

Assembly Tutorial

Michael Schatz

Nov. 17, 2014

CSHL Adv Sequencing Course

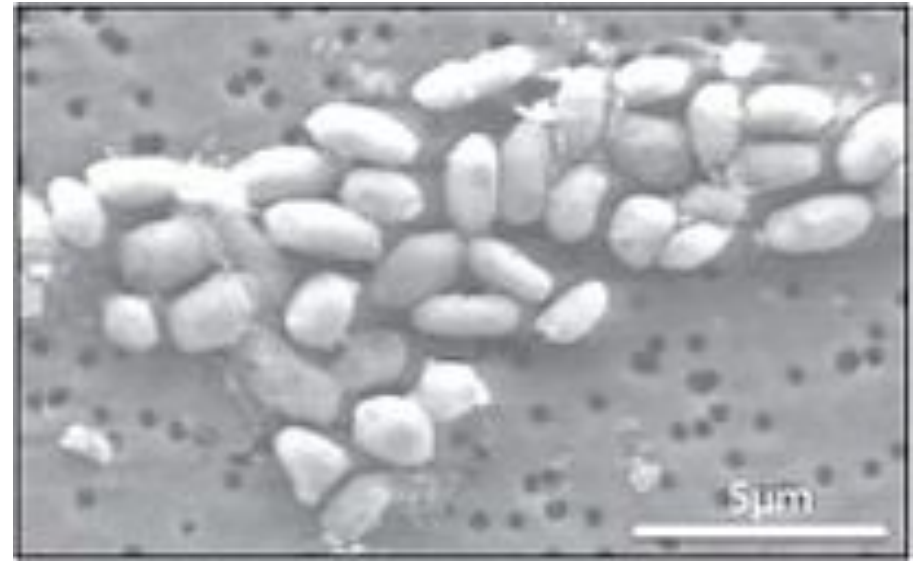


Outline

1. Sample Data for your mission!
2. ALLPATHS-LG
3. MUMmer



Halomonas sp. GFAJ-1



Library 1: Fragment

Avg Read length: 100bp

Insert length: 180bp

Library 2: Short jump

Avg Read length: 50bp

Insert length: 2000bp

A Bacterium That Can Grow by Using Arsenic Instead of Phosphorus

Wolfe-Simon et al (2010) *Science*. 332(6034)1163-1166.

Mission Impossible

1. Setup virtual machine

sudo make install

2. Download Reference Genome

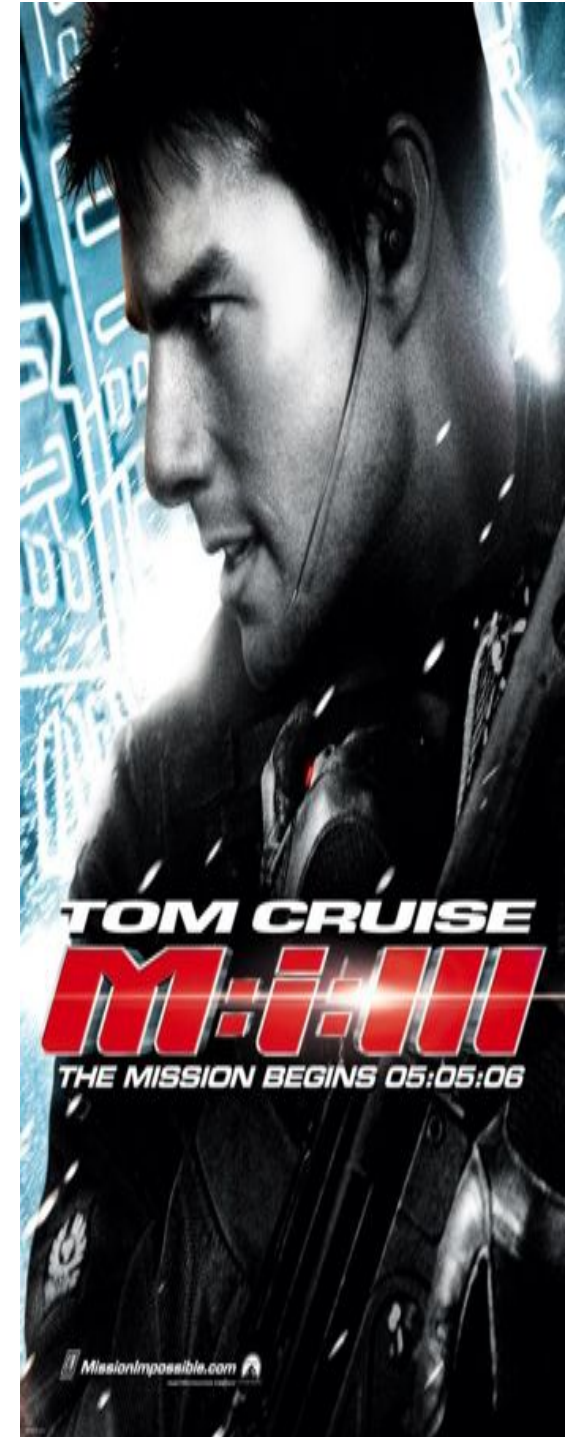
```
wget http://schatzlab.cshl.edu/asm.tgz
```

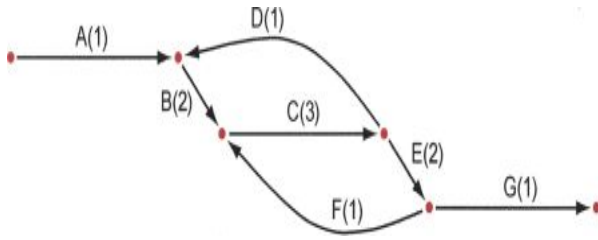
```
tar xzvf asm.tgz
```

3. Decode the secret message

1. Assemble the reads, Align to reference, Extract secret

2. `dna-encode.pl -d`





Running ALLPATHS-LG Iain MacCallum

How to use ALLPATHS-LG

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

ALLPATHS-LG sequencing model

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

*See next slide.

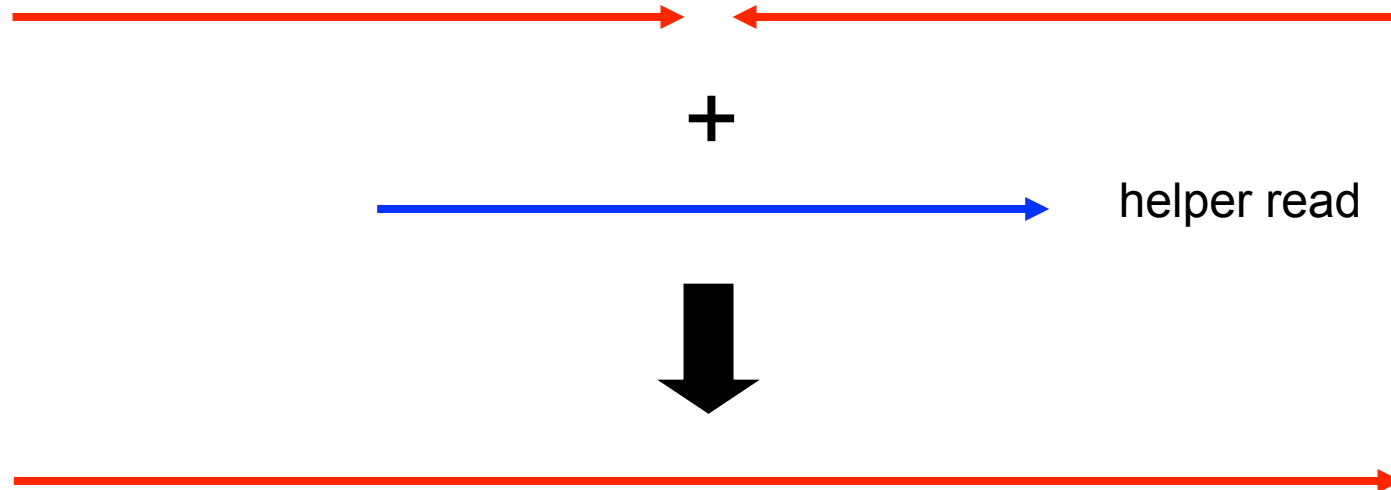
**For best results. Normally not used for small genomes.
However essential to assemble long repeats or duplications.

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

All: protocols are either available, or in progress.

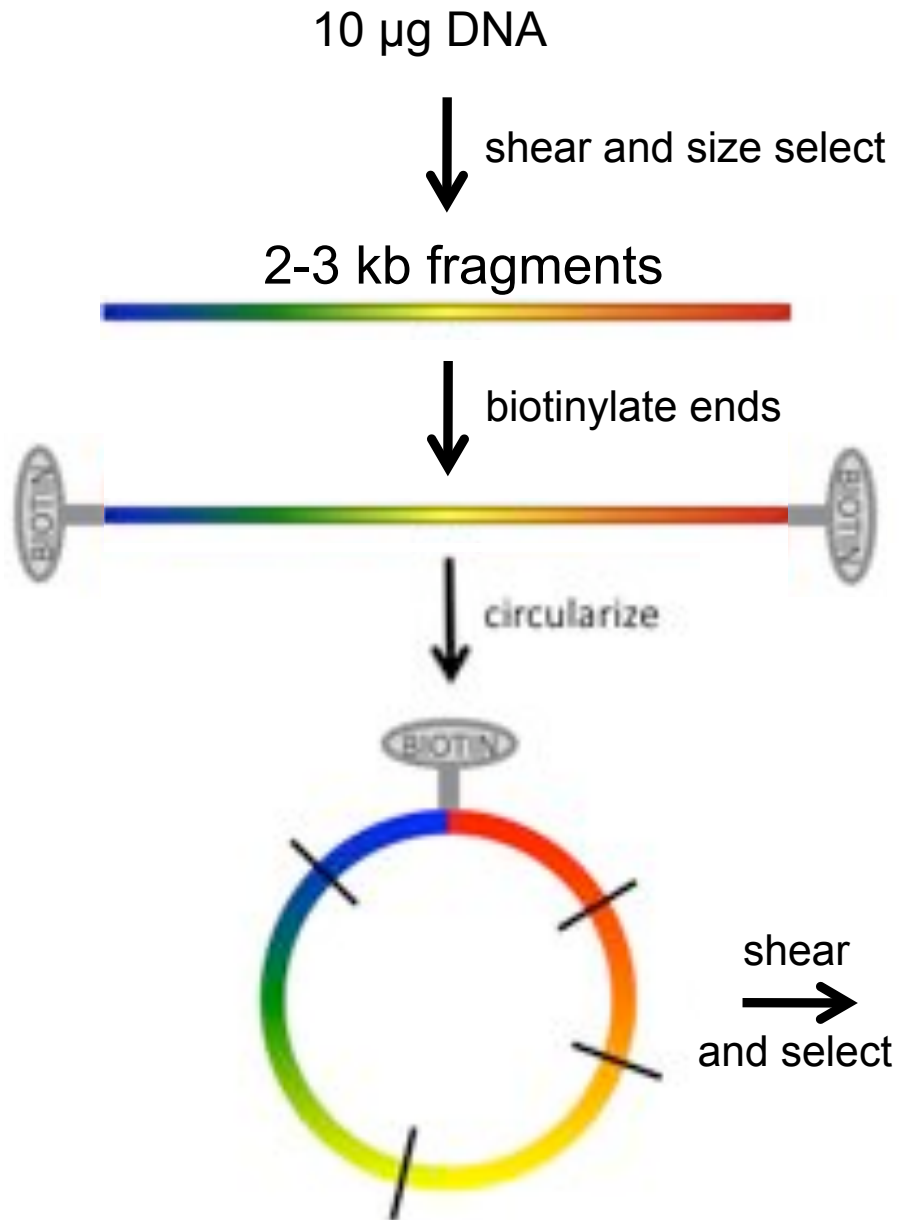
Libraries from 180 bp fragments

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

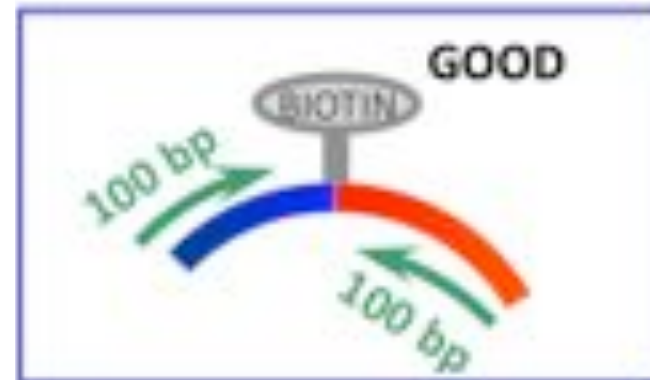


For longer reads, fragment size would be increased proportionally.

Short jumping libraries (2-3 kb)



Illumina protocol, blunt-end ligation



Short jumping libraries (2-3 kb)

Problem 1. 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 2. 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



How to use ALLPATHS-LG

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

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)

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`

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

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 2)

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 ≥ 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 (2 of 2)

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 – 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`.

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

Example:

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

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>
```

- `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.
- `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.

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.

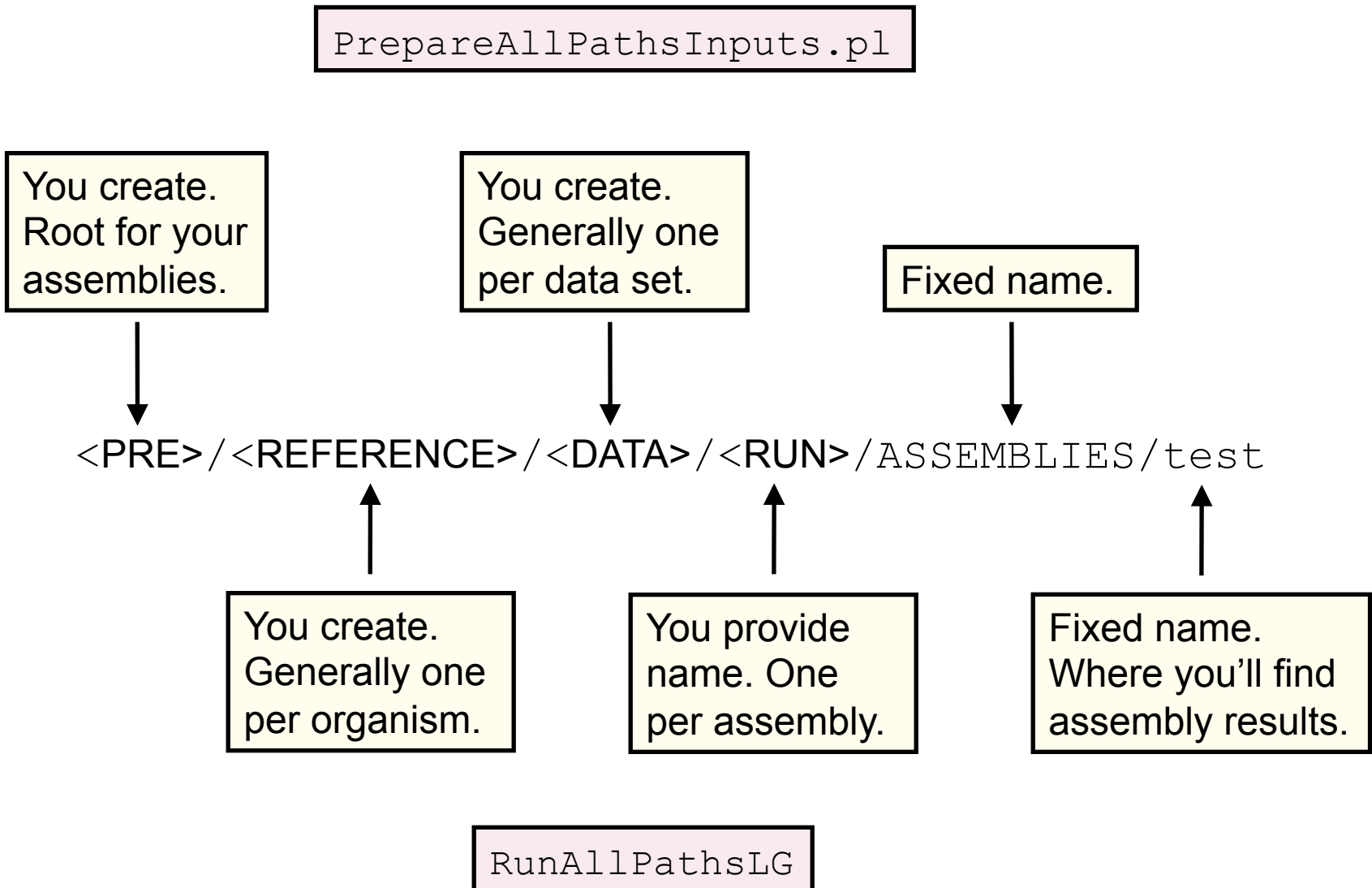
2. Prepare input files

```
% cd /path/to/data/  
% PrepareAllPathsInputs.pl \  
  DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

How to use ALLPATHS-LG

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

ALLPATHS-LG directory structure



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. Prepare input files

```
% PrepareAllPathsInputs.pl \  
  DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

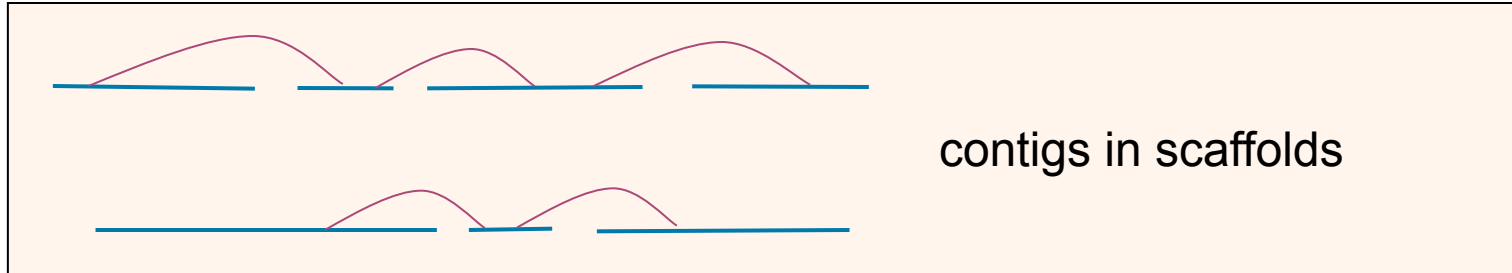
3. Assemble.

```
% RunAllPathsLG \  
  PRE=. REFERENCE_NAME=. \  
  DATA_SUBDIR=. RUN=default THREADS=4 >& run.log
```

How to use ALLPATHS-LG

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

1. Linear assemblies



contig: a contiguous sequence of bases....

```
CTGCCCCCTGTGCCAATGGGTTTGAGGCTCTTCCCACTTCCTTTTCTATTAGATTCAATGTATCTGGTTTTATGTTGAGG
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
```

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

```
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTGAGAATTGCTCTTCCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
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
GTTATAACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN

AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCATGTCTTTGAAGAATTGTGTT
NN

GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAACTTGAAGTTATTGTCATACA

>scaffold_2

CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATTGCTTATGTATGCTATCATATCACTTGCAAA
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCATTGATCTCTTTCTGTTGTCTTATTGTTCTAGCTAACACTT
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTCAAGTGGGATTGCTTCAAGTATG

3. Linearized graph assemblies

Efasta

...ACTGTTT{A,C}GAAAT... A or C at site

...CGCGTTTTTTTTT{T,TT}CAT... 0 or 1 or 2 Ts at site

Example of an assembly in efasta format

```
>scaffold_1
TCCTAGATCCACTTGGACTTGAGCTTTGTATATATATATATATATATA{,TA}CAAGATGACATATATAGGAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCATTGTATATTTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAACTTGAAGTTATTGTCATACA
>scaffold_2
CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATT{A,C,G}GCTTATGTATGCTATCTTGCAA
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCATTGATCTCTTTCTGTTGTCTTATTGTTCTAGCTAACACTT
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTTCAGTGGGATTGCTTCAAGTATG
```

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. Prepare input files

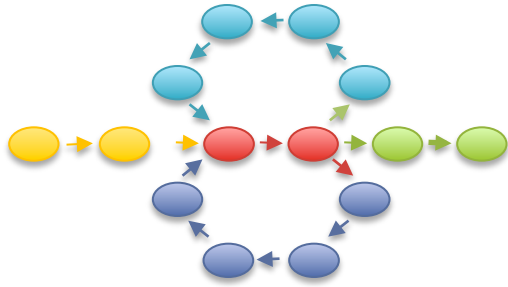
```
% PrepareAllPathsInputs.pl \  
    DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

3. Assemble.

```
% RunAllPathsLG \  
    PRE=. REFERENCE_NAME=. \  
    DATA_SUBDIR=. RUN=default THREADS=4 >& run.log
```

4. Get the results (four files).

```
% cd default/ASSEMBLIES/test/  
% less final.{assembly,contigs}.{fasta,efasta}
```

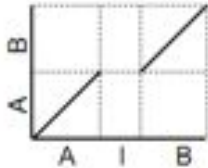


Whole Genome Alignment with MUMmer

SV Types

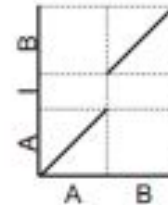
Insertion into Reference

R: AIB
Q: AB



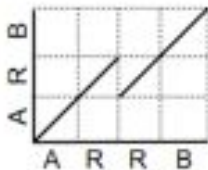
Insertion into Query

R: AB
Q: AIB



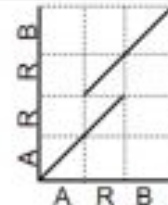
Collapse Query

R: ARRB
Q: ARB



Collapse Reference

R: ARB
Q: ARRB



Collapse Query
w/insertion

R: ARIRB
Q: ARB

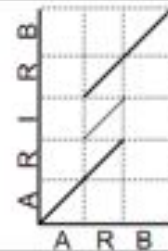
Exact tandem
alignment if I=R



Collapse Reference
w/insertion

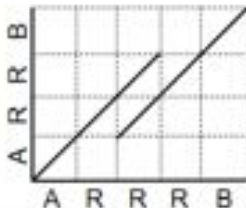
R: ARB
Q: ARIRB

Exact tandem
alignment if I=R



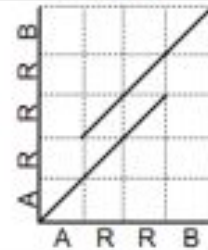
Collapse Query

R: ARRRB
Q: ARRB



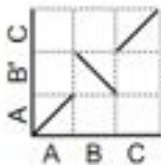
Collapse Reference

R: ARRB
Q: ARRRB



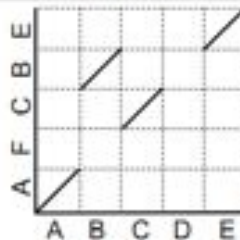
Inversion

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

Find and decode

```
nucmer -maxmatch ref.fasta \  
default/ASSEMBLIES/test/final.contigs.fasta -p refctg  
-maxmatch      Find maximal exact matches (MEMs) without repeat filtering  
-p refctg      Set the output prefix for delta file
```

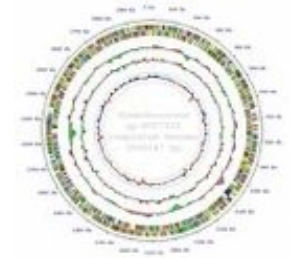
```
mummerplot --layout refctg.delta  
-r      Show the dotplot
```

```
show-coords -rclo refctg.delta  
-r      Sort alignments by reference position  
-c      Show percent coverage  
-l      Show sequence lengths  
-o      Annotate each alignment with BEGIN/END/CONTAINS
```

```
samtools faidx default/ASSEMBLIES/test/final.contigs.fasta
```

```
samtools faidx default/ASSEMBLIES/test/final.contigs.fasta \  
contig_XXX:YYY-ZZZ | ./dna-encode -d
```

Resources



- **Assembly Competitions**
 - Assemblathon: <http://assemblathon.org/>
 - GAGE: <http://gage.cbcb.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.cbcb.umd.edu/software/quake/>
 - AMOS: <http://amos.sf.net>

Questions?

<http://schatzlab.cshl.edu/>

