbor Symposia on Quantitative Biology*, 51(1):263–273, 1986.

[148] Jeong-Hwan Mun, Hee Chung, Won-Hyong Chung, Mijin Oh, Young-Min Jeong, Namshin Kim, Byung Ohg Ahn, Beom-Seok Park, Suhyoung Park, Ki-Byung Lim, Yoon-Jung Hwang, and Hee-Ju Yu. Construction of a reference genetic map of Raphanus sativus based on genotyping by whole-genome resequencing. *TAG. Theoretical and applied genetics. Theoretische und angewandte Genetik*, 128(2):259–272, February 2015.

[149] Marvin Mundry, Erich Bornberg-Bauer, Michael Sammeth, and Philine G D Feulner. Evaluating Characteristics of De Novo Assembly Software on 454 Transcriptome Data: A Simulation Approach. *PLoS ONE*, 7(2):e31410, 2012.

[150] I. Navas-Delgado, N. Moreno-Vergara, A.C. Gomez-Lora, M. del Mar Roldan-Garcia, I. Ruiz-Mostazo, and J.F. Aldana-Montes. Embedding semantic annotations into dynamic Web contents. *Proceedings. 15th International Workshop on Database and Expert Systems Applications, 2004.*, 2004.

[151] Gil Neiger. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, 2006.

[152] B. Nevado, S. E. Ramos-Onsins, and M. Perez-Enciso. Resequencing studies of nonmodel organisms using closely related reference genomes: Optimal experimental designs and bioinformatics approaches for population genomics. *Molecular Ecology*, 23(7):1764–1779, 2014.

[153] I. Nikolaidis. Implementing CIFS: The Common Internet File System [Book Review]. *IEEE Network*, 18, 2004.

[154] Masaaki Nishikiori. Server virtualization with VMware vSphere 4. *Fujitsu Scientific and Technical Journal*, 47:356–361, 2011.

[155] Christine Noonan. Active Directory Cookbook. *ter: Technology Electronic Reviews*, 14:1, 2007.

[156] Rachelly Normand and Itai Yanai. An introduction to high-throughput sequencing experiments: Design and bioinfor-

matics analysis. *Methods in Molecular Biology*, 1038:1–26, 2013.

[157] E Novaes, D Drost, W Farmerie, G Pappas, D Grattapaglia, R Sederoff, and M Kirst. High-throughput gene and SNP discovery in Eucalyptus grandis, an uncharacterized genome. *BMC Genomics*, 9(1):312, 2008.

[158] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An Overview of the Scala Programming Language. *System*, pages 1–130, 2004.

[159] Aisling O'Driscoll, Jurate Daugelaite, and Roy D. Sleator. 'Big data', Hadoop and cloud computing in genomics, 2013.

[160] Oracle Corporation. MySQL :: What is MySQL?, 2012.

[161] Putty Org. PuttySSH, 2000.

[162] Joshua Orvis, Jonathan Crabtree, Kevin Galens, Aaron Gussman, Jason M. Inman, Eduardo Lee, Sreenath Nampally, David Riley, Jaideep P. Sundaram, Victor Felix, Brett Whitty, Anup Mahurkar, Jennifer Wortman, Owen White, and Samuel V. Angiuoli. Ergatis: A web interface and scalable software system for bioinformatics workflows. *Bioinformatics*, 26:1488–1492, 2010.

[163] Vicki Pandey, Robert C. Nutter, and Ellen Prediger. Applied Biosystems SOLiD??? System: Ligation-Based Sequencing. In *Next Generation Genome Sequencing: Towards Personalized Medicine*, pages 29–42. 2008.

[164] Paritosh Pandya. Principles of concurrent and distributed programming, 1991.

[165] H Parkinson, U Sarkans, M Shojatalab, N Abeygunawardena, S Contrino, R Coulson, A Farne, G Garcia Lara, E Holloway, M Kapushesky, P Lilja, G Mukherjee, A Oezcimen, T Rayner, P Rocca-Serra, A Sharma, S Sansone, and A Brazma. ArrayExpress–a public repository for microarray gene expression data at the EBI. *Nucleic acids research*, 33(Database issue):D553–D555, 2005.

[166] Konrad Paszkiewicz and David J. Studholme. De novo assembly of short sequence reads. *Briefings in Bioinformatics*, 11(5):457–472, 2010.

[167] Claude R Phipps. Virtualization : Benefits and Challenges. *Information Security*, 3343:575–582, 2010.

[168] Kim D. Pruitt, Garth R. Brown, Susan M. Hiatt, Françoise Thibaud-Nissen, Alexander Astashyn, Olga Ermolaeva, Catherine M. Farrell, Jennifer Hart, Melissa J. Landrum, Kelly M. McGarvey, Michael R. Murphy, Nuala A. O'Leary, Shashikant Pujar, Bhanu Rajput, Sanjida H. Rangwala, Lillian D. Riddick, Andrei Shkeda, Hanzhen Sun, Pamela Tamez, Raymond E. Tully, Craig Wallin, David Webb, Janet Weber, Wendy Wu, Michael Dicuccio, Paul Kitts, Donna R. Maglott, Terence D. Murphy, and James M. Ostell. RefSeq: An update on mammalian reference sequences. *Nucleic Acids Research*, 42(D1), 2014.

[169] Wiley Publishing. *Xen ® Virtualization*, volume 1. 2007.

[170] Megan J. Puckelwartz, Lorenzo L. Pesce, Viswateja Nelakuditi, Lisa Dellefave-Castillo, Jessica R. Golbus, Sharlene M. Day, Thomas P. Cappola, Gerald W. Dorn, Ian T. Foster, and Elizabeth M. Mcnally. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30:1508–1513, 2014.

[171] Python Software Foundation. Python Programming Language – Official Website, 2011.

[172] Mohammed A. Qadeer, Mohammad Salim, and M. Sana Akhtar. Profile management and authentication using LDAP. In *Proceedings - 2009 International Conference on Computer Engineering and Technology, ICCET 2009*, volume 2, pages 247–251, 2009.

[173] Michael A Quail, Miriam Smith, Paul Coupland, Thomas D Otto, Simon R Harris, Thomas R Connor, Anna Bertoni, Harold P Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC genomics*, 13(1):341, January 2012.

[174] Joshua Quick, Aaron R Quinlan, and Nicholas J Loman. A reference bacterial genome dataset generated on the MinION(TM) portable single-molecule nanopore sequencer. *GigaScience*, 3(1):22, January 2014.

[175] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. Hybrid MPI and OpenMP Parallel Programming. *Performance Computing*, 4192:11, 2006.

[176] Sergio Ramírez, Antonio Muñoz Mérida, Johan Karlsson, Maximiliano García, Antonio J. Pérez-Pulido, M. Gonzalo Claros, and Oswaldo Trelles. MOWServ: A web client for integration of bioinformatic resources. *Nucleic Acids Research*, 38(SUPPL. 2), 2010.

[177] Antonia Rana and Fabrizio Foscarini. Linux distributions for bioinformatics: an update. *EMBnet. news*, 15(3):pp—-35, 2009.

[178] Neil Rasmussen. The Different Types of UPS Systems. *Schneider Electric – Data Center Science Center*, pages 1–10, 2011.

[179] 000 rice genomes project The 3. The 3,000 rice genomes project. *GigaScience*, 3(1):7, 2014.

[180] Guillaume Rizk and Dominique Lavenier. GASSST: Global alignment short sequence search tool. *Bioinformatics*, 26(20):2534–2540, 2010.

[181] Richard J Roberts, Mauricio O Carneiro, and Michael C Schatz. The advantages of SMRT sequencing. *Genome biology*, 14(6):405, 2013.

[182] M.M. Roldan-Garcia, I. Navas-Delgado, and J.F. Aldana-Montes. A design methodology for semantic Web database-based systems. *Third International Conference on Information Technology and Applications (ICITA'05)*, 1, 2005.

[183] Paolo Romano. Automation of in-silico data analysis processes through workflow management systems, 2008.

[184] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Quo. The discovery net system for high throughput bioinformatics. In *Bioinformatics*, volume 19, 2003.

[185] Matthew Ruffalo, Thomas LaFramboise, and Mehmet Koyutürk. Compa-

rative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics*, 2011.

[186] Stephen M. Rumble, Phil Lacroute, Adrian V. Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. SHRiMP: Accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5), 2009.

[187] Alistair G. Rust, Emmanuel Mongin, and Ewan Birney. Genome annotation techniques: New approaches and challenges, 2002.

[188] Federico D. Sacerdoti, Sandeep Chandra, and Karan Bhatia. Grid systems deployment and management using rocks. In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, pages 337–345, 2004.

[189] M Salson, T Lecroq, M Léonard, and L Mouchard. Burrows-Wheeler Transform. *Theoretical Computer Science accepted 2009*, pages 13–25, 2008.

[190] The Sam and Format Specification. The SAM Format Specification (v1.4-r985). *Read*, pages 1–11, 2011.

[191] San Diego: Accelrys Software Inc. Discovery Studio Modeling Environment, Release 3.5, 2012.

[192] Alberto Sanz, Rafael Asenjo, Juan López, Rafael Larrosa, Angeles Navarro, Vassily Litvinov, Sung Eun Choi, and Bradford L. Chamberlain. Global data re-allocation via communication aggregation in chapel. In *Proceedings - Symposium on Computer Architecture and High Performance Computing*, pages 235–242, 2012.

[193] Herbert M Sauro and Frank T Bergmann. Standards and ontologies in computational systems biology. *Essays in biochemistry*, 45:211–222, 2008.

[194] Eric E Schadt, Steve Turner, and Andrew Kasarskis. A Window into Third Generation Sequencing. *Human molecular genetics*, 19(2):227–240, 2010.

[195] F Schmuck and R Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. *Proceedings of the First USENIX Conference on File and Storage Technologies*, pages 231–244, 2002.

[196] Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome biology*, 10(9):R98, 2009.

[197] Philip Schwan. Lustre: Building a File System for 1,000-node Clusters. *Proceedings of the Linux Symposium*, pages 401–409, 2003.

[198] Pedro Seoane, Rosario Carmona, Rocío Bautista, and Others. AutoFlow: an easy way to build workflows.

[199] J Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol, 2006.

[200] Specification Sheet. Helicos ™ Genetic Analysis System. *Analysis*, pages 1–4, 2010.

[201] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[202] S Shepler, B Callaghan, D Robinson, R Thurlow, C Beame, M Eisler, and D Noveck. Network File System (NFS) version 4 Protocol, 2003.

[203] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J M Jones, and Inanç Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[204] Guy St C Slater and Ewan Birney. Automated generation of heuristics for biological sequence comparison. *BMC bioinformatics*, 6:31, 2005.

[205] Andrew D. Smith, Wen Yu Chung, Emily Hodges, Jude Kendall, Greg Hannon, James Hicks, Zhenyu Xuan, and Michael Q. Zhang. Updates to the RMAP short-read mapping software. *Bioinformatics*, 25(21):2841–2842, 2009.

[206] Douglas R. Smith. Applications of a strategy for designing divide-and-conquer algorithms. *Science of Computer Programming*, 8(3):213–229, June 1987.

[207] Gregory Smith. *PostgreSQL 9.0 high performance*. 2010.

[208] L Stein. Genome annotation: from sequence to biology. *Nature reviews. Genetics*, 2(7):493–503, 2001.

[209] Lincoln D. Stein. Using GBrowse 2.0 to visualize and share next-generation sequence data. *Briefings in Bioinformatics*, 14:162–171, 2013.

[210] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A Parallel Workstation For Scientific Computation. In *Proceedings of the 24th International Conference on Parallel Processing*, volume 1, pages 11–14, 1995.

[211] Qinglan Sun, Li Liu, Linhuan Wu, Wei Li, Quanhe Liu, Jianyuan Zhang, Di Liu, and Juncai Ma. Web Resources for Microbial Data. *Genomics, Proteomics & Bioinformatics*, 13(1):69–72, 2015.

[212] CH Swaroop. A Byte of Python. *A Byte of Python*, page 110, 2003.

[213] Siun Chee Tan and Beow Chin Yiap. DNA, RNA, and protein extraction: the past and the present. *Journal of biomedicine & biotechnology*, 2009:574398, 2009.

[214] Taurin Tan-atichat and Joseph Pasquale. VNC in high-latency environments and techniques for improvement. In *GLOBECOM - IEEE Global Telecommunications Conference*, 2010.

[215] Bixia Tang, Yanqing Wang, Junwei Zhu, and Wenming Zhao. Web Resources for Model Organism Studies. *Genomics, proteomics & bioinformatics*, 13(1):64–68, 2015.

[216] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The triana workflow environment: Architecture and applications. In *Workflows for e-Science: Scientific Workflows for Grids*, pages 320–339. 2007.

[217] Ronald C Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11 Suppl 1:S1, 2010.

[218] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid. In *Grid computing: Making the global infrastructure a reality*, pages 299–335. 2003.

[219] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The Condor experience, 2005.

[220] The MPI Forum. MPI : A Message Passing Interface. In *Proceedings of the Conference on High Performance Networking and Computing*, pages 878–883, 1993.

[221] Mario Trangoni and Matías Cabral. A Comparison of Provisioning Systems for Beowulf Clusters.

[222] Chun-Yi Tsai and Tzao-Lin Lee. A remote collaboration system design and construction. In *Proceedings of the 5th International Confernece on Ubiquitous Information Management and Communication - ICUIMC '11*, page 1, 2011.

[223] Daniele Turi, Paolo Missier, Carole Goble, David De Roure, and Tom Oinn. Taverna workflows: Syntax and semantics. In *Proceedings - e-Science 2007, 3rd IEEE International Conference on e-Science and Grid Computing*, pages 441–448, 2007.

[224] Randy Urbano. *Oracle Database Advance Replication*. PhD thesis, 2007.

[225] W. M P Van Der Aalst and A. H M Ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[226] Sander Van Vugt. *The Definitive Guide to SUSE Linux Enterprise Server*. Apress, 2006.

[227] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwali, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN : Yet Another Resource Negotiator. In *ACM Symposium on Cloud Computing*, page 16, 2013.

[228] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature reviews. Genetics*, 10(1):57–63, 2009.

[229] René L. Warren, Granger G. Sutton, Steven J M Jones, and Robert A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, 2007.

[230] Thomas Wicker, Edith Schlagenhauf, Andreas Graner, Timothy Close, Beat Keller, and Nils Stein. 454 sequencing put to the test using the complex genome of barley. *BMC Genomics*, 7(1):275, 2006.

[231] Wiki. List of biological databases, 2015.

[232] Wikipedia. Convention over Configuration, 2007.

[233] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37:633–652, 2011.

[234] Weidong Wu, Brian P. Stupi, Vladislav A. Litosh, Dena Mansouri, Demetra Farley, Sidney Morris, Sherry Metzker, and Michael L. Metzker. Termination of DNA synthesis by N6-alkylated, not 3???-O-alkylated, photocleavable 2???-deoxyadenosine triphosphates. *Nucleic Acids Research*, 35(19):6339–6349, 2007.

[235] Yingjie Xia, Xingmin Shi, Li Kuang, and Junhua Xuan. Parallel geospatial analysis on windows HPC platform. In *2010 2nd Conference on Environmental Science and Information Application Technology, ESIAT 2010*, volume 1, pages 210–213, 2010.

[236] Xun Xu, Shengkai Pan, Shifeng Cheng, Bo Zhang, Desheng Mu, Peixiang Ni, Gengyun Zhang, Shuang Yang, Ruiqiang Li, Jun Wang, Gisella Orjeda, Frank Guzman, Michael Torres, Roberto Lozano, Olga Ponce, Diana Martinez, Germán De la Cruz, S K Chakrabarti, Virupaksh U Patil, Konstantin G Skryabin, Boris B Kuznetsov, Nikolai V Ravin, Tatjana V Kolganova, Alexey V Beletsky, Andrei V Mardanov, Alex Di Genova, Daniel M Bolser, David M A Martin, Guangcun Li, Yu Yang, Hanhui Kuang, Qun Hu, Xingyao Xiong, Gerard J Bishop, Boris Sagredo, Nilo Mejía, Wlodzimierz Zagorski, Robert Gromadka, Jan Gawor, Pawel Szczesny, Sanwen Huang, Zhonghua Zhang, Chunbo Liang, Jun He, Ying Li, Ying He, Jianfei Xu, Youjun Zhang, Binyan Xie, Yongchen Du, Dongyu Qu, Merideth Bonierbale, Marc Ghislain, Maria del Rosario Herrera, Giovanni Giuliano, Marco Pietrella, Gaetano Perrotta, Paolo Facella, Kimberly O'Brien, Sergio E Feingold, Leandro E Barreiro, Gabriela A Massa, Luis Diambra, Brett R Whitty, Brieanne Vaillancourt, Haining Lin, Alicia N Massa, Michael Geoffroy, Steven Lundback, Dean DellaPenna, C Robin Buell, Sanjeev Kumar Sharma, David F Marshall, Robbie Waugh, Glenn J Bryan, Marialaura Destefanis, Istvan Nagy, Dan Milbourne, Susan J Thomson, Mark Fiers, Jeanne M E Jacobs, Kåre L Nielsen, Mads Sø nderkær, Marina Iovene, Giovana A Torres, Jiming Jiang, Richard E Veilleux, Christian W B Bachem, Jan de Boer, Theo Borm, Bjorn Kloosterman, Herman van Eck, Erwin Datema, Bas te Lintel Hekkert, Aska Goverse, Roeland C H J van Ham, and Richard G F Visser. Genome sequence and analysis of the tuber crop potato. *Nature*, 475(7355):189–195, 2011.

[237] Yadong Yang, Xunong Dong, Bingbing Xie, Nan Ding, Juan Chen, Yongjun Li, Qian Zhang, Hongzhu Qu, and Xiangdong Fang. Databases and Web Tools for Cancer Genomics Study. *Genomics, Proteomics & Bioinformatics*, 13(1):46–50, 2015.

[238] T. Ylonen and C. Lonvick. RFC 4254: The Secure Shell (SSH) Connection Protocol, 2006.

[239] Guangchuang Yu, Li-Gen Wang, Xiao-Hua Meng, and Qing-Yu He. LXtoo: an integrated live Linux distribution for the bioinformatics community, 2012.

[240] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[241] Guoqing Zhang, Yunsheng Zhang, Yunchao Ling, and Jia Jia. Web Resources for Pharmacogenomics. *Genomics, proteomics & bioinformatics*, 13(1):51–54, 2015.

[242] Dong Zou, Lina Ma, Jun Yu, and Zhang Zhang. Biological Databases for Human Research. *Genomics, Proteomics & Bioinformatics*, 13(1):55–63, 2015.

Parte VIII

APÉNDICES

Apéndice A

Otras publicaciones

AutoFlow: an easy way to build workflows

Pedro Seoane, Rosario Carmona, Rocío Bautista, Darío Guerrero-Fernández y M. Gonzalo Claros

Plataforma Andaluza de Bioinformática & Dpto de Biología Molecular y Bioquímica, Universidad de Málaga, 29071 Málaga (Spain)

Abstract. Many bioinformatics tasks require the use of different software, making workflows a current need in this research field. There are workflow builders that usually try to simplify the interface disregarding a complex use. This may lead to a non-scalability limitation, or the dependence on the facilities available. Here it is presented AutoFlow, a workflow builder that can handle most computer systems. It has been developed in Ruby and accepts any kind of software that can even use very specific resources (such as GPU or FPGA). AutoFlow has been designed to automatically launch tasks to the queue system. It can then handle big workflows that can overflow the maximum execution time of the queue system provided that each individual task can be finished within the maximum execution time. AutoFlow has been implemented with iterative task capability. Other interesting capability is an environment variable system that allows the persistence of certain data for all tasks. This allows the data transfer from a task to the next task and so on, enabling the inclusion of decisions that can affect downstream tasks. AutoFlow includes tools to monitor task status, graphic representations of workflows, file searching and timing. Two case-of use are presented to illustrate AutoFlow capabilities: one workflow for assembling and annotation of several libraries of Roche 454 sequences and another workflow for RNA-seq analysis.

1 Introduction

The high-throughput sequencing technologies produce a large amount of data that require the development of large and complex workflows with lots of instructions. For example, genome sequencing generates large files of reads that must be pre-processed, assembled, verified, and then annotated. Typically, assembly and annotation can be performed using various programs and parameters to obtain different results that require further reconciliation or selection. Some way of automation of this repetitive task will be beneficial. Moreover, when selection is required, downstream tools are depending on this decision, providing different results. Therefore, a workflow or a pipeline is designed to encompass all the programs and parameters used to get the final result, and allows the user to save time and efforts by not having to launch the different tasks on his own.

Pipelining tasks allows to automatise the use different software tools, where the output data of a tool is used as input for other tools, or different input files must

converge to provide a single, final result. Nowadays, many platforms are able to build and execute workflows, such as Ergatis[1], Kepler[2], Triana[3], or Dyscovery Net[4], although the most widely used are Galaxy[5], Taverna[6] and their fusion, known as Tavaxy[7]. Both platforms are based on web services that users can combine to design and execute customised workflows. The above platforms have been designed to simplify the creation and execution of workflows, so that users without computing skills can use them. Among the most noticeable features, these platforms have a graphical user interface (GUI) that enables the user to design and execute workflows. Nevertheless, the usability is limiting the flexibility, and complex workflows or workflows with new software are no easy to handle. Furthermore, the user has not control on used resources and this can diminish the workflow performance.

Here it is presented AutoFlow, a Ruby-based workflow manager that allows building any desired workflow or pipeline. It is self-contained and the only dependences are GNUplot and dot[8]. It enables execution control by the user. It allows the design of dynamic workflows that can take decisions about the data while the workflow is running. Its main goal is to simplify repetitive tasks while removing limitations inherent to other workflow tools. On the other hand, it requires that users must have computing skills and knowledge about the software to be used.

2 Methods

2.1 Implementation of tasks

AutoFlow is a ruby gem that has been developed on Ruby 1.9 and SLES Linux. It uses a template script where every task is described with all its attributes and afterwards it is launched to the queue system (in this case, we use a module sentence of SLURM queue system, but other queue systems can be implemented). Each task is identified by a unique tag, which will be used for reference purposes and graphical representation. The general structure of a task is written as:

```
listing){
          module load software
          ?
          ls folder
}
```

where the first line is the task name or tag (i.e. *listing*) finishing with a ')' character. This tag will be used to identify the task in graphical representations (see below). The task definition, written between '{}', has two parts separated by a line starting by the control character '?'. The lines before '?' serve to initialise the environment, and the lines after '?' are the commands to be executed (i.e. *ls folder*). The first word of the first command is used as reference for the output storage. Since the sintax is bash-based, any software or platform based on command lines, such as Matlab, C, Ruby or Python, can be used.

Task iteration, changing only some parameter, can be easily done writing:

```
listing_[user;system;folder]){
```

```
        module load software
        ?
        ls (*)
}
```

where the parameter succession within brackets separated by semicolons (*[user;system;folder]*) serves to create a new task for each parameter. This is done replacing the '(*)' tag by each parameter. This is very useful for software benchmarkings and searching the best parameters in a program with a particular dataset.

When a task is depending on finishing a previous task, it can be declared as follows:

```
listing){
        module load software
        ?
        ls folder > temp
}

show){
        module load software
        ?
        cat listing)/temp # it will not be launched until 'temp' is finished
}
```

The string variables implemented in AutoFlow are helpful in, for example, task decisions. They are alphanumeric strings that begin with '$' or '@' characters (i.e. *$sentence* in the following example).

```
message){
        module load software
        ?
        echo $sentence
}
```

The variables must be declared in the command line that launches the workflow, allowing to the user to change parameters in the workflow without modifying the template. Therefore, the workflow is completely independent of the used data. '$' character indicates that the variable will not be changed along the workflow. On the contrary, variables beginning with '@' can be modified. The '@' variables are very useful to transfer certain data between tasks and it helps in making-decisions.

2.2 Launching workflows

Workflows can be launched as follows:

```
Autoflow -w template
```

where -w indicates that *template* is the input data. This is the only mandatory parameter.

Before launching a new workflow, it must be checked to find errors and inconsistencies. The command line option *--graph* generates a graphical representation of the workflow, where the tasks are the nodes and the relations are the dependencies. The representation can be semantic (Fig. 1 and Fig. 2A) or structural

(Fig. 2B): in the semantic representation, the tasks are represented with their tags, allowing to the user to interpret the workflow easily. The structural representation uses the main command name of every task. These plots show which programs are used in the workflow and where the data are saved by each task. As a result, relation inconsistencies are becoming apparent, facilitating the identification and fixing by the user.

Another command line option, *--verbose* that generates a list of tasks with their attributes on command line, can also be helpful in debugging.

When a template is launched, AutoFlow generates a job for every task in the queue system. To do so, a folder per task is created with the name of their main command (as in the structural representation, Fig. 2B). All the folders are saved within the default folder *exec* where AutoFlow is running. Within each folder, a bash file is created with all necessary information for the queue system. AutoFlow replaces all key characters by their values and all the dependencies by absolute paths. Then, each bash file is sent to the queue system and AutoFlow takes its job ID that is used to control dependencies (if any) and the task launch timing. AutoFlow ends his work when the last task is queued.

Three additional information state files are created: (1) a log file containing the start and the end of each task; (2) a file containing the relations between tag task and where has it been saved; and (3) a file where all dynamic variables (that begins with '@' character) are defined. This file is loaded by all the task and it is created only if there is set a dynamic variable in the template.

2.3 Experimental data

Two dataset have been used in this work to illustrate AutoFlow capabilities: (i) for Case 1, a 454 dataset of two different tissues: A (372 750 reads) and B (429 909 reads), and (ii) for Case 2, an Illumina dataset of two different conditions (control and treatment) with three experimental replicates each one (experimental replicates have 2 457 983, 2 866 872 and 988 173 reads, and controls have ≈ 3 000 000 reads).

3 Results

3.1 Case 1: sequence assembly and annotation

The 454 dataset has been used to obtain three different assemblies with interconnected tasks: one for each tissue and another one with the whole dataset. The three assemblies can be obtained with the same template (http://www.scbi.uma.es/web/pedro/assembly_annotation.txt). This workflow was launched as follows:

```
Autoflow -w assembly_annotation -s -c 100 -u '-1' -t '5-
00:00:00'
```

The *-c* parameter says to the program how many cores must be used by each task. Previously, the user must replace the number of cores parameter in a command by the '*[cpu]*' string instead when he writes the template. *-s* says to Autoflow that tasks can use different nodes and *-u '-1'* that queue system must assign the best number of nodes to do the job. Besides, *-t* indicates the limit of time per task (in this case 5 days) .

The process described in Fig 1 allows the evaluation of the assembly quality based on the number of unigenes with orthologue, the number of unique unigenes, and the number of unique full length unigenes (Table 1). The manual inspection of Table 1 allows the selection of column "All-kmer29" as the better assembly, and these unigenes can be submitted to a full annotation with Sma3 [13]. However, based on these results, a new decision task was included (results not shown) to automatically select the best assembly (All-kmer29) that will be directed to Sma3 annotation that provided 34 971 annotated sequences. Sma3s lacks parallelisation capabilities but can be executed taking advantage of array jobs. Although AutoFlow cannot handle array jobs as such, it can be included within a wrapped easily created with SCBI_MapReduce[14] to confer parallelization capabilities to Sma3s. In future workflows, a wrapped Sma3 can be included in AutoFlow workflows.

In conclusion, this case-of-use of AutoFlow takes advantage of two features of AutoFlow: (1) its capability for using results generated in previous executions and (2) the capability of taking decisions when the workflow is running.

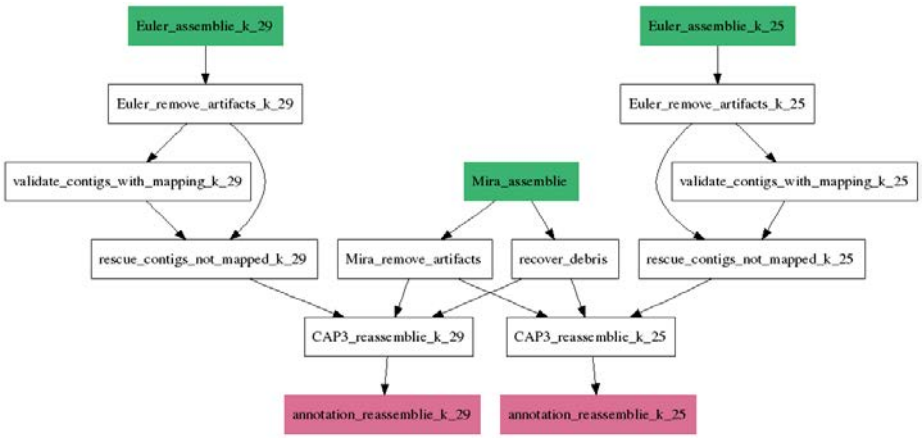


Fig. 1. Semantic representation of Case 1 workflow. Green boxes correspond to starting tasks, and magenta boxes are the finishing tasks. Right and left branches of the workflow differ on the assembly k-mer used by Euler[10]. Central branch correspond to an assembly using Mira[11]. Initial assemblies are analysed to recover putative useful «debris» and to remove artefactual contigs using Full-LengtherNext and Bowtie[12], respectively. CAP3[9] is used to reconcile Euler and Mira verified contigs, and the resulting supercontigs are analysed using Ful-LengtherNext to obtain data of Table 1.

Table 1. Key parameters of Case 1 workflow

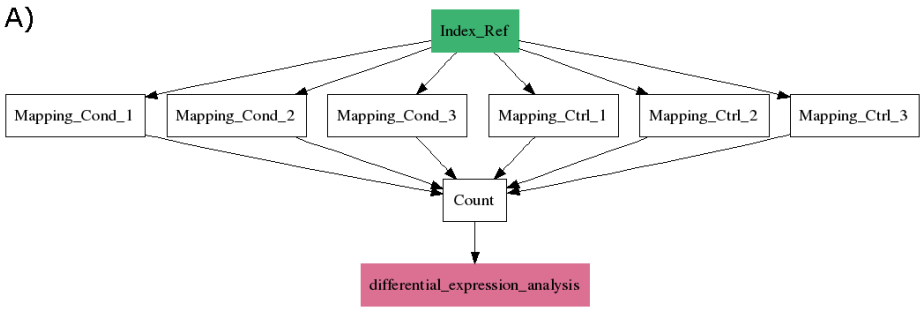
	Tissue A		Tissue B		All	
	kmer 25	kmer 29	kmer 25	kmer 29	kmer 25	kmer 29
Unigenes	37283	37235	14178	14191	44837	44858
Unique unigenes	13775	13786	5962	5955	15328	15307
Unique full-length unigenes	3882	3942	1512	1504	4736	4765

3.2 Case 2: RNA-seq analyses

This workflow (Fig. 2; http://www.scbi.uma.es/web/pedro/expression_analysis.txt) is a simple example that takes advantage of iterative capabilities of AutoFlow. The launching command was:

```
Autoflow -w template -c 4 -V '$evalue=p-value'
```

This command line has the new parameter *-V* that is used by the user for set internal variables to desired values. The same workflow with the same data has been launched three times, setting the static variable *$evalue* to 0.01, 0.05, and 0.1. The *$evalue* variable is the cut-off *P*-value that uses our in-house pipeline for analysing RNA-seq data to determine the statistic significance. The results were 2 780, 2 937 and 2 995 differentially expressed genes, respectively. As expected, the lower the *P*-value, the greater the number of genes. More sophisticated, iterative analyses with AutoFlow could help in the determination of the best *P*-value threshold without the need of human intervention.



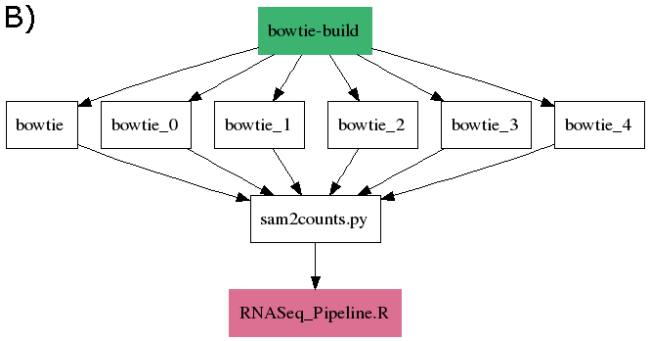


Fig. 2. Semantic (A) and structural (B) representation of the RNA-seq analysis workflow of Case 2. The workflow first creates a mapping reference index (*Index_ref*), then makes the mapping with the control and conditions using Bowtie2 (*Mapping_Cond_x* and *Mapping_Ctrl_x*), then makes the stats (*Count*) using the Python script sam2counts.py, and finally makes the differential expression analysis (*differential_expression_analysis*) with our in-house RNASeq_Pipeline.R script.

4 Discussion

AutoFlow is an easy-to-use and efficient workflow manager for researchers with programming skills but without expertise in workflow design. This tool can manage large and complex workflows, or can simplify repetitive workflows taking advantage of the automatic management of iterative tasks.

The parameters and steps of workflow of Case 1 was fine-tuned using different input data (results not shown). This demonstrates that the same workflow can be used with different input data, allowing its re-use for several different experiments. Moreover, the same workflow can be shared among laboratories provided that they have the same software and queue system. Therefore, when several researches in a project need to face data from different sources (laboratories) or organisms (animals, plants, microorganisms...), they can use the same AutoFlow workflow changing or tuning some parametres or programs.

As is shown in Figs. 1 and 2, and since AutoFlow is not limited by any database or ontology, it allows the incorporation of any desired software in a workflow, provided that the software works as a command line tool. Tools programmed in different languages (C, C++, Ruby, Perl, Python, R, Java, etc.) can be combined without any problem.

Finally, since AutoFlow is using bash for building the tasks and the environment variables system, it contains the flexibility and power of command lines and scripts. Consequently, the workflows can be dynamic and can take decisions when tasks are on-going.

References

1. Orvis J, Crabtree J, Galens K, Gussman A, Inman JM, Lee E, Nampally S, Riley D, Sundaram JP, Felix V, Whitty B, Mahurkar A, Wortman J, White O, Angiuoli SV: Ergatis: A web interface and scalable software system for bioinformatics workflows. Bioinformatics 26 (12). (2010) 1488-1492
2. Ludäscher B, Altintas I, Berkley C. D H: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience 13(10) (2006) 1039–1065
3. Taylor I, Shields M, Wang I, Harrison A: The Triana workflow environment: Architecture and Applications. In: Workflows for e-Science, Springer (2007) 320–339.
4. Ghanem M, Curcin V, Wendel P, Guo Y. Building and using analytical workflows in discovery net. In: Data mining on the Grid, John Wiley and Sons (2008).
5. Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M. R., Li P., Oinn T.: Taverna: a tool for building and running workflows of services. Nucl. Acids Res. 34 (suppl 2). (2006) W729-W732
6. Goecks J., Nekrutenko A., Taylor J. and The Galaxy Team: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biology 11(8). (2010) R86
7. Abouelhoda M., Issa S. A., Ghanem M.: Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. BMC Bioinformatics 13 (2012) 77
8. Ellson J. , Gansner E. R., Koutsofios E., North S. C., Woodhull G.: Graphviz and dynagraph – static and dynamic graph drawing tools. In: Graph Drawing Software. (2004) 127-148
9. Huang, X. and Madan, A.: CAP3: A DNA sequence assembly program. Genome Res 9. (1999) 868-877.
10. Pevzner PA., Tang H., Waterman MS.: An Eulerian path approach to DNA fragment assembly. PNAS 18(17) (2001) 9748–9753
11. Chevreux, B., Wetter, T. and Suhai, S.: Genome sequence assembly using trace signals and additional sequence information. In: Computer Science and Biology: Proceedings of the German Conference on Bioinformatics. (1999) 45-56.
12. Langmead B., Salzberg S.: Fast gapped-read alignment with Bowtie 2. Nature Methods 9. (2012) 357-359.
13. Muñoz-Mérida A, Viguera E., Claros M. G., Trelles O., Pérez-Pulido A.J.: Sma3s: a three-step modular annotator for large sequence datasets. DNA Research. In press. (2014)
14. Guerrero-Fernández D., Falgueras J., and Claros M. G.: SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+. Computational Biology Journal 2013 (2013) ID 707540

Biology **2012**, *1*, 439-459; doi:10.3390/biology1020439

OPEN ACCESS

biology

ISSN 2079-7737

www.mdpi.com/journal/biology

Review

Why Assembling Plant Genome Sequences Is So Challenging

Manuel Gonzalo Claros [1,2,*], Rocío Bautista [2], Darío Guerrero-Fernández [2], Hicham Benzerki [1,2], Pedro Seoane [1] and Noé Fernández-Pozo [1]

[1] Department of Molecular Biology and Biochemistry, Faculty of Sciences, University of Malaga, 29071 Málaga, Spain; E-Mails: bhicham538@gmail.com (H.B.); seoanezonjic@uma.es (P.S.); noefp@uma.es (N.F.-P.)

[2] Bioinformatics Andalusian Platform, Bio-innovation Building, University of Malaga, 29590 Málaga, Spain; E-Mails: rociobm@uma.es (R.B.); dariogf@uma.es (D.G.-F.)

***** Author to whom correspondence should be addressed; E-Mail: claros@uma.es; Tel.: +34-951-952-787; Fax: +34-952-132-041.

Received: 16 July 2012; in revised form: 5 September 2012 / Accepted: 6 September 2012 / Published: 18 September 2012

Abstract: In spite of the biological and economic importance of plants, relatively few plant species have been sequenced. Only the genome sequence of plants with relatively small genomes, most of them angiosperms, in particular eudicots, has been determined. The arrival of next-generation sequencing technologies has allowed the rapid and efficient development of new genomic resources for non-model or orphan plant species. But the sequencing pace of plants is far from that of animals and microorganisms. This review focuses on the typical challenges of plant genomes that can explain why plant genomics is less developed than animal genomics. Explanations about the impact of some confounding factors emerging from the nature of plant genomes are given. As a result of these challenges and confounding factors, the correct assembly and annotation of plant genomes is hindered, genome drafts are produced, and advances in plant genomics are delayed.

Keywords: plant sequencing; NGS; complexity; repeats; assemblers; polyploidy; bioinformatics

---

1. Introduction

Higher plants are the Earth's dominant vegetation in nearly all ecosystems. They sustain living beings (including humans) by providing oxygen, food, fiber, fuel, medicines, spirits, erosion defense, flooding control, soil regeneration, (bio)remediation, urban cooling, green spaces (including gardens) and CO_2 lowering, and contributing to the control of global warming [1]. Higher plants also exhibit a wide range of forms, with individuals ranging in size from floating *Wolffia* plants of 1 mm in length to trees of more than 100 m in height or with a trunk diameter exceeding 10 m (such as the angiosperm *Eucaliptus regnans* and the gymnosperms *Sequoia sempervirens* and *Taxodium mucronatum*). Plants also contain the longest-living organisms (with *Pinus longaeva, Taxus baccata* and *Picea abies* individuals living on Earth for nearly 5,000–8,000 years). Moreover, plants are stuck in place and cannot escape enemies or uncomfortable conditions and need to develop strategies that improve their chances of survival due to sessility. So, plants have evolved tens of thousands of chemical compounds which they use to ward off competition from other plants, to fight infections, and to respond generally to the environment [2]. In consequence, plant species have larger and more complex genome sizes and structures than animal species and exhibit tremendous diversity in both size and structure [3]. Therefore, plants seem to be an important source of biological knowledge and economic profit, but relatively few plant species have been sequenced. In fact, in a world with >370,000 known plant species (with probably many thousands more still unclassified), only ~80,000 species have at least one single sequence in GenBank.

The publication of the first plant genome sequence of *Arabidopsis thaliana* [4] provided and improved the genetic landscape for studying all plants and has paved the way for sequencing several other plant genomes. It has also transformed the methods and tools for plant research and crop improvement [5]. *Arabidopsis*, and later *Oryza sativa* (rice) [6], *Carica papaya* (papaya) [7] and *Zea mays* (maize) [8] were sequenced using the classical Sanger method. The arrival of next-generation sequencing (NGS) technologies has allowed the rapid and efficient development of genomic resources for non-model or orphan plant species [9–13]. However, only *Arabidopsis* and rice—sequenced by Sanger's method using a BAC-by-BAC approach—have been really finished to date, the rest being drafts in a greater or lesser stage of completion. Unfortunately, even the complete or gold standard genomes contain gaps in their sequences corresponding to highly repetitive sequences, which are recalcitrant to sequencing and assembly methods [14]. A summary of all published plant genome sequences to date can be found in Table 3 in [14] and in Table 3 in [5].

Since there is no central focus in the scientific plant world, the choice of plant genomes for sequencing has been driven mainly by cost efficiency and the avoidance of complexity, and hence only plants with relatively small genomes (median size of 466 Mbp) were selected for sequencing in the first instance, although the most important crops have a median size of 766 Mbp [5]. In fact, *Arabidopsis thaliana* proves to be an outlier amongst plants because its genome has undergone a 30% reduction in genome size and at least nine rearrangements in the short time since its divergence to *Arabidopsis lyrata* [1,15]. In many plant species, it is now clear that a single genome sequence does not necessarily reflect the entire genetic complement [16,17], opening a new branch in the study of pan-genomes and core genomes [18].

Most plant sequencing efforts have been dedicated to angiosperms, mainly the eudicots, under which the most economically important crops are classified [19,20]. But sequencing efforts should be expanded beyond the traditional commodity crops and include other non-commodity crops and non-model species (e.g., conifers, ferns and other bryophytes). We present here the current state of the art of challenges and confounding factors that explain why plant genomics is less developed than animal genomics and remains so focused on small genomes. We also discuss why challenges are not overcome by the arrival of NGS.

2. From Sanger Technology to NGS: Getting Plants off the Ground

While extremely successful in the past, Sanger sequencing [21] does present the following drawbacks for actual sequencing projects: (1) requirement of nucleic acid subcloning, (2) clone amplification in hosts, (3) low throughput, (4) slow sequencing speed, and (5) high costs (both in terms of consumables and salaries, averaging $1,330 per Mbp [22]). This is the reason why sequencing projects with Sanger technology have always been carried out by international consortia [4,8,23,24].

NGS strategies allow a single template molecule to be directly used to generate millions of bases at low cost with a less cumbersome laboratory protocol. There are three NGS platforms widely used nowadays that are considered to be second-generation sequencing: (1) the Genome Sequencer FLX+/454 from Roche which is capable of producing over a million reads of up to 800 bases per 10 hour run, yielding a total of 0.7–1 Gbp at a price of approximately $90 per megabase; (2) the Genome Analizer from Illumina, of which the latest version, HiSeq2000, yields 100 Gbp of bases per day (26–150 bp read length) at a cost of $4 per megabase; and (3) the Applied Biosystems SOLiD (Sequencing by Oligo Ligation and Detection) that produces 10–300 Gbp of short reads (up to 75 bp) per run at a similar cost. The three platforms offer the paired-end sequencing technique. As a result, even large plant genomes can count on relatively inexpensive deep coverage with reads of 100 bp and paired-end libraries from 1 to 5 kbp (we will see that deep coverage does not allow for complete plant sequencing). A detailed description is beyond the scope of this article, and several reviews illustrate the rapid evolution of these and the newest NGS technologies (to cite a few, [25–31]). While 454 FLX+ and Sanger technologies are considered to produce long reads (600–800 pb in average), the other two produce short reads (<150 bp in average). Short-read technologies compensate the shortness of the sequences with a high coverage, so that bacteria can be successfully sequenced with a 40×–50× coverage, but as the genome increases in complexity, coverage of 100× may still be inefficient [32–34]. In contrast, long-read technologies do not need such deep coverage, with 20×–30× being enough for a good compromise between costs and assembly quality [32].

NGS is becoming the new sequencing standard for the following reasons: (1) simplification of the sequencing process (DNA cloning is not required); (2) miniaturization and parallelization (low cost); and (3) good adaptation to a broad range of biological phenomena (genetic variation, RNA expression, protein-DNA interactions, gene capture, methylation, *etc.*). But not everything about NGS is an advantage [25]: (i) the base calls are at least tenfold less accurate than Sanger sequencing base calls; (ii) the sequence length is shorter than in Sanger technology and requires dedicated assembly algorithms; and (iii) the quality of the NGS assemblies is also lower than Sanger assemblies. As a result, most plant genomes sequenced by NGS produce “drafts” that are suitable for (1) establishing

gene catalogues, (2) deciphering the repeat content, (3) glimpsing evolutionary mechanisms, and (4) performing early studies on comparative genomics and phylogeny. Unfortunately, drafts (i) hinder the progress of capturing accurately the information embedded in the repetitive fraction of the genome; (ii) make it difficult to distinguish genes from pseudogenes; and (iii) make it difficult to differentiate between alleles and even paralogues [35]. If only draft genomes are produced in the short future, plant genomics may face a crisis since, although the complex genomes of many more species are now accessible, the portion of each genome that can be reliably accessed has diminished substantially (<80%). The expertise and motivation for sequencing plant genomes to a high quality is disappearing, pushed by the rapid publication of a new draft genome lacking up to 20% of the genome [33].

Widespread adoption of NGS technology is tightly bound to bioinformatics. Integration of the many complex and rich sequencing datasets has yielded cohesive views of cellular activities and dynamics (for example, see [36–38]). The increase in plant sequence data has also prompted the development of dedicated repositories, such as the general purpose Phytozome [13], the comparative plant genomics resource PLAZA [39], plant family databases such as TreeGenes for forest tree genome data [40], or species specific databases (e.g., EuroPineDB for maritime pine [41], Eucawood for *Eucalyptus* [42], or MeloGen for *Cucumis melo* [43]). It is worth mentioning the iPlant project [44], which emerged with the aim of creating an innovative, comprehensive and foundational cyber infrastructure to support plant biology research, the VirtualPlant platform [45], integrating genome-wide data on the known and predicted relationships among genes, proteins, and molecules in order to enable scientists to visualize, integrate, and analyze genomic data from a systems biology perspective or the Plantagora platform [34], which addresses the gap between having the technical tools for plant genome sequencing and knowing precisely the best way to use them.

NGS can be said to have accelerated biological research in plants by enabling the comprehensive analysis of genomes, transcriptomes and interactomes. Moreover, translational research has been spurred by NGS, the most successful case being the application of a gene from *A. thaliana* to improve abiotic stress tolerance traits in crops [5]. But if NGS only produces draft genomes, it could drive plant functional genomics into a dead end in the near future.

3. Challenging Features of Plant Genomes

Genome size, duplications and repeat content are factors to be considered for all genomes to be sequenced. In particular, plant genomes usually appear as gene islands among the background of high copy repeats (usually >80%), where 95% coverage of the genes is assumed, based on comparisons with cDNA databases. This discouraging situation can be explained by several plant features that hinder the sequence assembly and annotation, and severely limiting genomics research productivity.

3.1. Sampling

The main drawback of plant sequencing is that it is often very hard to extract large quantities of high-quality DNA from plant material, making it difficult to prepare proper libraries for sequencing. Additionally, although any genome sequencing project is afforded with samples from a single plant, the situation is completely different in transcriptome sequencing, where the traditional approach was to use a variety of tissues and conditions from different multiple accessions by different researchers,

resulting in many extremely similar unigenes representing the same gene [41–43]. When such a heterogeneous transcriptome is studied using long reads, the presence of multiple alleles does not significantly hamper the unigene assembly [22], but when the transcriptome is studied with NGS technologies providing reads <100 bp, alleles and paralogues really do impair the assembly result.

3.2. Genome Size and Complexity

Plant-specific needs are sustained by new genes that may arise from gene duplications, alternative gene splicing, ploidy or gene retention following genome duplication, making plant genomes large and complex, as pointed out in the introduction. In fact, genome sizes across land plants range over two to three orders of magnitude, with an average around 6 Gbp, which is one order of magnitude larger than the average size of genomes sequenced so far [3]. Current sequencing technologies can manage large, complex genomes, such as wheat (*Triticum aestivum* with 16 Gbp split in 21 chromosomes) or pines (22–33 Gbp split in 12 chromosomes), so the genome size is not an unassailable issue anymore. The real problem is not the genome size *per se* but the complexity of the genomes, since the number of genes does not vary to the same extent as the genome size. The length of single-copy regions (always flanked by repeated sequences [12]) varies widely among plant species. In general, two types of arrangements are recognized: (1) short period interspersion (single copy sequences of 300–1,200 bp interspersed as islands among short lengths (50–2,000 nt) of repeat sequences); and (2) long period interspersion (single copy sequence islands of 2,000–6,000 bp interspersed among long repeat sequences). Genome size appears to be related to the type of interspersion: Plant species with small genomes, such as *Arabidopsis*, have long period interspersion and longer lengths of non repetitive sequences; on the contrary, plant species with large genomes, such as wheat, rye or maize, have short period interspersion and shorter non-repetitive sequences [46]. This confirms the intuitive notion that small genomes are therefore less difficult to assemble than larger genomes. The different factors that can contribute to the large variation of genome size and complexity in plants are discussed below.

3.3. Transposable Elements

During evolution, transposons have introduced profound changes in genome size, structure and function between species and within species [18], accounting for the major force in reshaping genomes [47]. This could explain why Chromosomes 1 and 2 of *A. thaliana* are a fusion of Chromosomes 1 and 2, and 3 and 4, respectively, of *A. lyrata* [15,47]. Transposable elements are by far the most highly represented repetitive sequence in plant genomics: due to the replicative nature of the retrotransposition process, Class I transposons (including retrotransposons) can account for up to 90% of all the transposons, while Class II elements are much less abundant [48]. Small-genome plants like *Arabidopsis* and rice are sparsely populated by transposons, containing 5.6% and 17% respectively. In contrast, the transposon-derived fraction of medium/large genomes may reach 85% in maize and >70% in barley [8,49,50]. Owing to their abundance and repetitive nature, transposable elements complicate genome assembly, particularly when short-read technologies are used [51].

3.4. Heterozygosity

Most plants are heterozygous, particularly those that have not been domesticated in laboratories [52]. Since it is a kind of redundancy, which is always a challenging factor in assemblies, only euchromatic regions of the genomes can be assembled, and a high percentage of NGS reads remain unassembled (15% in poplar [*Populus trichocarpa*] [53]). This happens even if a hierarchical clustering guided by a physical map is used to guide the sequence assembly. As a result, the poplar genome seems to contain a duplicated gene content since most loci present both possible alleles. The relative incompleteness of both heterozygous genomes demonstrates the difficulty of producing high-quality genome sequences for a natural, heterozygous cultivar with current sequencing technologies. As a consequence, some plant-sequencing projects tend to focus on homozygous derivatives, even if they are not commercially or agronomically important. This was the case, for example, for the highly homozygous genotype of *Vitis vinifera* (grape) in 2007 [54]. Another problem introduced by heterozygosity is the creation of false segmental duplications in assemblies that occur when heterozygous sequences from two haplotypes are assembled into separate contigs and are scaffolded adjacent to each other rather than being merged [55]. In conclusion, only the use of longer reads would improve the ability to assemble separate haplotypes within a genome (see 'Polyploidy' section below).

3.5. Polyploidy

Polyploidy is the result of the fusion of two or more genomes within the same nucleus. It originates from either whole-genome doubling (autopolyploidy) or by interspecific or intergeneric hybridizations followed by chromosome doubling (allopolyploidy). Genome duplication has the following potential advantages: (i) it is a source of genes with new functions and new phenotypes, (ii) some polyploids appear to be better adapted as a consequence of genome plasticity [56], and (iii) others lose their self-incompatibility, gain asexual reproduction, and produce higher levels of heterozygosity; this may explain the widespread occurrence of polyploids in plants [57]. Polyploidization is therefore one of the major driving forces in plant evolution and is extremely relevant to speciation and diversity [1,58]. An ancestral triplication affecting most (or perhaps all) dicots was followed by two additional whole-genome duplications [1,15]. Every plant lineage shows traces of additional, independent and more recent whole genome duplications somewhere between 50 and 70 million years ago [15]. Some genes have been repeatedly restored to single-copy status following many different genome duplications [59], with the degree of gene retention differing substantially in the different taxa. Therefore, the resulting assembly of a plant genome is dependent on whether the species is an autopolyploid, an allopolyploid, or on the age of the ploidization event. Sequencing of recent polyploids will be especially complex depending on the divergence of the duplicated genes, particularly in the case of many important crops that are true polyploids (banana, potato, cotton, wheat or sugarcane). The redundancy created by the presence of two or more sets of genes within a nucleus can affect the accuracy of the assembly, and the need to differentiate between homologues could influence the final utility of the obtained sequence. Indeed, contigs can break at polymorphic regions or misassemblies can be obtained between large-scale duplications.

The ploidy issue has been 'resolved' in different ways. For example, since most cultivated potatoes are tetraploids, the Potato Genome Sequencing Consortium decided to use as reference a doubled monoploid that was homozygous for a single set of the 12 chromosomes [60]. The authors found that the two haplotypes within a heterozygous diploid were more divergent from each other than from the single haplotype used as reference. In the case of the cultivated strawberry, which is allo-octaploid, the diploid species *Fragaria vesca* (woodland strawberry) was sequenced to bypass the difficulties of polyploidy [61]. For hexaploid wheat, the Wheat Genome Initiative has decided to follow another strategy: a flow cytometry separation of the 10 chromosomes one by one or in groups, the construction of a tiling BAC physical map, and subsequent sequencing of each chromosome using a BAC-by-BAC strategy [8].

3.6. Gene Content and Gene Families

The gene content in plants can be very complex, as shown by the presence of large gene families and abundant pseudogenes derived from recent genome duplication events and transposon activity (see above and [8]). For example, there are remnants of chloroplast and mitochondrial genes in the nuclear genome that skew coverage levels [7], such as ~270 kbp of the mitochondrial genome inserted into Chromosome 2 of *Arabidopsis* [62]. But gene duplication is regarded as a major force in the origin of new genes and genetic functions. By way of example, the appearance of C4 photosynthesis has evolved from the C3 pathway and has appeared independently on at least 50 occasions during plant evolution [63]. Other examples of gene duplication are the striking increase in the number of starch-associated genes in papaya (39) with respect to *Arabidopsis* (20), or the expanded number of kinase family members, cytochromes P450 and the enzymes engaged in plant secondary metabolism [64]. However, recent comparisons of *Arabidopsis*, poplar, grapevine, papaya and rice genomes estimated that the angiosperm ancestor should contain between 12,000 and 14,000 genes [15]. As a result, more than half of plant genes are really a gene family, 45% of them with the same function but different expression patterns [65]. Specific strategies are required to distinguish alleles from paralogues when sequencing natural heterozygous isolates, although this is not expected to have a very promising success in the near future [59]. Moreover, the presence of out-paralogues produced by duplication prior to the divergence of two lineages and in-paralogues produced in each lineage, together with the multiple rounds of polyploidy in plant lineages, accentuate these problems as divergence between paralogues occurs at different paces.

A curious finding in virtually all eukaryotic genomes sequenced to date (including plants) is the existence of lineage-specific genes for which an orthologue cannot be discerned in closely related species. Lineage-specific genes are a tantalizing target for functional studies since they should distinguish closely related taxa, but unfortunately, these apparently 'lineage-specific genes' could simply be the result of misassemblies [1]. Attention should be paid to these genes before a promising theory can be proposed. Bioinformatic efforts should be made to distinguish real, new genes from misassembled sequences, since we suspect that apparently new genes in sequences <150 bp in length correspond to misassemblies [66]. This also explains the fact that gene sequences may not always be correct, since nearly identical gene families are notoriously difficult to assemble and may collapse into a mosaic sequence without necessarily representing any member of the family [67].

Finally, gene movements can affect plant genome assembly. Gene movement studies found that many gene categories in *Arabidopsis*, papaya and grape were recently transposed at a basal frequency of 5%. The most striking result was that some gene families exhibited very high movement frequencies (50%–90%) [1,68]. This should not be a problem for any assembly procedure since jumping usually occurred a long time ago and the sequences have diverged, but the real drawback is that the region around the transposed gene is enriched with authentic transposons, phantom transposons and pseudogenes [69]. This situation directly impinges on the problem of assembly of repeated sequences and can cause gene loss in the assembly due to collapse of the repetitive surroundings.

3.7. Non-Coding RNAs

Non-coding RNAs (ncRNAs) were first described in plants in 1993 [70] and since then, they have provided new insights into gene regulation in plant and animal systems. The advent of NGS has produced a profound impact on the discovery of new ncRNAs. There are small ncRNAs with mature lengths below 30 bp, such as microRNA (miRNA), small interfering RNA (siRNA) and Piwi-interacting RNAs (piRNAs, usually found in animals). Long ncRNAs (200 bp long or more) are another subset of ncRNAs that contain many signatures of mRNAs, including 5' capping, splicing and poly-adenylation, but have little or no open reading frame [71]. Genomic sequences within ncRNAs are often shared within a number of different coding and non-coding transcripts in the sense and antisense directions giving rise to a complex hierarchy of overlapping isoforms. To add even more complexity to ncRNAs, a high proportion of them are variants of protein-coding cDNAs. When using short-read NGS strategies, sequence complexity frustrates the assembly of ncRNA precursors due to their repetitive nature since most ncRNAs contain fragments that are complementary to one or more genes, which causes the collapse of assemblers at the exon or, primarily, at the ncRNA gene [72]. Only long read-based strategies could cover both mature ncRNAs and ncRNA precursors provided that long ncRNAs are not longer than the read lengths.

3.8. Widely Distributed Repetitive Sequences (Low Complexity Sequences)

Plants share with other organisms a common source of general repetitive sequences [73] that are a source of low complexity regions, which are always a problem for assemblies. The main sources of repeats are the following:

- **Repetitions among chromosomes:** Duplications occurring both within chromosomes (e.g., ~250 tandem duplications each of ~10 kbp on Chromosome 2 of *Arabidopsis*) and between chromosomes (e.g., ~4 Mbp long regions between Chromosomes 2 and 4, or 700 Mbp long regions between Chromosomes 1 and 2 in *Arabidopsis*; ~3 Mbp at the termini of the short arms of Chromosomes 11 and 12 in rice, as well as Chromosomes 5 and 8 in sorghum) [62,74].
- **rDNA units:** These contain the rRNA genes, which are presented as hundreds of copies. Each unit is typically 10 kbp in plants and as a whole they represent up to 10% of the genome (for example, 8% in *Arabidopsis* [75]). They have not been resolved by any sequencing technology.
- **Satellites:** These are arrays of many tens or even thousands of identical or nearly identical copies of a repeated unit. They are abundant at centromeres and constitutive heterochromatin. For example, a

total of 3% of the *Arabidopsis* genome consists of the 180 bp centromeric repeat [76]. As a result of microsatellites, most sequenced chromosomes are split into two sequences, the right arm and the left arm, since the repetitive, centromeric sequence is unknown.

- **Microsatellites or SSRs (simple sequence repeats):** These are short tandem repeats (in the range of kbp) of short motifs (1–5 bp) repeated a few hundred times or less, with different microsatellites having different motifs. They are often highly polymorphic with regard to the number of repeat units in a repeat [77]. Microsatellites are mainly located at the subtelomeric region that forms a border between distally positioned structural genes and telomeres, but they can also be found elsewhere.
- **Telomeric sequences:** These consist of a short repeat of a sequence motif similar to TTTAGGG in tandem arrays many hundreds of units long at the physical end of each chromosome arm. The number of telomeric repeats is a species-specific characteristic ranging from 2–5 kbp in *Arabidopsis* to 60–160 kbp in tobacco [62]. Moreover, the number of copies of the repeat motif also differs among the chromosome arms for the same genome, and may even vary from cell to cell and tissue to tissue [78]. They are usually still unknown at the sequence level in most species sequenced to date since they are nearly impossible to assemble.

4. Confounding Factors for Plant Genome Assembly

The apparent disconnection between the limitations of sequencing technologies (several hundreds of base pairs per read in the better cases) and their successful application to genome projects (several hundreds of megabase pairs for small-genome plants) can be explained by the clever combination of sequencing and computation. The resulting reads of a sequencing run must be combined into a reconstruction of the original genome using a computer program called ‘assembler’. The assembler tries to construct a ‘superstring’ that contains all reads as ‘substrings’. It must be understood that different assemblers are needed for *de novo* genome assembly, transcriptome assembly, or genome resequencing (the different rationales for assemblers are beyond the scope of this article), so no assembler is suitable for all approaches. Assembly and analysis of raw sequence data requires substantial bioinformatic effort and expertise [79]. In spite of the fact that different sequencing goals will require different assemblers, the confounding factors emerging from the nature of plant genomes, which are discussed in the following sections, complicate any assembly of plant reads.

4.1. Repetitive Nature of Plant Genomes

Most of the challenging features of plant genomes discussed above produce some kind of repeats in DNA. Repeat sequences are difficult to assemble since high-identity reads could come from different portions of the genome, generating gaps, ambiguities and collapses in alignment and assembly, which, in turn, can produce biases and errors when interpreting results. Simply ignoring repeats is not an option, as this creates problems of its own and may mean that important biological phenomena are missed [50]. Repeats would be easily resolved if a single read could span a repeat instance with sufficient unique sequence on either side of the repeat. But repeats longer than the read length specifically create gaps in the assembly and can only be resolved if there are paired-ends that span the repeat instance. Nearly identical tandem repeats (>97% identity) are often collapsed into fewer copies, and it is difficult for an assembler to determine the true copy number since genomic regions that share

the same repeats can be indistinguishable, especially if the repeats are longer than the reads [50]. Inexact repeats (<95% identity) can be separated using high-stringency parameters. Repeats were not so critical in Sanger sequencing in which misassemblies and collapses occurred for only ~8% of the genome when duplications or repeats exceeded 95% sequence identity. Consequently, it is expected that repeats longer than 800 bp will suffer from the read-length methodology, regardless of whether it is NGS or Sanger [33]. It can be speculated that NGS short reads will have less power to resolve genomic repeats and require higher coverages to increase the chance of spanning short repeats. As a consequence, the most recent genome assemblies are much more fragmented than assemblies from a few years ago [51].

Repeat separation is assisted by high coverage but confounded by high sequencing error frequency: error tolerance leads to false positive joints that can induce chimeric assemblies, and this becomes especially problematic with reads from inexact (polymorphic) repeats. As a result, depletion of repeated sequences in assemblies becomes acute when the sequence identity exceeds 85%, resulting in the loss of ~16% of the genome [33], or ~5% of the genome being misassembled or missing [5]. The presence of duplicated and repetitive sequences in introns (a frequent event for genes in regions with >50% repetitive content) complicates complete gene assembly and annotation, leading to genes being broken among multiple sequence scaffolds: the more repetitive the region, the more scaffolds are obtained for the gene. After an assembly, nearly 70% of the genes are usually contained in single scaffolds [33], although exon shuffling is an artifact present in ~0.2% of those genes.

The current and most robust methods for overcoming the repeat issues when assembling shotgun reads are: (1) increasing the read length (in fact, nowadays, a compromise solution is to combine short reads with long reads), (2) producing paired-end reads longer than the repeated regions [12], and (3) correlating contigs with genetic maps and/or FISH. This can be easily seen with recently assembled potato [60], tomato [80] and melon [81] genomes. In conclusion, the day that sequencing platforms generate error-free reads at high coverage and assembly software can operate at 100% stringency, repeats would be resolved and a single superstring solution would be obtained. However, advances in the newer technologies based on single-molecule sequencing are giving longer reads (2,000–5,000 bp by now), which will clearly help in the resolution of long repetitive DNAs.

4.2. DNA Contamination

Plant nuclear DNA extractions are always contaminated with mitochondrial and chloroplast DNAs that can confound further assemblies since there always are homologous genes between organelle and nucleus DNA. Moreover, samples from, for example, plant roots where the rhizosphere is not easily removed, are usually highly contaminated with cells from other organisms; and these contaminating cells contain their own DNA, which is usually not of interest in the sequencing goals. Also, contamination can be introduced during laboratory manipulation (adaptors, vector, linkers, poly-A, *etc.*). Unfortunately contamination is especially difficult to discern when sequencing is based on short reads. In fact, it has been found that contaminating sequences are usually present in the targeted, species-specific sequences, mainly in those that do not match with any homologous sequence in databases [33]. Therefore, in order to obtain a reliable assembly of genomes or transcriptomes, any possible contamination or artifact-prone sequence must be removed with pre-processing software

(better than manual or in-house scripting methods), such as SeqTrimNext [82] (an evolution of SeqTrim fully prepared for NGS [83]). It must be taken into account, particularly in the case of genome assembly, that the phrase 'garbage in, garbage out' holds 100%, and that it can even be converted to 'garbage in, nothing out'. Reads devoid of any contamination are less cumbersome to assemble and less prone to misassembling, and produce more reliable consensus [84].

4.3. Sequencing Errors

If sequencing datasets were completely error-free, every read (substring) should be contained within a superstring. But real biological sequences are more complicated since error rates may be as high as 1–4% per nucleotide, implying that many reads contain mismatches with respect to the solution superstring [85]. For example, it has been reported that the Illumina sequencers result in sequence-specific miscalls, GC biased errors [86,87], and more substitution-type miscalls than indel-type miscalls [88]. Roche/454 sequencers produce more indel-type miscalls than substitution-type miscalls due to well-known homopolymer length inaccuracy concerns [89]. The newer technologies based on single-molecule sequencing have been reported to have a 5–15% error rate [90]. Error frequencies can explain the sequence coverage variability and the unfavorable bias observed in reads [91]. In practice, tolerance for sequencing errors makes it difficult to resolve a wide range of genomic phenomena, ranging from polymorphisms to paralogues.

4.4. Read Length

Shorter reads are inherent to NGS technologies and deliver less information per read, thus confounding the computational problem of assembly by hindering the detection of contamination, repeats or polymorphisms/errors. Short reads cannot be assembled using any typical overlap-layout-consensus algorithm [92] because the repetitive sequences are usually longer than the reads, so many reads cannot be unambiguously assigned, resulting in very short sequence contigs. This prompted the development of new bioinformatic approaches such as de Bruijn graphs combined with Eulerian paths [93,94], and the over-sampling of the target genome from random positions. Assemblies constructed from short-read datasets are highly fragmented and require long reads to increase their contiguity [60,80]. The assemblers mostly recommended for short reads are ALLPATHS-LG, SOAPdenovo and SGA, each one with its own pros and cons with respect to assembly length and consensus errors [95]. The advent of technologies based on single-molecule sequencing are now giving reads of 2,000–5,000 bp in length [90], which could simplify the assembling process in the near future.

4.5. Quality Values

The quality value (QV) of each called base was widely used for Sanger sequences assembling [96]. Since its use greatly increases CPU and RAM requirements, QVs are used only by a small set of NGS assemblers [92]. Consequently, to save time and computational resources, most current assemblers assume that base calls are reliable. The presence of low-quality reads will reduce the effective coverage and obscure true overlaps between sequencing reads, thus fragmenting the assembly and risking the collapse of more repeats. This reinforces the need for a good pre-processing of NGS reads

(e.g., using SeqTrimNext as explained before) to discard low QV fragments before assembly in order to avoid the assembling of inexistent sequences. For example, a 30 Gbp file of mate-pairs from HiSeq2000 could not be assembled within one week due to the presence of low quality nucleotides in the sequencer output; but this assembly was finished in four days in the same mainframe when reads were filtered for QV20 nucleotides [97].

4.6. Number of Reads and Coverage

Assembly is confounded by locations in which there are not enough overlapping reads to extend the sequence with confidence. It is easy to deduce that shorter read lengths will produce a larger number of gaps. The Lander-Waterman model offers a theoretical prediction of the minimum coverage needed to assemble large contigs depending on the read length [98]. For example, a three-fold (3×) coverage is sufficient when using Sanger technology, but a minimum of 15× coverage is required to assemble 100 bp reads into large contigs. However, considering the challenges depicted in the previous section, a minimum coverage of 7×–10× can work for Sanger technology, while 80×–100× is recommended in practice for short reads [32,33]. This high coverage will not resolve the concern about repeats but it is required to compensate the effective shorter length and sequencing errors of NGS technologies, which increase assembly complexity and intensify computational issues related to large datasets.

Short-read NGS technologies nowadays provide terabyte-sized data files, so coverage does not seem to be an issue, and previously intractable plant genomes (for example, pine genomes, which are seven- to 10-fold longer than the human genome and probably contain >95% repetitive sequences) are now feasible, at least in theory. Variation in coverage is introduced by chance, by variation of the copy number within DNA (*i.e.*, repeats), and by the technology *per se*. But when coverage is homogeneous along the genome, local biases can be interpreted as follows: Gaps are a consequence of very low coverage, and high-coverage is a diagnosis of an over-collapsed repeat. Unfortunately, coverage variability is the rule and undermines the coverage-based diagnostics. It can be speculated that the sequencing itself needs to be improved to reduce the biases, for example from GC composition and PCR, so that the coverage along the genome will be uniform and complete [99].

The overwhelming throughput of NGS raises a collateral issue related to data overload on a laboratory, institutional and community scale. In fact, the infrastructure costs for data storage, processing and handling are becoming more worrying than the costs of generating the reads. Since sequencing throughput is expected to increase in coming years, data storage and handling are becoming a real concern [14]. A more critical issue is computation: The comparison of each read with others required by the overlap-layout-consensus algorithms as well as the resolution of the Eulerian paths for de Bruijn graphs are the most time-consuming part of the assembling process. Therefore, the task could become never-ending or result in a faulty execution when temporary data do not fit in available RAM. The situation could arise that the right data and the right algorithm are available, but the right computer or software to hold calculations and memory are not. The most recent assemblers are focused on distributing among CPUs the processing load that cannot be managed with current serial algorithms. The de Bruijn graphs methods for assembly have the advantage of avoiding the all-*versus*-all comparisons, but their use is limited when there are too many errors or there is too low coverage, since they lead to infinite loops in the Eulerian paths that produce erroneous

'superstrings' [100]. In conclusion, the type of choices to be made for plant sequencing using NGS remain the same: The importance of assembly size should be balanced against the cost of sequencing, the bioinformatics resources available, and the time the research team has to devote to the project (as in Heisenberg's uncertainty principle, less costs and time in sequencing, more costs and time in assembling).

5. Seeking for the Best Assembly

When discussing plant genome assembly, it is important to distinguish between *de novo* approaches (where the aim is to reconstruct a new genome or transcriptome) and comparative approaches (referred to as mapping since the assembly uses a genome or transcriptome reference, or both). Mathematically, *de novo* assembly is such a difficult problem that, as yet, there is no efficient computational solution; in contrast, mapping is a much easier task. Neither approach is exclusive since after resequencing (mapping), there are always regions that differ significantly from the reference that can only be reconstructed through *de novo* assembly. Since *de novo* assemblies constructed from NGS technologies are highly fragmented, it has been proposed that a good genome assembly would have $N50_{contigs}$ > 30 kbp, $N50_{scaffolds}$ > 250 kbp, $N50_{super\text{-}scaffolds}$ > 1 Mbp, >90% of the genes represented (as measured by previous transcriptomics analyses), and >90% coverage of full-length cDNAs [14]. For now, it should be evident that the ability to assemble plant genome data is constrained by the absence of bioinformatics tools designed to cope with the challenging features present in all plant genomes. Hence, genome assembly is far from being a resolved problem, and the worst consequence is the probably unexpected, artifactual explosion of apparent lineage-specific genes leading to gross incongruities [1]. It is a fact that different transcriptomics projects contain 20–40% unigenes that do not have an orthologue in another plant (e.g., [41,42]). Besides the species-specific genes, the most part of these unigenes may represent 'garbage sequences' generated by errors within the amplification and/or sequencing technology. The percentage of this garbage will be known more precisely as more and more transcriptomes and genomes are reported. In the meantime, we have developed the bioinformatics tool Full-LengtherNext [101] that can inform which unigenes may be garbage or putative species-specific unigenes [66].

Many assemblers designed to handle Sanger reads were found to be impractical when dealing with NGS data. The response was to develop new assemblers employing qualitatively new approaches that seemed to be appropriate for assembly from human to *Arabidopsis* genomes (to cite a few, CABOG, Newbler, ABySS, SOAPdenovo or ALLPATHS), although their true success depends largely on the sophistication of their heuristics for real reads to solve the existing issues [12]. They generally require servers or clusters with >500 gigabytes of RAM and many terabytes of available disk space. The decrease in cost of servers, the emergence of supercomputing centers, and the development of cloud computing, mean that they are available at a negligible cost. But new sequencing projects such as loblolly pine [102] or maritime pine [103] with 22–30 Gbp genomes, are increasing the computational demands by nearly another order of magnitude, and no proven technology is available to resolve this assembly. Assembler performance was evaluated last year in a competitive framework with both simulated and real datasets of small, simple genomes. Results confirmed that the final sequences were highly dependent on the assembler and pipeline used [95], although it can be said that assemblers for

long reads produce longer contigs and scaffolds with more indels and underrepresentation of repeats, while the de Bruijn-based assemblers include shorter contigs and scaffolds, more mismatches and the highest representation for repeat regions [34]. Most assemblies nowadays rely on one single assembler, but as different assemblers use different underlying algorithms, combining different optimal assemblies from different programs can give a more credible final assembly [104]. The combination usually increased the N50 and median contig size, mapped more original reads, and diminished the final number of contigs/scaffolds. This strategy is currently used for transcriptomes, and CAP3 [96] or Minimus [105] are good candidates for the second assembly process [106–108]. In the case of genome assembly, mammalian genomes have recently been assembled using this combined strategy [109], running SOAPdenovo and ABySS separately, and then combining the assembly with GAP5 to generate the final consensus sequences.

As the choices made at the beginning of any study will determine the degree of success of the sequencing project, it can be concluded that there is a strong need to develop plant-specific assemblers that can overcome the challenges of these genomes; moreover, new software should expend efforts in producing user-friendly interfaces since most bioinformatics projects are developing software tailored to their needs, which leads to the same software being reinvented over and over again by different research groups [79]. Researchers have to decide which plant genome will be sequenced, which NGS technology will be applied, and which assembling approach should be used. In Plantagora [34], researchers can find a substantial body of information for comparing different approaches to sequencing a plant genome, providing a platform of metrics and tools for studying the process of sequencing and assembling that can aid in the critical decision-making required for planning a plant-sequencing project.

6. Concluding Remarks

Plant genome sequencing is a long way away from automatic sequencing and assembly providing a completely finished genome at low cost. At the moment, we are able to afford the reconstruction of complex plant genomes into highly useful drafts. The need remains for an assembler that can deal with the plant genome features that challenge sequencing and assembly, *i.e.*, mainly large, repetitive genomes; moreover, incremental algorithms that can update the assembly as new data become available are also desirable. To circumvent the bioinformatics bottleneck in the near future, efforts should be invested in (1) parallelization of the assembly process, which has been shyly approached with ABySS [110] and ALLPATHS-LG [109]; (2) processing speed and storage capacity of computers; and (3) developing a new sequencing platform that can provide longer reads with unbiased coverage that can overcome the complex repeats. This last point refers to the so-called third-generation sequencing based on single-molecule sequencing, which is very promising with reads of 2,000–5,000 nt [90]. However, these technologies are relatively immature for immediate widespread application to plant genomes since to date an error rate of 5–15% has been reported.

Acknowledgments

The authors would like to acknowledge the computer resources of the Plataforma Andaluza de Bioinformática of the University of Málaga, Spain. This study was funded by Spanish MICINN (BIO2009-07490) and Junta de Andalucía (P10-CVI-6075 and BIO-114).

References

1. Paterson, A.H.; Freeling, M.; Tang, H.; Wang, X. Insights from the comparison of plant genome sequences. *Annu. Rev. Plant Biol.* **2010**, *61*, 349–372.
2. Sterck, L.; Rombauts, S.; Vandepoele, K.; Rouze, P.; van de Peer, Y. How many genes are there in plants (... and why are they there)? *Curr. Opin. Plant Biol.* **2007**, *10*, 199–203.
3. Gregory, T.R. The C-value enigma in plants and animals: A review of parallels and an appeal for partnership. *Ann. Bot.* **2005**, *95*, 133–146.
4. Arabidopsis Genome, I. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature* **2000**, *408*, 796–815.
5. Feuillet, C.; Leach, J.E.; Rogers, J.; Schnable, P.S.; Eversole, K. Crop genome sequencing: Lessons and rationales. *Trends Plant Sci.* **2011**, *16*, 77–88.
6. International Rice Genome Sequencing, P. The map-based sequence of the rice genome. *Nature* **2005**, *436*, 793–800.
7. Ming, R.; Hou, S.; Feng, Y.; Yu, Q.; Dionne-Laporte, A.; Saw, J.H.; Senin, P.; Wang, W.; Ly, B.V.; Lewis, K.L.; *et al.* The draft genome of the transgenic tropical fruit tree papaya (*Carica papaya Linnaeus*). *Nature* **2008**, *452*, 991–996.
8. Schnable, P.S.; Ware, D.; Fulton, R.S.; Stein, J.C.; Wei, F.; Pasternak, S.; Liang, C.; Zhang, J.; Fulton, L.; Graves, T.A.; *et al.* The B73 maize genome: Complexity, diversity, and dynamics. *Science* **2009**, *326*, 1112–1115.
9. Duvick, J.; Fu, A.; Muppirala, U.; Sabharwal, M.; Wilkerson, M.D.; Lawrence, C.J.; Lushbough, C.; Brendel, V. PlantGDB: A resource for comparative plant genomics. *Nucleic Acids Res.* **2008**, *36*, D959–D965.
10. Varshney, R.K.; Close, T.J.; Singh, N.K.; Hoisington, D.A.; Cook, D.R. Orphan legume crops enter the genomics era! *Curr. Opin. Plant Biol.* **2009**, *12*, 202–210.
11. Armstead, I.; Huang, L.; Ravagnani, A.; Robson, P.; Ougham, H. Bioinformatics in the orphan crops. *Brief. Bioinform.* **2009**, *10*, 645–653.
12. Imelfort, M.; Edwards, D. *De novo* sequencing of plant genomes using second-generation technologies. *Brief. Bioinform.* **2009**, *10*, 609–618.
13. Goodstein, D.M.; Shu, S.; Howson, R.; Neupane, R.; Hayes, R.D.; Fazo, J.; Mitros, T.; Dirks, W.; Hellsten, U.; Putnam, N.; *et al.* Phytozome: A comparative platform for green plant genomics. *Nucleic Acids Res.* **2012**, *40*, D1178–D1186.
14. Hamilton, J.P.; Buell, C.R. Advances in plant genome sequencing. *Plant J.* **2012**, *70*, 177–190.
15. Proost, S.; Pattyn, P.; Gerats, T.; van de Peer, Y. Journey through the past: 150 million years of plant genome evolution. *Plant J.* **2011**, *66*, 58–65.

16. Ossowski, S.; Schneeberger, K.; Clark, R.M.; Lanz, C.; Warthmann, N.; Weigel, D. Sequencing of natural strains of *Arabidopsis thaliana* with short reads. *Genome Res.* **2008**, *18*, 2024–2033.
17. Springer, N.M.; Ying, K.; Fu, Y.; Ji, T.; Yeh, C.T.; Jia, Y.; Wu, W.; Richmond, T.; Kitzman, J.; Rosenbaum, H.; *et al.* Maize inbreds exhibit high levels of copy number variation (CNV) and presence/absence variation (PAV) in genome content. *PLoS Genet.* **2009**, *5*, e1000734.
18. Morgante, M.; de Paoli, E.; Radovic, S. Transposable elements and the plant pan-genomes. *Curr. Opin. Plant Biol.* **2007**, *10*, 149–155.
19. Plant Genomes Central. Available online: http://www.ncbi.nlm.nih.gov/genomes/PLANTS/PlantList.html (accessed on 14 September 2012).
20. List of Sequenced Plant Genomes. Available online: http://en.wikipedia.org/wiki/List_of_sequenced_plant_genomes (accessed on 14 September 2012).
21. Sanger, F.; Nicklen, S.; Coulson, A.R. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. USA* **1977**, *74*, 5463–5467.
22. Bräutigam, A.; Gowik, U. What can next generation sequencing do for you? Next generation sequencing as a valuable tool in plant research. *Plant Biol. (Stuttg)* **2010**, *12*, 831–841.
23. Goff, S.A.; Ricke, D.; Lan, T.H.; Presting, G.; Wang, R.; Dunn, M.; Glazebrook, J.; Sessions, A.; Oeller, P.; Varma, H.; *et al.* A draft sequence of the rice genome (*Oryza sativa* L. ssp. *japonica*). *Science* **2002**, *296*, 92–100.
24. Yu, J.; Hu, S.; Wang, J.; Wong, G.K.; Li, S.; Liu, B.; Deng, Y.; Dai, L.; Zhou, Y.; Zhang, X.; *et al.* A draft sequence of the rice genome (*Oryza sativa* L. ssp. *indica*). *Science* **2002**, *296*, 79–92.
25. Shendure, J.; Ji, H. Next-generation DNA sequencing. *Nat. Biotechnol.* **2008**, *26*, 1135–1145.
26. Mardis, E.R. Next-generation DNA sequencing methods. *Annu. Rev. Genomics Hum. Genet.* **2008**, *9*, 387–402.
27. Ansorge, W.J. Next-generation DNA sequencing techniques. *N. Biotechnol.* **2009**, *25*, 195–203.
28. Kircher, M.; Kelso, J. High-throughput DNA sequencing—Concepts and limitations. *Bioessays* **2010**, *32*, 524–536.
29. Zhou, X.; Ren, L.; Meng, Q.; Li, Y.; Yu, Y.; Yu, J. The next-generation sequencing technology and application. *Protein Cell* **2010**, *1*, 520–536.
30. Niedringhaus, T.P.; Milanova, D.; Kerby, M.B.; Snyder, M.P.; Barron, A.E. Landscape of next-generation sequencing technologies. *Anal. Chem.* **2011**, *83*, 4327–4341.
31. Pareek, C.S.; Smoczynski, R.; Tretyn, A. Sequencing technologies and genome sequencing. *J. Appl. Genet.* **2011**, *52*, 413–435.
32. Finotello, F.; Lavezzo, E.; Fontana, P.; Peruzzo, D.; Albiero, A.; Barzon, L.; Falda, M.; di Camillo, B.; Toppo, S. Comparative analysis of algorithms for whole-genome assembly of pyrosequencing data. *Brief. Bioinform.* **2012**, *13*, 269–280.
33. Alkan, C.; Sajjadian, S.; Eichler, E.E. Limitations of next-generation genome sequence assembly. *Nat. Methods* **2011**, *8*, 61–65.
34. Barthelson, R.; McFarlin, A.J.; Rounsley, S.D.; Young, S. Plantagora: Modeling whole genome sequencing and assembly of plant genomes. *PLoS One* **2011**, *6*, e28436.
35. Wang, L.; Li, P.; Brutnell, T.P. Exploring plant transcriptomes using ultra high-throughput sequencing. *Brief. Funct. Genomics* **2010**, *9*, 118–128.

36. Vandepoele, K.; Quimbaya, M.; Casneuf, T.; de Veylder, L.; van de Peer, Y. Unraveling transcriptional control in *Arabidopsis* using *cis*-regulatory elements and coexpression networks. *Plant Physiol.* **2009**, *150*, 535–546.
37. He, F.; Zhou, Y.; Zhang, Z. Deciphering the *Arabidopsis* floral transition process by integrating a protein-protein interaction network and gene expression data. *Plant Physiol.* **2010**, *153*, 1492–1505.
38. Alvarez, J.M.; Vidal, E.A.; Gutierrez, R.A. Integration of local and systemic signaling pathways for plant N responses. *Curr. Opin. Plant Biol.* **2012**, *15*, 185–191.
39. Proost, S.; van Bel, M.; Sterck, L.; Billiau, K.; van Parys, T.; van de Peer, Y.; Vandepoele, K. PLAZA: A comparative genomics resource to study gene and genome evolution in plants. *Plant Cell* **2009**, *21*, 3718–3731.
40. Wegrzyn, J.L.; Lee, J.M.; Tearse, B.R.; Neale, D.B. TreeGenes: A forest tree genome database. *Int. J. Plant Genomics* **2008**, *2008*, 412875.
41. Fernandez-Pozo, N.; Canales, J.; Guerrero-Fernandez, D.; Villalobos, D.P.; Diaz-Moreno, S.M.; Bautista, R.; Flores-Monterroso, A.; Guevara, M.A.; Perdiguero, P.; Collada, C.; *et al.* EuroPineDB: A high-coverage web database for maritime pine transcriptome. *BMC Genomics* **2011**, *12*, 366.
42. Rengel, D.; San Clemente, H.; Servant, F.; Ladouce, N.; Paux, E.; Wincker, P.; Couloux, A.; Sivadon, P.; Grima-Pettenati, J. A new genomic resource dedicated to wood formation in *Eucalyptus*. *BMC Plant Biol.* **2009**, *9*, 36.
43. Gonzalez-Ibeas, D.; Blanca, J.; Roig, C.; Gonzalez-To, M.; Pico, B.; Truniger, V.; Gomez, P.; Deleu, W.; Cano-Delgado, A.; Arus, P.; *et al.* MELOGEN: An EST database for melon functional genomics. *BMC Genomics* **2007**, *8*, 306.
44. Goff, S.A.; Vaughn, M.; McKay, S.; Lyons, E.; Stapleton, A.E.; Gessler, D.; Matasci, N.; Wang, L.; Hanlon, M.; Lenards, A.; *et al.* The iPlant collaborative: Cyberinfrastructure for plant biology. *Front. Plant Sci.* **2011**, *2*, 34.31–34.16.
45. Katari, M.S.; Nowicki, S.D.; Aceituno, F.F.; Nero, D.; Kelfer, J.; Thompson, L.P.; Cabello, J.M.; Davidson, R.S.; Goldberg, A.P.; Shasha, D.E.; *et al.* VirtualPlant: A software platform to support systems biology research. *Plant Physiol.* **2010**, *152*, 500–515.
46. Lapitan, N.L.V. Organization and evolution of higher plant nuclear genome. *Genome* **1992**, *35*, 171–181.
47. Janicki, M.; Rooke, R.; Yang, G. Bioinformatics and genomic analysis of transposable elements in eukaryotic genomes. *Chromosome Res.* **2011**, *19*, 787–808.
48. Wicker, T.; Sabot, F.; Hua-Van, A.; Bennetzen, J.L.; Capy, P.; Chalhoub, B.; Flavell, A.; Leroy, P.; Morgante, M.; Panaud, O.; *et al.* A unified classification system for eukaryotic transposable elements. *Nat. Rev. Genet.* **2007**, *8*, 973–982.
49. Bousios, A.; Darzentas, N.; Tsaftaris, A.; Pearce, S.R. Highly conserved motifs in non-coding regions of Sirevirus retrotransposons: The key for their pattern of distribution within and across plants? *BMC Genomics* **2010**, *11*, 89.
50. Treangen, T.J.; Salzberg, S.L. Repetitive DNA and next-generation sequencing: Computational challenges and solutions. *Nat. Rev. Genet.* **2012**, *13*, 36–46.

51. Schatz, M.C.; Delcher, A.L.; Salzberg, S.L. Assembly of large genomes using second-generation sequencing. *Genome Res.* **2010**, *20*, 1165–1173.
52. Hochholdinger, F.; Hoecker, N. Towards the molecular basis of heterosis. *Trends Plant Sci.* **2007**, *12*, 427–432.
53. Tuskan, G.A.; Difazio, S.; Jansson, S.; Bohlmann, J.; Grigoriev, I.; Hellsten, U.; Putnam, N.; Ralph, S.; Rombauts, S.; Salamov, A.; *et al.* The genome of black cottonwood, *Populus trichocarpa* (Torr. & Gray). *Science* **2006**, *313*, 1596–1604.
54. Jaillon, O.; Aury, J.M.; Noel, B.; Policriti, A.; Clepet, C.; Casagrande, A.; Choisne, N.; Aubourg, S.; Vitulo, N.; Jubin, C.; *et al.* The grapevine genome sequence suggests ancestral hexaploidization in major angiosperm phyla. *Nature* **2007**, *449*, 463–467.
55. Kelley, D.R.; Salzberg, S.L. Detection and correction of false segmental duplications caused by genome mis-assembly. *Genome Biol.* **2010**, *11*, R28.
56. Comai, L. The advantages and disadvantages of being polyploid. *Nat. Rev. Genet.* **2005**, *6*, 836–846.
57. Meyers, L.A.; Levin, D.A. On the abundance of polyploids in flowering plants. *Evolution* **2006**, *60*, 1198–1206.
58. Bento, M.; Gustafson, J.P.; Viegas, W.; Silva, M. Size matters in *Triticeae* polyploids: Larger genomes have higher remodeling. *Genome* **2011**, *54*, 175–183.
59. Tang, H.; Bowers, J.E.; Wang, X.; Ming, R.; Alam, M.; Paterson, A.H. Synteny and collinearity in plant genomes. *Science* **2008**, *320*, 486–488.
60. Potato Genome Sequencing, C.; Xu, X.; Pan, S.; Cheng, S.; Zhang, B.; Mu, D.; Ni, P.; Zhang, G.; Yang, S.; Li, R.; *et al.* Genome sequence and analysis of the tuber crop potato. *Nature* **2011**, *475*, 189–195.
61. Shulaev, V.; Sargent, D.J.; Crowhurst, R.N.; Mockler, T.C.; Folkerts, O.; Delcher, A.L.; Jaiswal, P.; Mockaitis, K.; Liston, A.; Mane, S.P.; *et al.* The genome of woodland strawberry (*Fragaria vesca*). *Nat. Genet.* **2011**, *43*, 109–116.
62. Heslop-Harrison, J.S. Comparative genome organization in plants: From sequence and markers to chromatin and chromosomes. *Plant Cell* **2000**, *12*, 617–636.
63. Giussani, L.M.; Cota-Sanchez, J.H.; Zuloaga, F.O.; Kellogg, E.A. A molecular phylogeny of the grass subfamily *Panicoideae* (*Poaceae*) shows multiple origins of C4 photosynthesis. *Am. J. Bot.* **2001**, *88*, 1993–2012.
64. Sappl, P.G.; Heazlewood, J.L.; Millar, A.H. Untangling multi-gene families in plants by integrating proteomics into functional genomics. *Phytochemistry* **2004**, *65*, 1517–1530.
65. Duarte, J.M.; Cui, L.; Wall, P.K.; Zhang, Q.; Zhang, X.; Leebens-Mack, J.; Ma, H.; Altman, N.; dePamphilis, C.W. Expression pattern shifts following duplication indicative of subfunctionalization and neofunctionalization in regulatory genes of *Arabidopsis*. *Mol. Biol. Evol.* **2006**, *23*, 469–478.
66. Fernández-Pozo, N.; Guerrero-Fernández, D.; Bautista, R.; Claros, M.G. Full-LengtherNext: A tool for fine-tuning *de novo* assembled transcriptomes of non-model organisms. Departamento de Biología Molecular y Bioquímica, Facultad de Ciencias, Universidad de Málaga, 29071 Málaga, Spain, and Plataforma Andaluza de Bioinformática, Centro de Supercomputación y

Bioinformática, Edificio de Bioinnovación, Universidad de Málaga, 29590 Málaga, Spain. Unpublished work, to be submitted for publication, 2012.

67. Phillippy, A.M.; Schatz, M.C.; Pop, M. Genome assembly forensics: Finding the elusive mis-assembly. *Genome Biol.* **2008**, *9*, R55.
68. Lai, J.; Li, Y.; Messing, J.; Dooner, H.K. Gene movement by Helitron transposons contributes to the haplotype variability of maize. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 9068–9073.
69. Freeling, M.; Lyons, E.; Pedersen, B.; Alam, M.; Ming, R.; Lisch, D. Many or most genes in *Arabidopsis* transposed after the origin of the order *Brassicales*. *Genome Res.* **2008**, *18*, 1924–1937.
70. Lindbo, J.A.; Silva-Rosales, L.; Proebsting, W.M.; Dougherty, W.G. Induction of a highly specific antiviral state in transgenic plants: Implications for regulation of gene expression and virus resistance. *Plant Cell* **1993**, *5*, 1749–1759.
71. Huang, R.; Jaritz, M.; Guenzl, P.; Vlatkovic, I.; Sommer, A.; Tamir, I.M.; Marks, H.; Klampfl, T.; Kralovics, R.; Stunnenberg, H.G.; *et al.* An RNA-Seq strategy to detect the complete coding and non-coding transcriptome including full-length imprinted macro ncRNAs. *PLoS One* **2011**, *6*, e27288.
72. Carninci, P.; Kasukawa, T.; Katayama, S.; Gough, J.; Frith, M.C.; Maeda, N.; Oyama, R.; Ravasi, T.; Lenhard, B.; Wells, C.; *et al.* The transcriptional landscape of the mammalian genome. *Science* **2005**, *309*, 1559–1563.
73. Gore, M.A.; Chia, J.M.; Elshire, R.J.; Sun, Q.; Ersoz, E.S.; Hurwitz, B.L.; Peiffer, J.A.; McMullen, M.D.; Grills, G.S.; Ross-Ibarra, J.; *et al.* A first-generation haplotype map of maize. *Science* **2009**, *326*, 1115–1117.
74. Wang, X.; Tang, H.; Bowers, J.E.; Paterson, A.H. Comparative inference of illegitimate recombination between rice and sorghum duplicated genes produced by polyploidization. *Genome Res.* **2009**, *19*, 1026–1032.
75. Pruitt, R.E.; Meyerowitz, E.M. Characterization of the genome of *Arabidopsis thaliana*. *J. Mol. Biol.* **1986**, *187*, 169–183.
76. Murata, M.; Ogura, Y.; Motoyoshi, F. Centromeric repetitive sequences in *Arabidopsis thaliana*. *Jpn. J. Genet.* **1994**, *69*, 361–371.
77. Horáková, M.; Fajkus, J. TAS4—A dispersed repetitive sequence isolated from subtelomeric regions of *Nicotiana tomentosiformis* chromosomes. *Genome* **2000**, *43*, 273–284.
78. Kilian, A.; Stiff, C.; Kleinhofs, A. Barley telomeres shorten during differentiation but grow in callus culture. *Proc. Natl. Acad. Sci. USA* **1995**, *92*, 9555–9559.
79. Schatz, M.C.; Witkowski, J.; McCombie, W.R. Current challenges in de novo plant genome sequencing and assembly. *Genome Biol.* **2012**, *13*, 243.
80. Tomato Genome, C. The tomato genome sequence provides insights into fleshy fruit evolution. *Nature* **2012**, *485*, 635–641.
81. Garcia-Mas, J.; Benjak, A.; Sanseverino, W.; Bourgeois, M.; Mir, G.; González, V.M.; Hénaff, E.; Cámara, F.; Cozzuto, L.; Lowy, E.; *et al.* The genome of melon (*Cucumis melo* L.). *Proc. Natl. Acad. Sci. USA* **2012**, in press.
82. SeqTrimNext. Available online: http://www.scbi.uma.es/seqtrimnext (accessed on 14 September 2012).

83. Falgueras, J.; Lara, A.J.; Fernandez-Pozo, N.; Canton, F.R.; Perez-Trabado, G.; Claros, M.G. SeqTrim: A high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics* **2010**, *11*, 38.
84. Guerrero-Fernaández, D.; Falgueras, J.; Claros, M.G. SCBI_MAPREDUCE: A task-farm, practical solution in Ruby for distribution of new and legacy bioinformatics software. *IEEE Trans. Parallel. Distr. Syst*. **2012**, submitted for publication.
85. Paszkiewicz, K.; Studholme, D.J. *De novo* assembly of short sequence reads. *Brief. Bioinform.* **2010**, *11*, 457–472.
86. Nakamura, K.; Oshima, T.; Morimoto, T.; Ikeda, S.; Yoshikawa, H.; Shiwa, Y.; Ishikawa, S.; Linak, M.C.; Hirai, A.; Takahashi, H.; *et al.* Sequence-specific error profile of Illumina sequencers. *Nucleic Acids Res.* **2011**, *39*, e90.
87. Minoche, A.E.; Dohm, J.C.; Himmelbauer, H. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and genome analyzer systems. *Genome Biol.* **2011**, *12*, R112.
88. Hoffmann, S.; Otto, C.; Kurtz, S.; Sharma, C.M.; Khaitovich, P.; Vogel, J.; Stadler, P.F.; Hackermuller, J. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Comput. Biol.* **2009**, *5*, e1000502.
89. Gilles, A.; Meglecz, E.; Pech, N.; Ferreira, S.; Malausa, T.; Martin, J.F. Accuracy and quality assessment of 454 GS-FLX Titanium pyrosequencing. *BMC Genomics* **2011**, *12*, 245.
90. Rasko, D.A.; Webster, D.R.; Sahl, J.W.; Bashir, A.; Boisen, N.; Scheutz, F.; Paxinos, E.E.; Sebra, R.; Chin, C.S.; Iliopoulos, D.; *et al.* Origins of the *E. coli* strain causing an outbreak of hemolytic-uremic syndrome in Germany. *N. Engl. J. Med.* **2011**, *365*, 709–717.
91. Balzer, S.; Malde, K.; Jonassen, I. Systematic exploration of error sources in pyrosequencing flowgram data. *Bioinformatics* **2011**, *27*, i304–309.
92. Miller, J.R.; Koren, S.; Sutton, G. Assembly algorithms for next-generation sequencing data. *Genomics* **2010**, *95*, 315–327.
93. Medvedev, P.; Pham, S.; Chaisson, M.; Tesler, G.; Pevzner, P. Paired de bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. *J. Comput. Biol.* **2011**, *18*, 1625–1634.
94. Compeau, P.E.; Pevzner, P.A.; Tesler, G. How to apply de Bruijn graphs to genome assembly. *Nat. Biotechnol.* **2011**, *29*, 987–991.
95. Earl, D.; Bradnam, K.; St. John, J.; Darling, A.; Lin, D.; Fass, J.; Yu, H.O.; Buffalo, V.; Zerbino, D.R.; Diekhans, M.; *et al.* Assemblathon 1: A competitive assessment of *de novo* short read assembly methods. *Genome Res.* **2011**, *21*, 2224–2241.
96. Huang, X.; Madan, A. CAP3: A DNA sequence assembly program. *Genome Res.* **1999**, *9*, 868–877.
97. Benzekri, H.; Bautista, R.; Guerrero-Fernández, D.; Claros, M.G. Departamento de Biología Molecular y Bioquímica, Facultad de Ciencias, Universidad de Málaga, 29071 Málaga, Spain, and Plataforma Andaluza de Bioinformática, Centro de Supercomputación y Bioinformática, Edificio de Bioinnovación, Universidad de Málaga, 29590 Málaga, Spain. Unpublished work, 2012.

98. Lander, E.S.; Waterman, M.S. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics* **1988**, *2*, 231–239.
99. Aird, D.; Ross, M.G.; Chen, W.S.; Danielsson, M.; Fennell, T.; Russ, C.; Jaffe, D.B.; Nusbaum, C.; Gnirke, A. Analyzing and minimizing PCR amplification bias in Illumina sequencing libraries. *Genome Biol.* **2011**, *12*, R18.
100. Li, Z.; Chen, Y.; Mu, D.; Yuan, J.; Shi, Y.; Zhang, H.; Gan, J.; Li, N.; Hu, X.; Liu, B.; *et al.* Comparison of the two major classes of assembly algorithms: Overlap-layout-consensus and de Bruijn-graph. *Brief. Funct. Genomics* **2012**, *11*, 25–37.
101. FullLengtherNext. Available online: http://www.scbi.uma.es/fulllengthernext (accessed on 14 September 2012).
102. Loblolly Pine Genome Project. Available online: http://dendrome.ucdavis.edu/NealeLab/lpgp/ (accessed on 14 September 2012).
103. Díaz-Sala, C.; Cervera, M. Promoting a functional and comparative understanding of the conifer genome-implementing applied aspects for more productive and adapted forests (ProCoGen). *BCM Proceedings* **2011**, *5*, P158.
104. Kumar, S.; Blaxter, M.L. Comparing de novo assemblers for 454 transcriptome data. *BMC Genomics* **2010**, *11*, 571.
105. Sommer, D.D.; Delcher, A.L.; Salzberg, S.L.; Pop, M. Minimus: A fast, lightweight genome assembler. *BMC Bioinformatics* **2007**, *8*, 64.
106. Zheng, Y.; Zhao, L.; Gao, J.; Fei, Z. iAssembler: A package for de novo assembly of Roche-454/Sanger transcriptome sequences. *BMC Bioinformatics* **2011**, *12*, 453.
107. Iorizzo, M.; Senalik, D.A.; Grzebelus, D.; Bowman, M.; Cavagnaro, P.F.; Matvienko, M.; Ashrafi, H.; van Deynze, A.; Simon, P.W. *De novo* assembly and characterization of the carrot transcriptome reveals novel genes, new markers, and genetic diversity. *BMC Genomics* **2011**, *12*, 389.
108. Martin, J.; Bruno, V.M.; Fang, Z.; Meng, X.; Blow, M.; Zhang, T.; Sherlock, G.; Snyder, M.; Wang, Z. Rnnotator: An automated de novo transcriptome assembly pipeline from stranded RNA-Seq reads. *BMC Genomics* **2010**, *11*, 663.
109. Gnerre, S.; Maccallum, I.; Przybylski, D.; Ribeiro, F.J.; Burton, J.N.; Walker, B.J.; Sharpe, T.; Hall, G.; Shea, T.P.; Sykes, S.; *et al.* High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. USA* **2011**, *108*, 1513–1518.
110. Simpson, J.T.; Wong, K.; Jackman, S.D.; Schein, J.E.; Jones, S.J.; Birol, I. ABySS: A parallel assembler for short read sequence data. *Genome Res.* **2009**, *19*, 1117–1123.

Detecting and correcting mis-assembled reads in contigs

Hicham Benzekri[1], Darío Guerrero-Fernández[1], Rocío Bautista[1], and M. Gonzalo Claros[1,2,*]

[1] Plataforma Andaluza de Bioinformática-SCBI, University of Málaga,
C/ Severo Ochoa 34, 29590 Málaga, Spain
{bhicham,dariogf,rociobm,claros}@uma.es
http://www.scbi.uma.es

[2] Molecular Biology and Biochemistry Department, University of Málaga,
Campus de Teatinos s/n, 29071 Málaga, Spain
http://www.bmbq.uma.es/fmp

Abstract. *De novo* assemblies do not have the possibility of quality control with an external sequence. In fact, accuracy and reliability of these assemblies is highly affected by sequencing errors and mis-assemblies. Here, a frequency-based algorithm is developed in Ruby and intended to discern assembly errors from polymorphisms/read errors and then edit or remove the misassembled read(s) to provide more but highly reliable contigs. The software reads and writes the ACE assembly format. Transcriptome and genome assemblies were tested.

Keywords: contigs, OLC, de novo, assembly.

1 Introduction

Sequence assembly errors exist in any *de novo* assembly. Identification of mis-assemblies is a difficult issue due to the high amount of data and its error-prone quality because of biochemical and mechanical complications in sequencers. This usually requires additional efforts for manual validation of the most accurate reconstruction of the analyzed genome or transcriptome [1]. Too often, assembly quality is judged only by contig size or N50, with larger contigs being preferred [2], even though large contigs can be chimeric as a result of mis-assembling.

A widely-used contig-testing tool is *Hawkeye* [3]. It can be used with assemblies of all sizes to facilitate the visual inspection of large-scale assembly data while minimizing the time needed to detect mis-assemblies and make accurate judgments for assembly quality. In fact, it guides users to the most likely areas of mis-assembly, allowing its manual edition and correction. In contrast to other contig editors such as GAP5 [4], *Hawkeye* combines computational predictors with interactive visualizations to decrease verification costs. However, visual inspection and manual edition are cumbersome tasks, particularly for assemblies from next-generation sequencing (NGS) data. This is the reason why *amosvalidate* [2], an automated

* Corresponding author

validation pipeline for contigs based on several independent criteria, was developed. But this tool only tagged regions that appear mis-assembled, and the correction requires visualization again and manual edition with *Hawkeye*.

The aim of our work is to develop a fully automated algorithm called CoMiner with the aim of editing and correcting prominent mismatches in contigs, reducing the manual intervention dedicated to increase the quality of assemblies, based only on the contig assembly *per se*.

2 Implementation

CoMiner was programmed in Ruby and tested in a dual core iMac at 3.06 GHz with 4 GB of RAM. Contig data can be read and written in ACE format [5], which is generated by various assembly programs, such as Phrap, CAP3, GAP4-5, Newbler, Arachne, Minimus and TIGR Assembler, all of them based on overlay-layout-consensus algorithms (CoMiner is not ready for analyzing De Bruijn contigs, but could be adapted for mapping alignments in a near future).

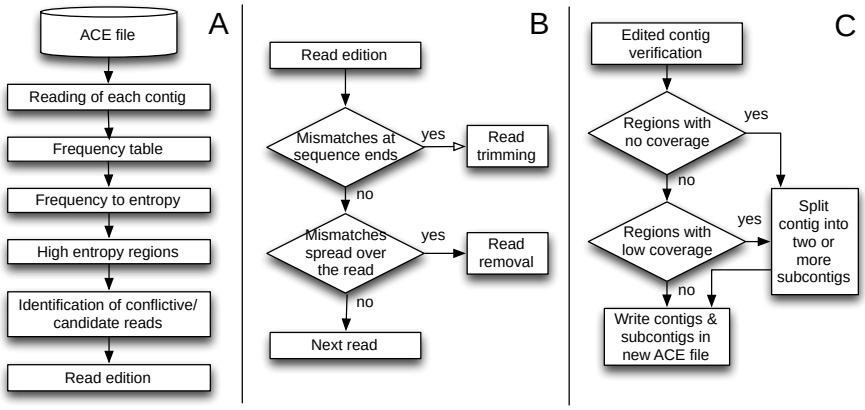


Fig. 1. Flowgram of CoMiner algorithm. A: Discovery of high entropy regions and identification of conflictive reads. B: Conflictive-read edition/removal within a contig depending on mismatch distribution. C: Once all conflictive reads in a contig have been edited, the contig coverage is verified, split in two or more contigs if necessary, and then saved into a new ACE file.

Since the final goal of CoMiner is to increase the assembly accuracy without human intervention, the algorithm (Fig. 1) can be divided in the following main steps: (i) discovery of high-entropy regions (HERs); (ii) identification of conflictive read(s); (iii) read edition (trimming or removal); (iv) contig verification and saving in a new ACE file.

(i) *HER discovery*: The aim of this step is to retrieve assembly fragments where reads do not align perfectly. We have elected the entropy of consensus nucleotide at position i [$-H(i)$] as a measure of the alignment goodness as described in [6]. Therefore, the frequency of each of the four nucleotides at each position of the

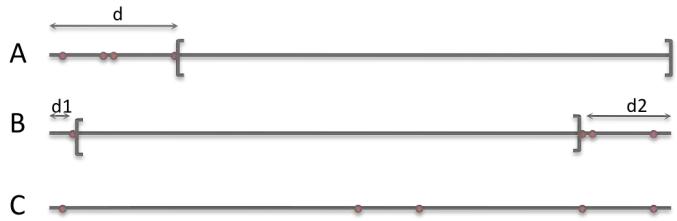


Fig. 2. Different instances of mismatch distribution in a conflictive read. A: All mismatches are located at one end; therefore, nucleotides within the distance *d* were trimmed from the read, provided that *d* < 40% of the read length, and the remaining read is longer than 40 nt. B: There are mismatches at both ends; again, nucleotides within distances *d1* and *d2* are trimmed provided that *d1* + *d2* < 40% of the read length, and the remaining read is longer than 40 nt. C: Mismatches are spread over the whole read; when the number of mismatches is over 2% of the read length, the complete read is removed.

consensus sequence is assessed and then used to calculate entropy at each consensus position. SNPs and point sequence are considered equivalent events in this rationale, and do not significantly affect assemblies unless they are closely located. This is the reason why entropy data were sieved by a fast Fourier transform as described [6] for converting contiguous sharp peaks into high entropy regions. Sequence ranges whose Fourier-transformed entropy is over a cutoff value that corresponds to the median entropy of the contig will be considered HERs and will focus subsequent analyses.

(ii) *Identification of conflictive-read(s)*: Several calculations are performed to determine whether a HER was caused by one or more mis-assembled reads or whether it was caused by mismatches scattered over all involved reads. These possibilities are discerned calculating the mismatch frequency of one read *r* [$F_{read}(r)$] as follows: for a contig containing *m* number of reads for which a *k* number of HERs have been defined, *n(j)* being the length of one HER, $F_{read}(r)$ is calculated dividing the total number of mismatches of the read *r* within all HERs against the consensus by the total number of nucleotides involved in all HERs:

$$F_{read}(r) = \sum_{j=1}^{k}\sum_{i=1}^{n} err(i,j) \Big/ \sum_{j=1}^{k} n(j)$$

The total mismatch frequency of the contig (F_{contig}) is calculated dividing the total number of mismatches of every contig read within all HERs by the total length of all HER regions in each read, as follows:

$$F_{contig} = \sum_{r=1}^{m}\sum_{j=1}^{k}\sum_{i=1}^{n} err(i,j,r) \Big/ \sum_{r=1}^{m}\sum_{j=1}^{k} n(j,r)$$

Both $F_{read}(r)$ and F_{contig} will define a robust F_{cutoff} value as:

$$F_{cutoff} = F_{contig} + K \times \left(\sum_{r=1}^{m} \left| F_{read}(r) - F_{contig} \right| \Big/ m \right)$$

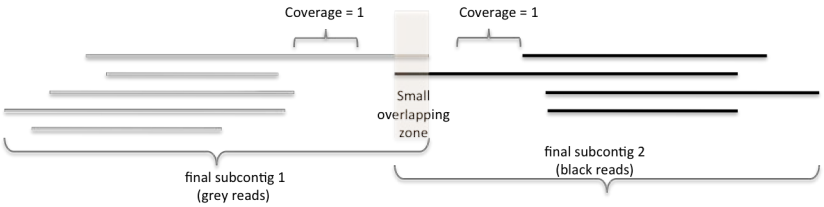


Fig. 3. Example of contig after read edition, where only two reads slightly connect two putative subcontigs. CoMiner will divide it in two new subcontigs.

where K = 1.4826 to consider outliers only those values beyond the third quartile. Therefore, reads with $F_{read}(r) > F_{cutoff}$ are considered conflictive and candidate for edition.

(iii) *Edition of candidate-read(s)*: The distribution of mismatches in candidate reads is analyzed as detailed in Fig. 2, driving to the trimming or removal of conflictive read(s) depending on mismatch distribution.

(iv) *Contig verification and saving*: Read edition may modify the contig coverage and some region(s) may be now devoid of any read. The algorithm looks for this type of situations and splits the contig in two new, independent subcontigs, each one with a new, independent consensus sequence. There is a special case where a contig can contain an overlapping region of two reads flanked by a coverage of only one read (Fig. 3). This contig will be split into two independent subcontigs when the overlapping fragment is below 40 nt or the identity is below 90%. Unedited contigs, edited contigs and new subcontigs are then written into a new ACE file.

Table 1: Results of two assemblies before (–) and after (+) CoMiner treatment

	Transcriptome		**Genome**	
CoMiner	–	+	–	+
Contig #	76 824	76 894	35 777	35 823
Mean contig size (nt)	429	428	471	470
N50 (nt)	484	482	506	505
N90 (nt)	251	252	307	307
Edited contigs		21 002		1465
Split contigs		146		74
Mapped contigs			35 412	35 458
Mapped nt			3 362 691	3 362 923

3 Results and Discussion

CoMiner performance was tested for transcriptome and genome assemblies (Table 1). A total of 1 110 923 454/FLX reads from *Solea senegalensis* transcriptome were trimmed using SeqTrimNext (http://www.scbi.uma.es/seqtrimnext) [7] and then

assembled using MIRA3 (http://www.scbi.uma.es/mira) with the standard parameters for 454/FLX data. In the resulting assembly (Table 1, Transcriptome columns), CoMiner detected HERs in 33 912 contigs (44.1%), but only edited 21 002 (27.3%), corresponding to contigs where at least one HER was caused by mismatches concentrated in at least one read. An example of this is shown in Fig. 4A, in which the algorithm identified a read containing several mismatches that were responsible for the wide, central HER. The right (3') end of this read contained all mismatches and was consequently trimmed. A new entropy analysis after CoMiner automatic edition showed that the HER had disappeared (Fig. 4B), suggesting that the edited contig was more reliable than the initial one.

Integrity of most contigs was unaffected by CoMiner edition, but 146 (0.7%) were split into two subcontigs and other 2 contigs were split into 3 different subcontigs each. It should be noted that when a subcontig consisted of only one read, it is not considered a contig and the read is removed from the final contig count. An example of contig splitting is shown in Fig. 5, where a 1570 bp contig was divided into two smaller subcontigs. Another example of this situation can be the chimeric contig group1_solea_c8241 of 1500 nt, since it was divided into a 5' subcontig of 887 nt

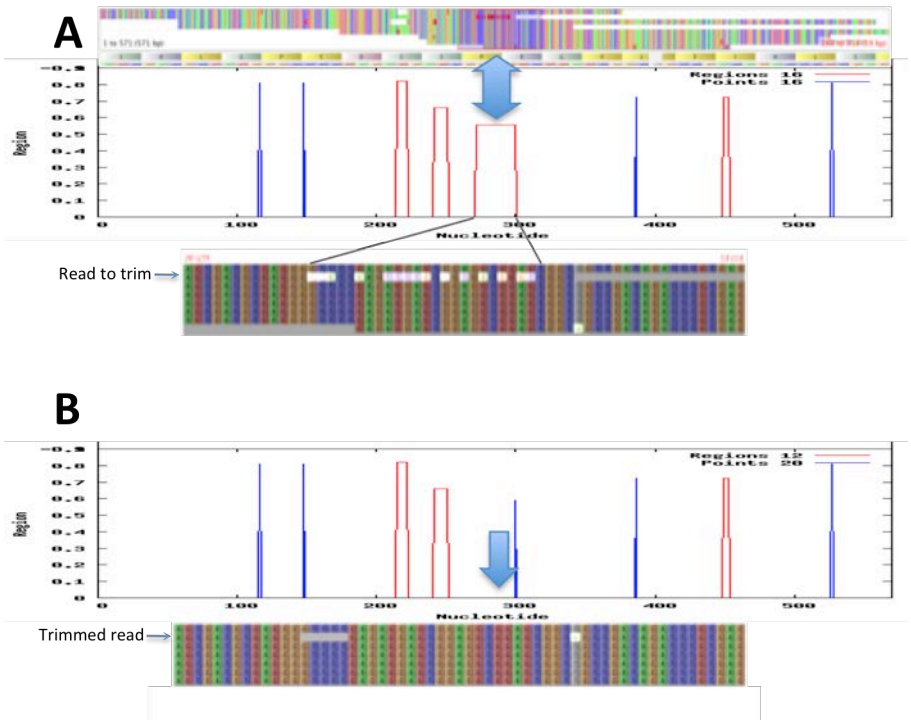


Fig. 4. Example of a 571 bp contig with several HERs before (A) and after (B) self-edition using CoMiner. After sieved entropy analysis, HERs spanning only one nucleotide are considered point errors or SNPs (in blue), while true HERs (in red) span two or more nucleotides. The wider HER (nt 273-296) is marked by a double arrow and magnified below, showing that all mismatches come from one single read. After edition (B), this wide HER disappeared. The other three smaller HERs were still present, suggesting that their mismatches were not concentrated in a single read and, therefore, will not be edited.

with similarity to an ORM1 like protein (B0V340; $E = 10^{-107}$) of 153 amino acids, and a 3' subcontig of 601 nt without similarity in databases. As a result, automatic edition of 27.3% of contigs did not significantly increase the number of contigs and did not significantly change the general parameters of the assembly (Table 1, Transcriptome columns), while contig reliability was presumably improved.

Genome DNA assembly was tested using 317 692 genomic reads of *Arabitopsis thaliana* (SRX105465). They were pre-processed with SeqTrimNext and assembled with CAP3 (http://www.scbi.uma.es/cap3) to obtain 35 777 contigs (Table 1, Genome columns). CoMiner detected 3259 contigs (9.1%) with one or more HERs, but only edited 1465 (4,1%), 74 of them being split into two or more contigs. Contigs before and after CoMiner treatment were mapped to *A. thaliana* genome using an in-house algorithm (H. Benzekri, unpublished results) to test the putative increase of contig reliability. A total of 35 412 (98.97%) and 35 458 (98.98%) contigs were mapped, respectively, providing a total of 3 362 691 and 3 362 923 mapped nucleotides, respectively (Table 1, Genome columns). When mapping was performed with a more restrictive mapper, such as Bowtie2 [8], 8453 original contigs (23.62%) and 8494 CoMiner-edited contigs (23.71%) were mapped. Edition slightly increased (1.001-1.004 fold) the amount of mapped contigs and nucleotides in any case. Unfortunately, this increase is in the same range as the total contig number, indicating that more analyses are required to provide statistical significance for this weak increase.

Even though CoMiner is currently only able to manage mismatches in overlay-layout-consensus assemblies, it seems to be a promising tool for automatic editing of mis-assembled reads. CoMiner performance was tested with transcriptome and genome data, and quality of edited contigs presumably seems improved. However, more real-world assemblies should be performed to give statistical significance to the qualitative results presented here. Finally, CoMiner edited contigs can always be

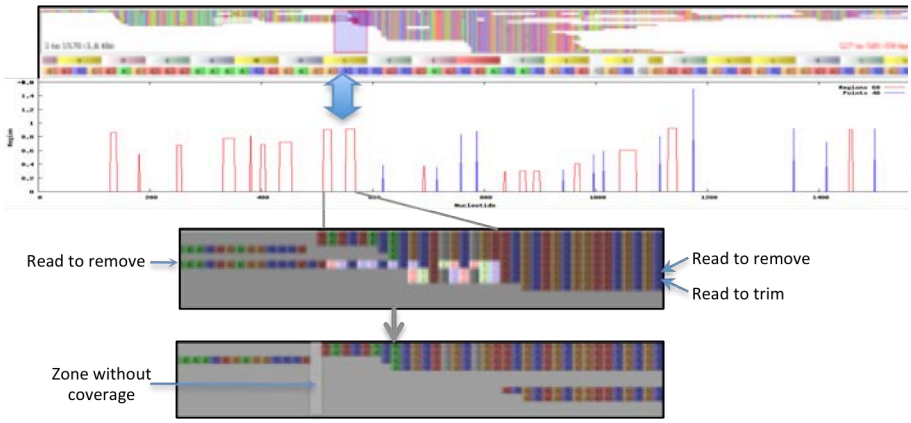


Fig. 5. Example of a 1570 bp contig with several real HERs in red. The arrow is signaling a couple of HERs that was resolved by left-trimming one read and removing two other reads. Edition caused a gap within the contig that CoMiner resolved splitting it into two subcontigs, the left subcontig of 551 nt, and the right subcontig of 1018 nt.

analyzed and visualized to search for more, different tentative errors by means of *amosvalidate* and *Hawkeye*, with the aim of spending less manual edition efforts.

Acknowledgement

The authors gratefully acknowledge Josep Planas (University of Barcelona, Spain) for kindly providing 454/FLX data from the AQUAGENET project (funded by Interreg-Sudoe). We would also like to acknowledge the computer resources and technical support provided by the Plataforma Andaluza de Bioinformática of the University of Málaga, Spain. This study was supported by grants from the Spanish MICINN (BIO2009-07490) and Junta de Andalucía (P10-CVI-6075), as well as institutional funding to the research group BIO-114 and an agreement with AQUAGENET project.

References

1. Istrail, S.; Sutton, G.G.; Florea, L.; Halpern, A.L.; Mobarry, C.M.; Lippert, R.; Walenz, B.; Shatkay, H.; Dew, I.; Miller, J.R.; Flanigan, M.J.; Edwards, N.J.; Bolanos, R.; Fasulo, D.; Halldorsson, B.V.; Hannenhalli, S.; Turner, R.; Yooseph, S.; Lu, F.; Nusskern, D.R.; Shue, B.C.; Zheng, X.H.; Zhong, F.; Delcher, A.L.; Huson, D.H.; Kravitz, S.A.; Mouchard, L.; Reinert, K.; Remington, K.A.; Clark, A.G.; Waterman, M.S.; Eichler, E.E.; Adams, M.D.; Hunkapiller, M.W.; Myers, E.W.; Venter, J.C. Whole-genome shotgun assembly and comparison of human genome assemblies. *Proc Natl Acad Sci U S A* **2004**, *101*, 1916-1921.
2. Phillippy, A.M.; Schatz, M.C.; Pop, M. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biol* **2008**, *9*, R55.
3. Schatz, M.C.; Phillippy, A.M.; Shneiderman, B.; Salzberg, S.L. Hawkeye: an interactive visual analytics tool for genome assemblies. *Genome Biol* **2007**, *8*, R34.
4. Bonfield, J.K.; Whitwham, A. Gap5--editing the billion fragment sequence assembly. *Bioinformatics* **2010**, *26*, 1699-1703.
5. Gordon, D.; Abajian, C.; Green, P. Consed: a graphical tool for sequence finishing. *Genome Res* **1998**, *8*, 195-202.
6. Guerrero, D.; Bautista, R.; Villalobos, D.P.; Canton, F.R.; Claros, M.G. AlignMiner: a Web-based tool for detection of divergent regions in multiple sequence alignments of conserved sequences. *Algorithms Mol Biol* **2010**, *5*, 24.
7. Falgueras, J.; Lara, A.J.; Fernandez-Pozo, N.; Canton, F.R.; Perez-Trabado, G.; Claros, M.G. SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read. *BMC Bioinformatics* **2010**, *11*, 38.
8. Langmead, B.; Salzberg, S.L. Fast gapped-read alignment with Bowtie 2. *Nat Methods* **2012**, *9*, 357-359.