Week 12, Lecture 23

István Albert
Bioinformatics Consulting Center
Penn State
Genome Assembly
From sequencing reads to whole genomes

Multistep process

1. **Assemble** short reads into longer sequences → contigs

2. **Scaffolding** – arrange contigs relative to one another or as seen on a similar known genome

3. Genome **finishing** (gap closure) → fill in gaps with directed sequencing procedures and manual curation
AMOS – A Modular Open-Source whole genome assembler

The AMOS consortium is committed to the development of open-source whole genome assembly software. The project acronym (AMOS) represents our primary goal -- to produce a Modular, Open-Source whole genome assembler. Open-source so that everyone is welcome to contribute and help build outstanding assembly tools, and modular in nature so that new contributions can be easily inserted into an existing assembly pipeline. This modular design will foster the development of new assembly algorithms and allow the AMOS project to continually grow and improve in hopes of eventually becoming a widely accepted and deployed assembly infrastructure. In this sense, AMOS is both a design philosophy and a software system.

Announcements

- August 5, 2010 - Version 3.1.0 of AMOS released!
- August 2, 2011 - AMOS Sample Data posted
- December 7, 2010 - Version 2.0 of AMOS released

It is not a single software rather than a collection of interoperable tools, standards and techniques.
Read assembly challenges

repeated elements: RPT A1 and RPT A2

A valid assembly of two contigs instead of one

Other challenges:

• genomic variation, heterozygosity, copy number variation
• misassembly due to sequencing errors
• chimeric

Images and content from
http://www.cbcb.umd.edu/research/assembly_primer.shtml
Scaffolding

- Orienting contigs via paired end (or mate-pair) information

Additional information to assist the process:

- use alignment positions in related genomes
- use gene synteny (co-localization of genetic loci)

There are fewer automated pipelines:

**BAMBUS** – Hierarchical Scaffolding With Bambus by M. Pop, D. Kosack and S. Salzberg
## Finishing genomes with limited resources: lessons from an ensemble of microbial genomes

Niranjan Nagarajan, Christopher Cook, MariaPia Di Bonaventura, Hong Ge, Allen Richards, Kimberly A Bishop-Lilly, Robert DeSalle, Timothy D Raed, and Mihai Pop

### Abstract

The process of finishing a genome and moving it from a draft stage (the result of sequencing and initial assembly) to a complete genome is typically a time and resource intensive task. The advent of new sequencing technologies has come with its own set of opportunities and pitfalls in the finishing process. While genomes can now be sequenced to high redundancy in a cost-effective manner, the process of assembling the genomes is more challenging and often draft genomes are fragmented into hundreds of contigs. Correspondingly, the task of producing the complete genome can involve months of lab work and thousands of finishing experiments and is usually done in large genome centers.

The work in our lab has focused on computational approaches to speed up the finishing process. Specifically, we have explored the use of optical mapping and mate-pair data to augment assemblies and direct finishing experiments. The tools developed in our lab have been used in several finishing projects, producing complete genomes (and near-complete ones) with surprisingly little computational and experimental effort (Nagarajan et al., in submission). The executables (as well as source code) for these tools are freely available here:

- **Scaffolding using Optical Restriction Mapping**
  Optical Maps are global, ordered maps of restriction site locations in a genome. This information can be quite useful in scaffolding contigs from a shotgun assembly to guide the finishing process. A set of programs to exploit optical maps for assembly can be found here: [SOMA v2.0](http://soma.mpimp-golm.mpg.de). This version of SOMA contains several improvements to programs in v1.0 as well as new scripts for working with multiple maps, contig graphs and scaffolds.
Genome assembly is an art

Many different approaches – substantial supervision/evaluation required at each step of the process.

Genomes can vary greatly in complexity – genome size/repetitiveness is usually the limiting factor

Constant tuning and evaluation is needed.
The N50 statistic

• N50 length is defined as the contig length L for which 50% of all bases in the sequences are in contigs of length less than L.

1. Sort all contigs by size from highest to lowest
2. Compute cumulative sum of lengths
3. Smallest number of contigs that add up to the half of the assembled length
Using the Velvet Assembler

Velvet
Sequence assembler for very short reads

- Current version: 1.1.06
- Manual and extension for Columbus in pdf format
- Public Git URL: git clone git://github.com/dzerbino/velvet.git
- For up-to-date info, you can consult and/or subscribe to the mailing list.
- For transcriptomic assembly Velvet is extended by Oases.

Download, unpack, and make Velvet

Download the 23.tar.gz dataset from the webpage

Velvet Assembly is a two step process:
- velveth → build a hashtable
- velvetg → run the assembly
Generate input files

```
$ wgsim -N 100000 -d 1000 23.fasta 1.fq 2.fq > mutation.txt

# we'll test out both single and paired end data
$ cat 1.fq 2.fq > all.fq

# run with haslenght 21 and put the results in the directory called h21
$ velveth h21 21 -fastq all.fq
```
The assembly is located in the file `h21/contigs.fasta`.
Unlike mapping where defaults usually work fine during assembly we need to tune parameters. Change both the hash size and graph assembly parameters.

```
$ # run velvetg on the directory created by velveth
$ velvetg h21

Final graph has 8617 nodes and n50 of 250, max 1656, total 581421, using 0 reads
```

```
$ velveth h31 31 -fastq all.fq

Final graph has 123 nodes and n50 of 14737, max 51168, total 489332, using 18026/200000 reads
```

The assembly is located in the file `h21/contigs.fasta`.
Unlike mapping where defaults usually work fine during assembly we need to tune parameters. Change both the hash size and graph assembly parameters.
Assembly evaluation

• Often feels surprisingly ad-hoc (people write home grown scripts to fetch statistics/subselect contigs etc)

• AMOS – contains visualizers hawkeye
Aligning contigs

**Overview**

MUMmer is a system for rapidly aligning entire genomes, whether in complete or draft form. For example, MUMmer 3.0 can find all 20-basepair or longer exact matches between a pair of 5-megabase genomes in 13.7 seconds, using 78 MB of memory, on a 2.4 GHz Linux desktop computer. MUMmer can also align incomplete genomes; it can easily handle the 100s or 1000s of contigs from a shotgun sequencing project, and will align them to another set of contigs or a genome using the NUCmer program included with the system. If the species are too divergent for a DNA sequence alignment to detect similarity, then the PROmer program can generate alignments based upon the six-frame translations of both input sequences. The original MUMmer system, version 1.0, is described in our 1999 *Nucleic Acids Research* paper. Version 2.1 appeared a few years later and is described in our 2002 *Nucleic Acids Research* paper, while MUMmer 3.0 was recently described in our 2004 *Genome Biology* paper. We have also developed a GPU accelerated version of MUMmer called MUMmerGPU.

For more information regarding the MUMmer package, please refer to the:

- **Distribution**
- **Online manual**
- **Online tutorials**

To receive software update notices, please join the users’ mailing list. This list will only be used to announce major version releases: mummer-users@lists.sourceforge.net
MUMmer tools

- nucmer $\rightarrow$ (NUCleotide MUMmer) DNA sequence alignment

- promer $\rightarrow$ PROmer (PROtein MUMmer) - all matching and alignment routines performed on the six frame amino acid translation of the DNA input sequence
$ nucmer --prefix=out contigs.fa 23.fasta

$ mummerplot -postscript out.delta
Homework 23

• Generate reads from the input dataset and assemble them with velvet

• (you may recompile velvet to allow larger kmer sizes by using

  make 'MAXKMERLENGTH=61’

Does the assembly improve with larger hash sizes?