

# Assembly Tutorial

Michael Schatz

Nov 15, 2011

CSHL Sequencing Course

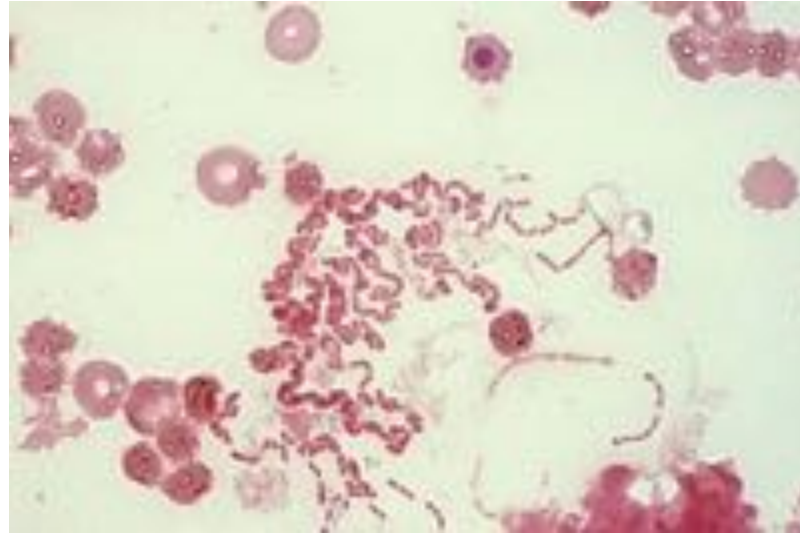




# Outline

1. Sample Data
2. ALLPATHS-LG
3. SOAPdenovo + Quake
4. MUMmer

# Rhodobacter sphaeroides



## Genome architecture:

- Chr1: 3Mbp, Chr2: 900kbp, 5 plasmid
- High GC: 69%

### Library 1: Fragment

Avg Read length: 101bp

Insert length: 180bp

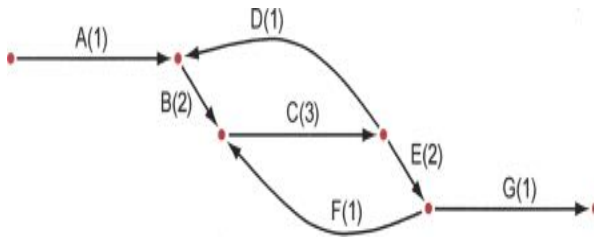
# of reads: 2,050,868

### Library 2: Short jump

Avg Read length: 101bp

Insert length: 3500bp

# of reads: 2,050,868



## Running ALLPATHS-LG Iain MacCallum

## ALLPATHS-LG mission

---

### Goal

- high quality 'automated' genome assembly from low-cost data

### How

- define sequencing model to optimize results
- develop algorithm that works well on these data
- change as sequencing technology changes

## DNA source

---

ALLPATHS-LG assumes that the DNA comes from a single individual or culture (one genome)

Mixed populations = mixed genomes

viral patient sample

many individuals from a species all squished together

many species (metagenomic)

These types of data would require a different algorithm....

## How to use ALLPATHS-LG

---

1. **What is an ALLPATHS-LG assembly?**
2. Data requirements
3. Computational requirements
4. Installation
5. Preparing your data
6. Assembling

## Three models for assembly

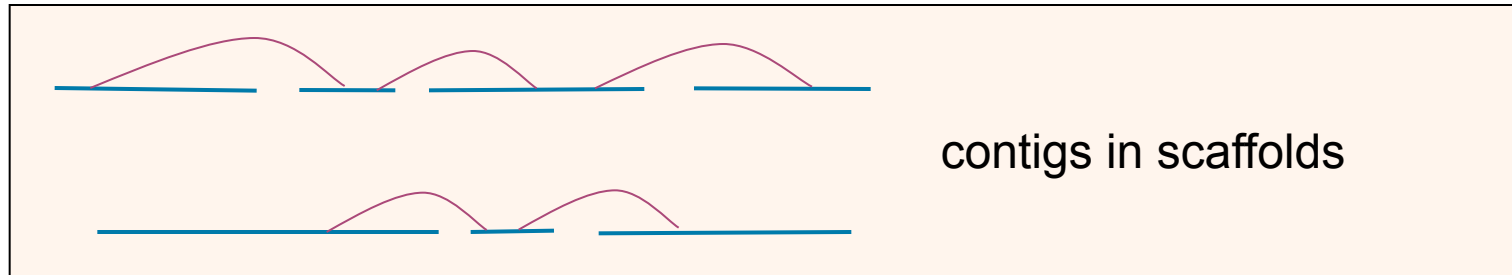
---

1. Linear assemblies
2. Graph assemblies
3. Linearized graph assemblies (ALLPATHS-LG)



# 1. Linear assemblies

---



**contig**: a contiguous sequence of bases....

```
CTGCCCCCTGTGCCAATGGGTTTGAGGCTCTTCCCACTTCCTTTTCTATTAGATTCAATGTATCTGGTTTTATGTTGAGG
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
```

**scaffold**: a sequence of contigs, separated by gaps....

```
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTGAGAATTGCTCTTCCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAACTTGAAGTTATTGTCATACA
```

Number of Ns = predicted gap size,  
with error bars (can't be displayed in fasta format)

# 1. Linear assemblies

---

## Example of an assembly in fasta format

>scaffold\_1

TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA  
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTTACTTCCTTGTCAAAAATCAAGTGA  
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC  
NNNNNNNN

AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA  
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCATGTCTTTGAAGAATTGTGTT  
NN

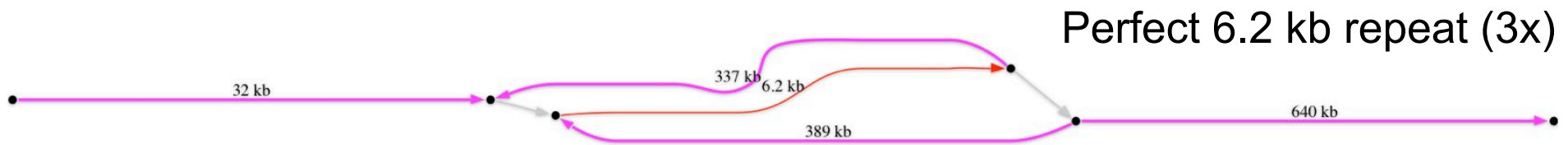
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT  
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAACTTGAAGTTATTGTCATACA

>scaffold\_2

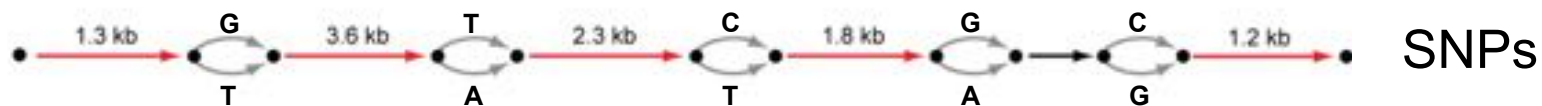
CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATTGCTTATGTATGCTATCATATCACTTGCAAA  
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCCATTGATCTCTTTCTGTTGTCTTATTGTTCTAGCTAACACTT  
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTCAAGTGGGATTGCTTCAAGTATG

## 2. Graph assemblies

Graph assemblies are different from linear assemblies: they can have branches



$T^n$ , not sure what n is



Graphs retain intrinsic ambiguity, allow alternatives  
Downside: complicated!

### 3. Linearized graph assemblies

---

#### Efasta


...ACTGTTT{A,C}GAAAT...

A or C at site

...CGCGTTTTTTTTTTT{T,TT}CAT...

0 or 1 or 2 Ts at site

Can represent 'linear' graphs like this:

Linearized graph assembly: an assembly consisting of contigs in scaffolds, with  
e  e these

ALLPATHS-LG produces these.

Still figuring out the ideal solution....

Note challenge for community to figure out how to use!

## Example of an assembly in efasta format

```
>scaffold_1
TCCTAGATCCACTTGGACTTGAGCTTTGTATATATATATATATATATA{,TA}CAAGATGACATATATAGGAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNNN
AGTTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTTCCCTAGTCTGGGTTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGTATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTCTTGAGAACTTGAAGTTATTGTCATACA
>scaffold_2
CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATT{A,C,G}GCTTATGTATGCTATCTTGCAAA
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCCATTGATCTCTTCTGTGTCTTATTGTTCTAGCTAACACTT
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTCAAGTGGGATTGCTTCAAGTATG
```

## How to use ALLPATHS-LG

---

1. What is an ALLPATHS-LG assembly?
2. **Data requirements (\*\*\*) most critical thing \*\*\*)**
3. Computational requirements
4. Installation
5. Preparing your data
6. Assembling

## ALLPATHS-LG sequencing model

---

Libraries (insert types)	Fragment size (bp)	Read length (bases)	Sequence coverage (x)	Required
Fragment	180*	$\geq 100$	45	yes
Short jump	3,000	$\geq 100$ preferable	45	yes
Long jump	6,000	$\geq 100$ preferable	5	no**
Fosmid jump	40,000	$\geq 26$	1	no**

\*See next slide.

\*\*For best results. Normally not used for small genomes.  
However essential to assemble long repeats or duplications.

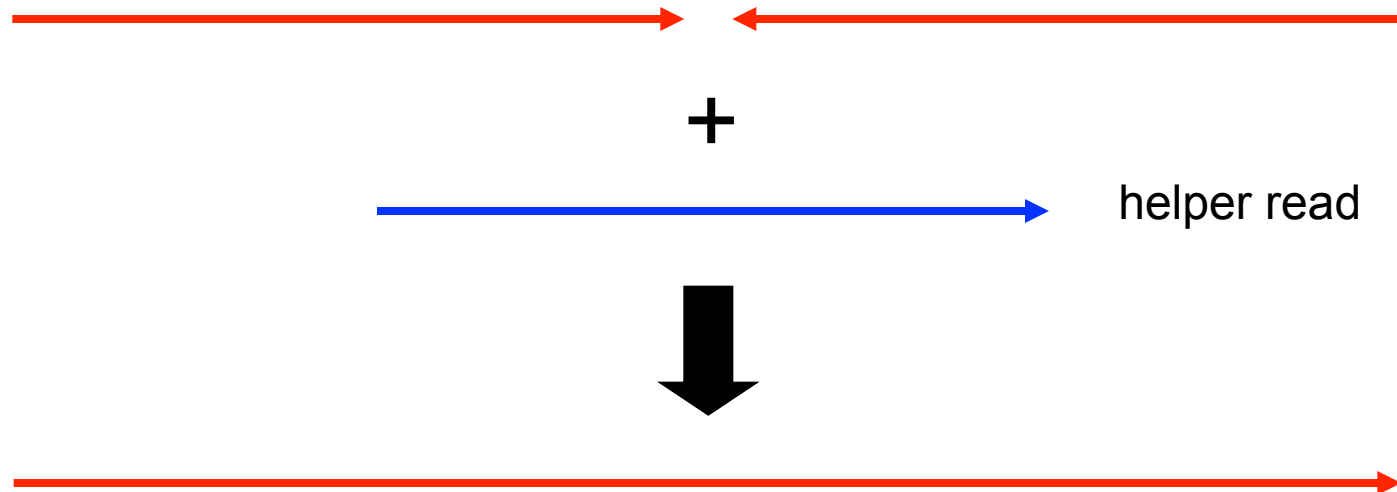
Cutting coverage in half still works, with some reduction in quality of results.

All: protocols are either available, or in progress.

## Libraries from 180 bp fragments

---

Pairs of 100 base reads from these libraries are merged to create 'reads' that are twice as long:



For longer reads, fragment size would be increased proportionally.

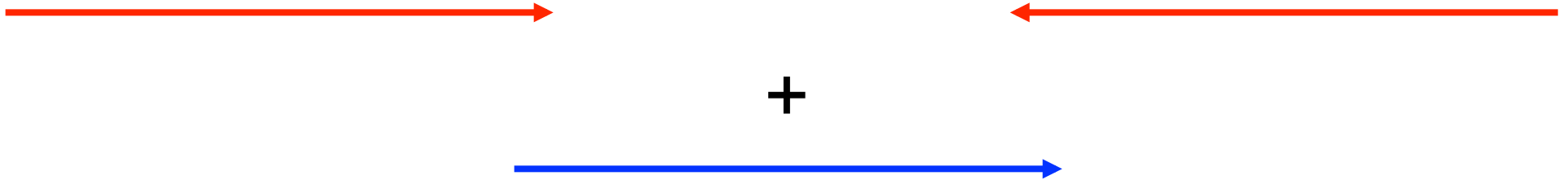


## Fragment libraries

---

Potential problems:

(a) too many pairs too far apart



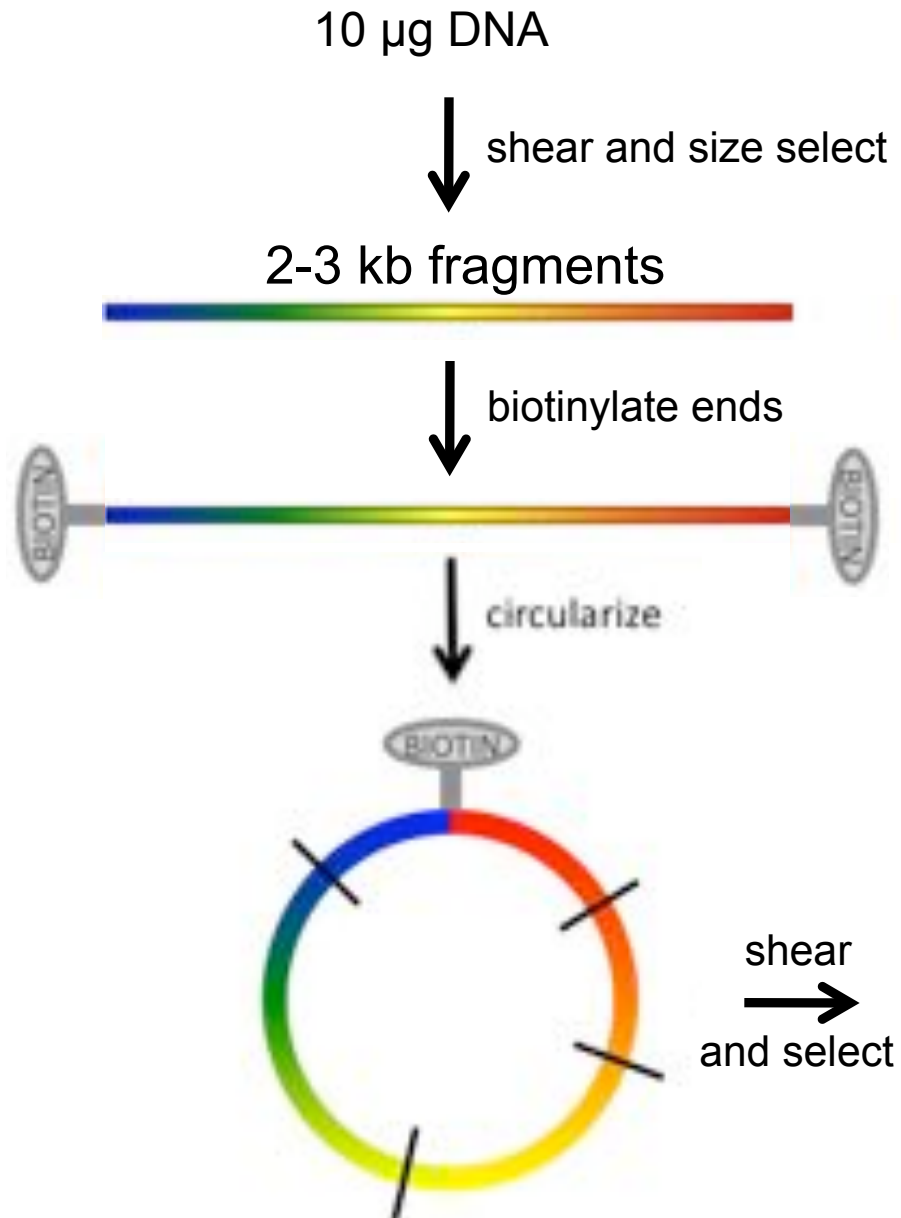
Consequence: these pairs lost by algorithm

(b) too many pairs too close together

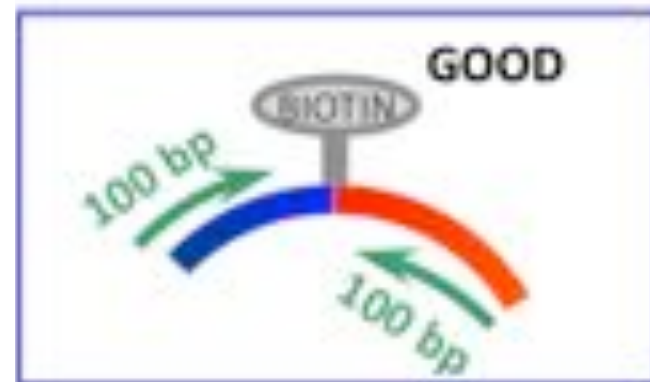


Consequence: merged 'reads' too short, power lost

## Short jumping libraries (2-3 kb)



Illumina protocol, blunt-end ligation



## Short jumping libraries (2-3 kb)

---

**Problem 1.** Many steps → many opportunities for failure.

Example: a reagent might degrade. (This has happened.)

## Short jumping libraries (2-3 kb)

---

**Problem 2.** Many steps → many DNA losses.

Here are *good* results for a mammalian genome:

Input: 10 µg DNA ↔ ~3,000,000x physical coverage

Output: (if fully sequenced) ~3,000x physical coverage

Loss: 99.9% (not including DNA between reads)

**Small genomes are much easier!**



## Short jumping libraries (2-3 kb)

---

**Problem 3.** Read passes through circularization junction. This reduces the effective read length (and complicates algorithm).



What might be done to reduce incidence of this:  
shear circles to larger size and select larger fragments

## Short jumping libraries (2-3 kb)

---

**Problem 4.** Reads come from nonjumped fragments and are thus in reverse orientation and close together on the genome. This reduces yield (and complicates algorithm).



Putative cause: original DNA is nicked or becomes nicked during process – biotins become ‘ectopically’ attached at these nicks



## Long jumping libraries (~6 kb)

---

**Method 1.** Instead of shearing circles, using EcoP15I restriction enzyme.

### Pros

- demonstrated to work
- no artifacts

### Cons

- read length = 26 bases

**Method 2.** Use Illumina blunt-end ligation protocol, but shear and size select larger fragments.

### Pros

- long reads

### Cons

- yield may be very low (probably not problem for small genomes)

## How to use ALLPATHS-LG

---

1. What is an ALLPATHS-LG assembly?
2. Data requirements
- 3. Computational requirements**
4. Installation
5. Preparing your data
6. Assembling



## Computational requirements

---

- 64-bit Linux
- runs multi-threaded on a single machine
- memory requirements
  - about 160 bytes per genome base, implying
    - need 512 GB for mammal (Dell R315, 48 processors, \$39,000)
    - need 1 GB for bacterium (theoretically)
  - if coverage different than recommended, adjust...
  - potential for reducing usage
- wall clock time to complete run
  - 5 Mb genome → 1 hour (8 processors)
  - 2500 Mb genome → 500 hours (48 processors)

## Prerequisite software

---

- g++ compiler. We use version 4.4.3. Older versions may not work.  
<http://gcc.gnu.org>
- C++ Boost library. We use version 1.39.  
<http://www.boost.org>
- Picard command-line utilities for SAM file manipulation.  
<http://picard.sourceforge.net>

## How to use ALLPATHS-LG

---

1. What is an ALLPATHS-LG assembly?
2. Data requirements
3. Computational requirements
- 4. Installation**
5. Preparing your data
6. Assembling

## Installing ALLPATHS-LG

---

Web page:

<http://www.broadinstitute.org/software/allpaths-lg/blog/>

General instructions:

<http://www.broadinstitute.org/science/programs/genome-biology/computational-rd/general-instructions-building-our-software>

## Getting the ALLPATHS-LG source

---

Our current system is to release code daily if it passes a test consisting of several small assemblies:

Download the latest build from:

<ftp://ftp.broadinstitute.org/pub/crd/ALLPATHS/Release-LG/>

Unpack it:

```
% tar xzf allpathslg-39099.tar.gz
```

(substitute the latest revision id for 39099)

This creates a source code directory `allpathslg-39099`:

```
% cd allpathslg-39099
```

## Building ALLPATHS-LG

---

**Step one:** `./configure`

Options:

```
-prefix=<prefix path>  
  put binaries in <prefix path>/bin, else ./bin  
--with-boost=<boost dir>  
  you only need this if configure cannot find boost
```

**Step two:** `make` and `make install`

Options:

```
-j<n>  
  compile with n parallel threads
```

**Step three:** add bin directory to your path

## How to use ALLPATHS-LG

---

1. What is an ALLPATHS-LG assembly?
2. Data requirements
3. Computational requirements
4. Installation
- 5. Preparing your data**
6. Assembling

## Preparing data for ALLPATHS-LG

---

Before assembling, prepare and import your read data.

ALLPATHS-LG expects reads from:

- At least one fragment library. One should come from fragments of size ~180 bp. This isn't checked but otherwise results will be bad.
- At least one jumping library.

IMPORTANT: use all the reads, including those that fail the Illumina purity filter (PF). These low quality reads may cover 'difficult' parts of the genome.



## ALLPATHS-LG input format

---

ALLPATHS-LG can import data from:

BAM, FASTQ, FASTA/QUALA or FASTB/QUALB files.

You must also provide two metadata files to describe them:

`in_libs.csv` - describes the libraries

`in_groups.csv` - ties files to libraries

FASTQ format: consists of records of the form

@<read name>

<sequence of bases, multiple lines allowed>

+

<sequence of quality scores, with Qn represented by ASCII code n+33, multiple lines allowed>

## Libraries – in\_libs.csv (1 of 3)

---

`in_libs.csv` is a comma separated value (CSV) file.  
For clarity, blanks and tabs are allowed and ignored.

The first line describes the field names, listed below.  
Each subsequent line describes a library.

`library_name` - a unique name for the library.

Each physically different library should have a different name!

## Libraries – in\_libs.csv (2 of 3)

---

### For fragment libraries only

frag\_size            - estimated mean fragment size  
frag\_stddev        - estimated fragment size std dev

### For jumping libraries only

insert\_size        - estimated jumping mean insert size  
insert\_stddev     - estimated jumping insert size std dev

These values determine how a library is used. If `insert_size` is  $\geq 20000$ , the library is assumed to be a Fosmid jumping library.

paired             - always 1 (only supports paired reads)  
read\_orientation   - inward or outward.

Paired reads can either point towards each other, or away from each other. Currently fragment reads must be `inward`, jumping reads `outward`, and Fosmid jumping reads `inward`.

## Libraries – in\_libs.csv (3 of 3)

---

Reads can be trimmed to remove non-genomic bases produced by the library construction method:

genomic\_start  
genomic\_end                      - inclusive zero-based range of read bases  
   to be kept; if blank or 0 keep all bases

Reads are trimmed in their original orientation.

Extra optional fields (descriptive only – ignored by ALLPATHS)

project\_name                      - a string naming the project.  
organism\_name                    - the organism name.  
type                                - fragment, jumping, EcoP15I, *etc.*

### EXAMPLE

library_name,	type,	paired,	frag_size,	frag_stddev,	insert_size,	insert_stddev,	read_orientation,	genomic_start,	genomic_end
Solexa-11541,	fragment,	1,	180,	10,	,	,	inward	,	
Solexa-11623,	jumping,	1,	,	,	3000,	500,	outward	0,	25

## Input files – required format

---

Each BAM or FASTQ file contains paired reads from *one* library.

Data from a single library can be split between files.  
Example, one file for each Illumina lane sequenced.

**For FASTQ format**, the paired reads can be divided in two files (`readsA.fastq`, `readsB.fastq`), or, if in a single file (`reads.fastq`), must be **interleaved**:

```
pair1_readA
pair1_readB
pair2_readA
pair2_readB
...
```

## Input files – in\_groups.csv

---

Each line in `in_groups.csv` comma separated value file, corresponds to a BAM or FASTQ file you wish to import for assembly.

The library name must match the names in `in_libs.csv`.

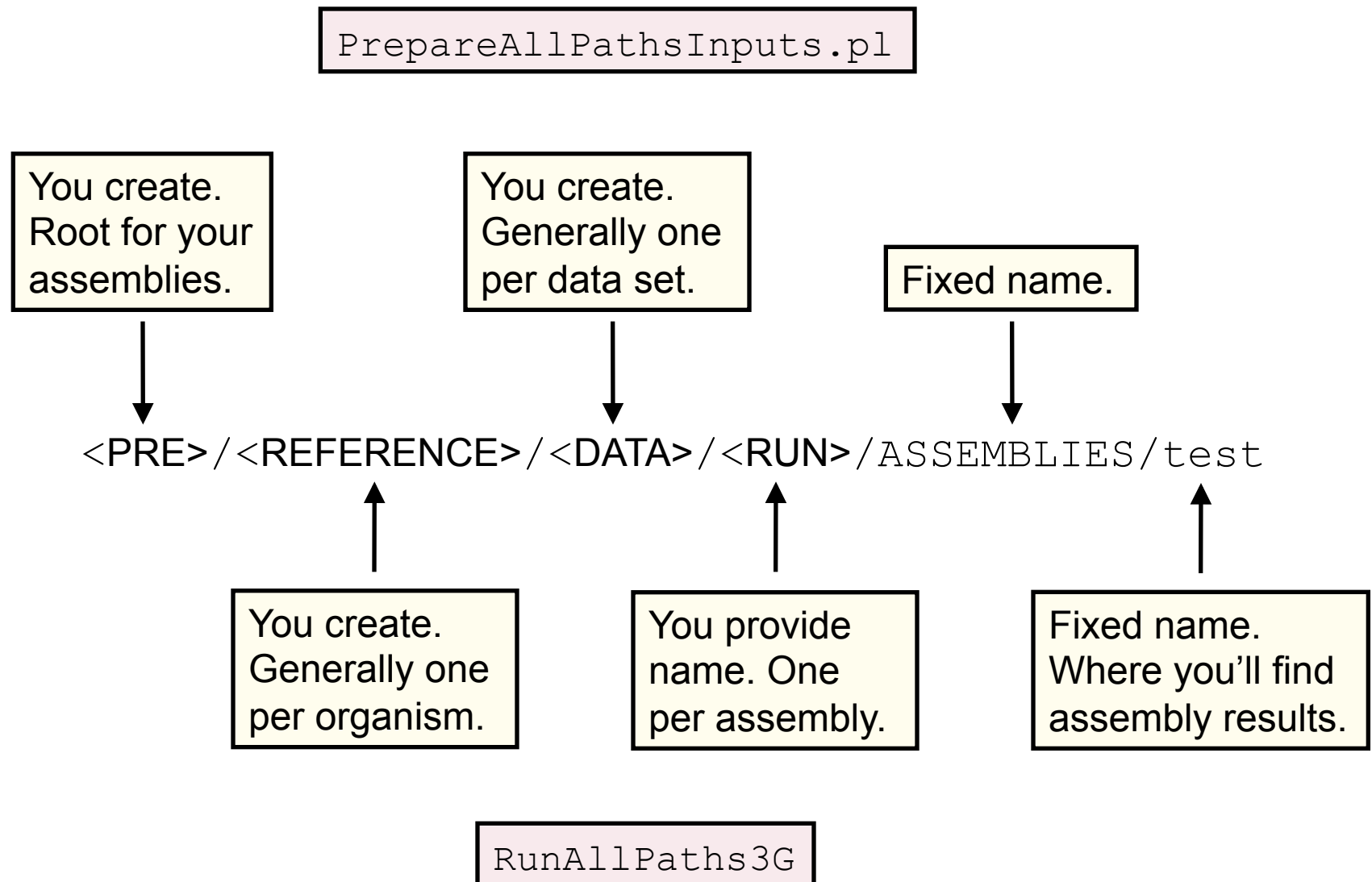
<code>group_name</code>	- a unique nickname for this file
<code>library_name</code>	- library to which the file belongs
<code>file_name</code>	- the absolute path to the file (should end in .bam or .fastq) (use wildcards '?', '*' for paired fastqs)

### Example:

<code>group_name,</code>	<code>library_name,</code>	<code>file_name</code>
302GJ,	Solexa-11541,	/seq/Solexa-11541/302GJABXX.bam
303GJ,	Solexa-11623,	/seq/Solexa-11623/303GJABXX.?.fastq

## ALLPATHS-LG directory structure

---



## How to import assembly data files

---

PrepareAllPathsInputs.pl

IN\_GROUPS\_CSV=<in groups file>

IN\_LIBS\_CSV=<in libs file>

DATA\_DIR=<full path of data directory>

PLOIDY=<ploidy, either 1 or 2>

PICARD\_TOOLS\_DIR=<picard tools directory>

HOSTS=<list of hosts to be used in parallel>

- IN\_GROUPS\_CSV and IN\_LIBS\_CSV: optional arguments with default values ./in\_groups.csv and ./in\_libs.csv. These arguments determine where the data are found.
- DATA\_DIR: imported data will be placed here.

*(continued)*



## How to import assembly data files

---

- `PLOIDY`: either 1 (for a haploid or inbred organism), or 2 (for a diploid organism) – we have not tried to assemble organisms having higher ploidy!
- `PICARD_TOOLS_DIR`: path to Picard tools, for data conversion from BAM.
- `HOSTS`: optional list of hosts to use in parallel; example:  
`HOSTS="2, 4 .host3"`, translates to: 2 processes forked on this machine; 4 processes forked on host3. Forking to remote hosts requires password-less ssh access, e.g. using ssh-agent/ssh-add.

## How to use ALLPATHS-LG

---

1. What is an ALLPATHS-LG assembly?
2. Data requirements
3. Computational requirements
4. Installation
5. Preparing your data
6. **Assembling**

## How to assemble

---

### Do this:

```
RunAllPathsLG \
  PRE=<prefix path> \
  REFERENCE_NAME=<reference dir> \
  DATA_SUBDIR=<data dir> \
  RUN=<run dir>
```

Automatic resumption. If the pipeline crashes, fix the problem, then run the same `RunAllPathsLG` command again. Execution will resume where it left off.

### Results. The assembly files are:

<code>final.contigs.fasta</code>	- fasta contigs
<code>final.contigs.efasta</code>	- efasta contigs
<code>final.assembly.fasta</code>	- scaffolded fasta
<code>final.assembly.efasta</code>	- scaffolded efasta

## Putting it all together

---

1. Collect the BAM or FASTQ files that you wish to assemble. Create a `in_libs.csv` metadata file to describe your libraries and a `in_groups.csv` metadata file to describe your data files.

2. Create a data directory and import your data.

```
% mkdir -p /assemblies/E.coli/experiment1
% PrepareAllPathsInputs.pl \
  DATA_DIR=/assemblies/E.coli/experiment1 \
  PICARD_TOOLS_DIR=<picard tools directory> \
  PLOIDY=1
```

3. Assemble.

```
% RunAllPathsLG PRE=/assemblies \
  REFERENCE_NAME=E.coli \
  DATA_SUBDIR=experiment RUN=assembly1
```

4. Get the results (four files).

```
/assemblies/E.coli/experiment1/ASSEMBLIES/assembly1/
final.{assembly,contigs}.{fasta,efasta}
```

## Putting it all together

---

### 1. Prepare input files

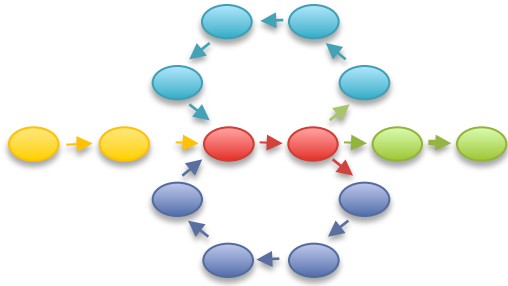
```
% cd /bluearc/data/schatz/mschatz/course/Data/original
% PrepareAllPathsInputs.pl \
  DATA_DIR=`pwd` \
  PICARD_TOOLS_DIR=~/.build/packages/picard-tools-1.52/ \
  PLOIDY=1 GENOME_SIZE=5000000
```

### 2. Assemble.

```
% RunAllPathsLG \
  PRE=/bluearc/data/schatz/mschatz/course/ \
  DATA_SUBDIR=original RUN=default \
  REFERENCE_NAME=Data K=96 THREADS=24
```

### 3. Get the results (four files).

```
/assemblies/E.coli/experiment1/ASSEMBLIES/assembly1/
final.{assembly,contigs}.{fasta,efasta}
```



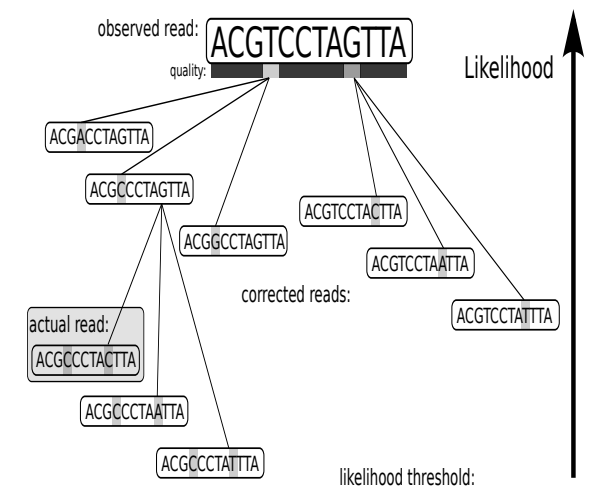
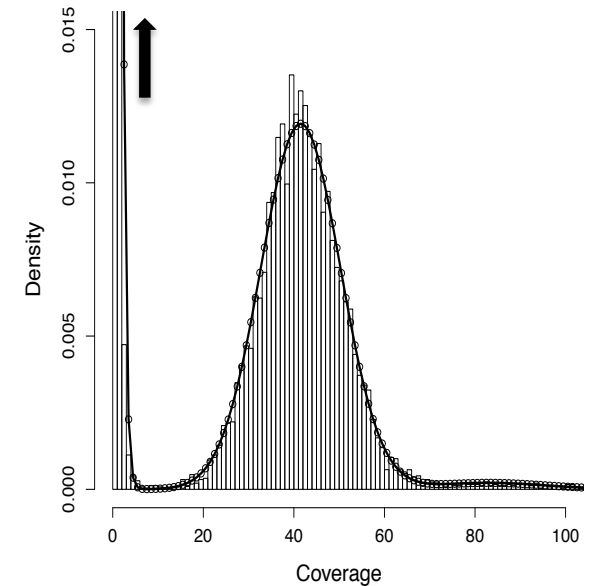
**Running Quake + SOAPdenovo**

# Quake

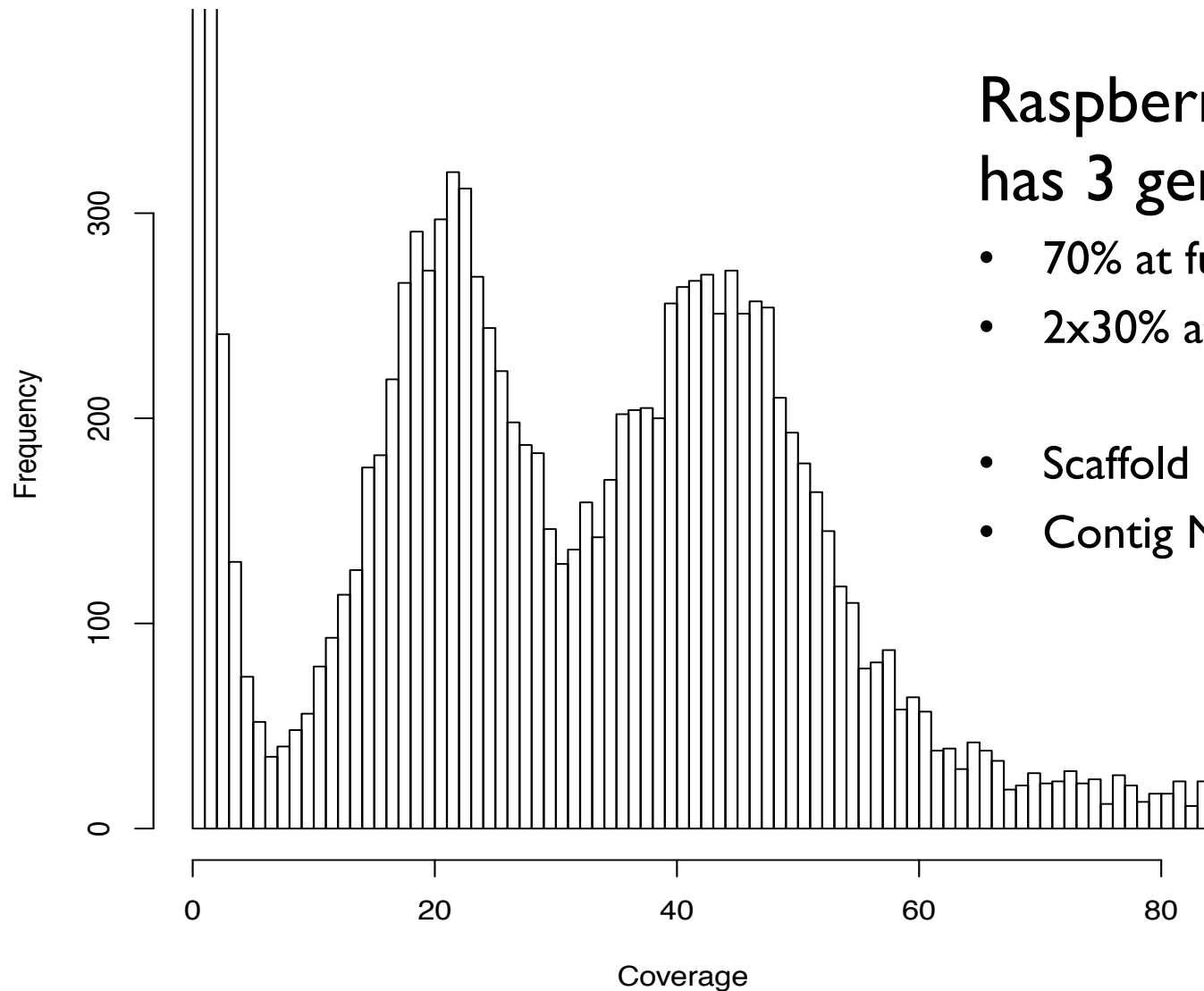
- Corrects substitution errors in Illumina reads by fitting kmers distribution to mixture model of erroneous/correct kmers
- Critical value is selecting appropriate k
  - Need to pick K so that k-mer expected to be unique in genome
    - For a 5 Mb genome set  $k \geq 15$
    - For a 3 Gb genome set  $k \geq 19$ .
  - Quake requires more RAM as K gets larger
    - Requires  $4^k$  bits of memory
    - 125Mb for  $k=15$
    - 32GB for  $k=19$

```
% cat > Data/original/quake_files.txt << EOF
frag_1.fastq frag_2.fastq
shortjump_1.fastq shortjump_2.fastq
EOF
```

```
% cd Data/original/ && ../../Quake/bin/quake.py
-f quake_files.txt -k 17 -p 20 --no_jelly
```



# Heterozygous Genomes



Raspberry effectively  
has 3 genomes

- 70% at full coverage
- 2x30% at half coverage
- Scaffold N50: 17kbp
- Contig N50: 12kbp

```
## From kmers.txt
```

```
% Rscript Quake/bin/kmer_hist.r
```

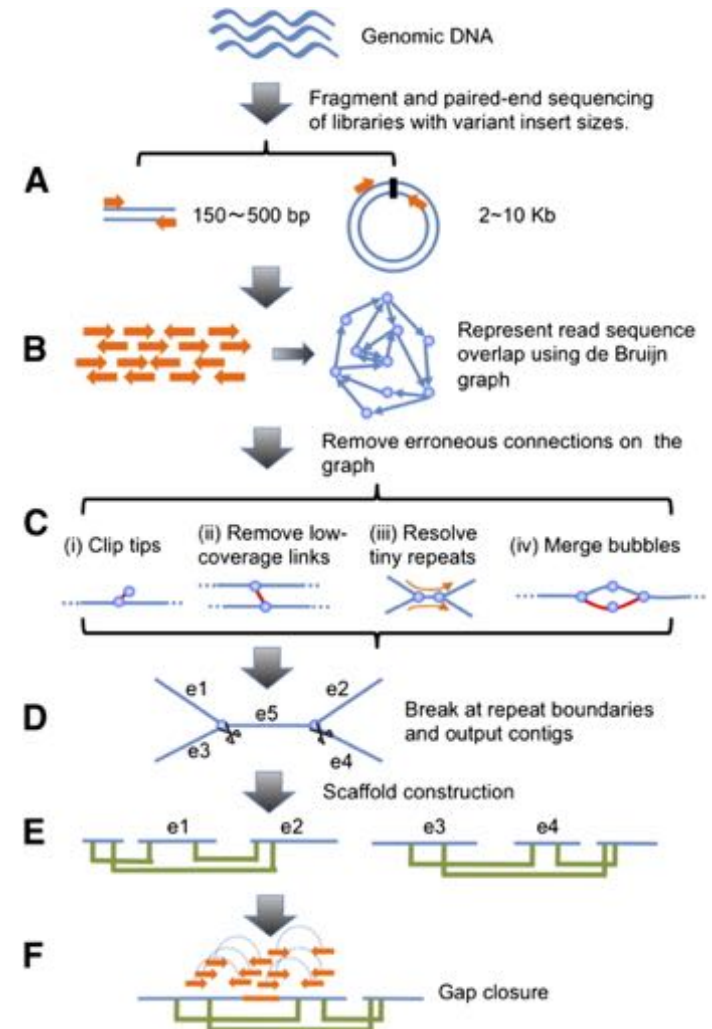


# SOAPdenovo

Li *et al.* (2010) De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*. 20: 265-272

- Designed specifically for Illumina reads
- Constructs de Bruijn graph with varying k-mer size
  - 31-mer, 63-mer, and 127-mer versions available
- 4 major phases:
  - pregraph (B): construct de Bruijn graph
  - contig (C): correct errors
  - map: (D) realign reads to initial contigs
  - scaff: (D,E) resolve repeats, and errors

```
% SOAPdenovo63mer all -K 31 \  
-p 16 -s SOAPdenovo.config \  
-o asm >> SOAPdenovo.log
```



# SOAPdenovo

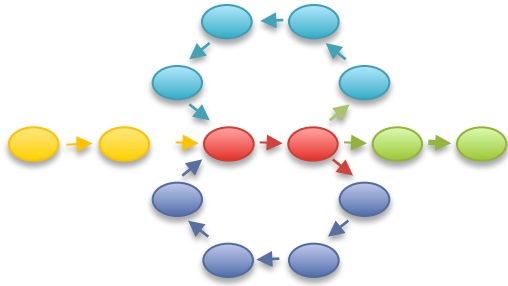
An example configuration file:

```
[LIB]
avg_ins=180
reverse_seq=0
asm_flags=1
rank=1
q1=../Data/original/frag_1.cor.fastq
q2=../Data/original/frag_2.cor.fastq

[LIB]
avg_ins=3500
reverse_seq=1
asm_flags=2
rank=2
q1=../Data/original/shortjump_1.cor.fastq
q2=../Data/original/shortjump_2.cor.fastq
```

# SOAPdenovo tips & tricks

- Input
  - Proper choice of K
  - Good error correction go reads to simplify SOAPdenovo's de Bruijn graph
- Execution
  - Can use different sets of reads for contig construction and for assembly
  - Simple and straightforward configuration file
  - Very fast, but high peak memory usage
- Output
  - asm.contig: initial contigs \*before\* scaffolding
  - asm.scafSeq: linear scaffolds
  - SOAPdenovo.log: Lots of status messages, intermediate files

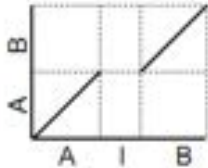


## Whole Genome Alignment with MUMmer

# SV Types

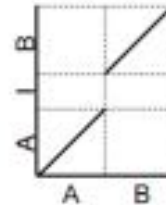
Insertion into Reference

R: A|B  
Q: AB



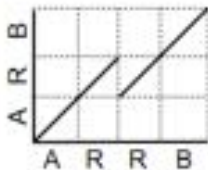
Insertion into Query

R: AB  
Q: A|B



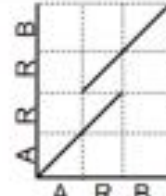
Collapse Query

R: ARRB  
Q: ARB



Collapse Reference

R: ARB  
Q: ARRB



Collapse Query  
w/ Insertion

R: ARIRB  
Q: ARB

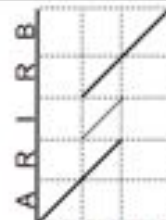
Exact tandem  
alignment if I=R



Collapse Reference  
w/ Insertion

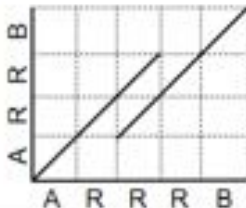
R: ARB  
Q: ARIRB

Exact tandem  
alignment if I=R



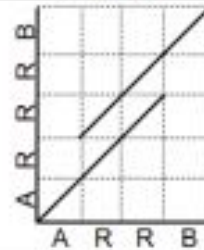
Collapse Query

R: ARRRB  
Q: ARRB



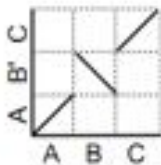
Collapse Reference

R: ARRB  
Q: ARRRB



Inversion

R: ABC  
Q: AB'C



Rearrangement  
w/ Disagreement

R: ABCDE  
Q: AFCBE



- Different structural variation types / misassemblies will be apparent by their pattern of breakpoints
- Most breakpoints will be at or near repeats
- Things quickly get complicated in real genomes

<http://mummer.sf.net/manual/AlignmentTypes.pdf>

# WGA Alignment

**nucmer -maxmatch ref.fasta qry.fasta**

-maxmatch Find maximal exact matches (MEMs)

**delta-filter -m out.delta > out.filter.m**

-m Many-to-many mapping

**show-coords -rcl out.delta.m > out.coords**

-r Sort alignments by reference position

-c Show percent coverage

-l Show sequence lengths

**show-aligns -rcl out.delta.m REFID QRYID > out.aligns**

**dnadiff out.delta.m**

Construct catalog of sequence variations

**mummerplot --large --layout out.delta.m**

--large Large plot

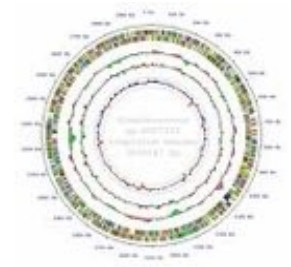
--layout Nice layout for multi-fasta files

--x11 Default, draw using x11 (--postscript, --png)

\*requires gnuplot

See manual at <http://mummer.sourceforge.net/manual>

# Resources



- Assembly Competitions
  - Assemblathon: <http://assemblathon.org/>
  - GAGE: <http://gage.cbc.umd.edu/>
- Assembler Websites:
  - ALLPATHS-LG: <http://www.broadinstitute.org/software/allpaths-lg/blog/>
  - SOAPdenovo: <http://soap.genomics.org.cn/soapdenovo.html>
  - Celera Assembler: <http://wgs-assembler.sf.net>
- Tools:
  - MUMmer: <http://mummer.sourceforge.net/>
  - Quake: <http://www.cbc.umd.edu/software/quake/>
  - AMOS: <http://amos.sf.net>

# Acknowledgements

## Schatzlab

Mitch Bekritsky

Matt Titmus

Hayan Lee

James Gurtowski

Anirudh Aithal

Rohith Menon

Goutham Bhat

## CSHL

Dick McCombie

Melissa Kramer

Eric Antonio

Mike Wigler

Zach Lippman

Doreen Ware

Ivan Iossifov

## JHU

Steven Salzberg

Ben Langmead

Jeff Leek

## NBACC

Adam Phillippy

Sergey Koren

## Univ. of Maryland

Mihai Pop

Art Delcher

Jimmy Lin

David Kelley

Dan Sommer

Cole Trapnell





# Thank You

<http://schatzlab.cshl.edu>  
[@mike\\_schatz](mailto:@mike_schatz)