
American Campus Tree Genomes

Release 0.1

Harkess A, Ficklin S

Jul 21, 2023

CONTENTS:

1	Course on Whole Genome Assembly and Annotation v0.1b	1
1.1	Introduction	1
1.2	Course Setup	6
1.3	Module 1: Plant Genomics	13
1.4	Module 2: Planning a Genome Project	33
1.5	Module 3: Genome Assembly	63
1.6	Module 4: Genome Annotation	105
1.7	Module 5: Comparative Genomics	117
1.8	Module 6: Publishing Scientific Results	119

COURSE ON WHOLE GENOME ASSEMBLY AND ANNOTATION V0.1B

DNA sequencing followed by whole-genome assembly and annotation is a critical first step for understanding the connection between DNA and physical traits of individuals. The [American Campus Tree Genomes \(ACTG\) project](#) is a national effort spearheaded by [Alex Harkess](#) of Auburn University and Hudson Alpha to provide a curriculum that teaches new scientists, through hands-on experience, the computational steps for whole-genome assembly, annotation, and scientific writing that culminates in a research-ready genome assembly, and a publishable manuscript for submission to a scientific journal. The project's focus is on trees that are beloved within a region, state, or university campus, but the course provides training that is applicable for any species. This document provides an implementation of the ACTG curriculum. Students will assemble and annotate a genome for a tree species. Students will contribute content to this manuscript and be included as co-authors on the final submitted manuscript.

1.1 Introduction

1.1.1 Learning Schedule

Warning: Details on this page not complete

Below is a recommended schedule for instructors teaching this course over a 15 week semester. Although instructors can adapt this schedule to the needs of their students.

Self-Instruction

Schedule 1: Monday, Wednesday and Friday Meetings

Schedule 2: Tuesday and Thursday Meetings

1.1.2 Computational Requirements

Warning: Details on this page not complete

Requirements

The following indicates the necessary computational requirements to learn this course. Instructors and self-learners can choose to learn using only the sample data provided for this course or with a new full genome. The requirements for each will be different.

For the Course Sample Data Only

Error: ADD DETAILS HERE

For a Full Genome

Error: ADD DETAILS HERE

Options

The following describes several ways to access the computational resources necessary for this course. Instructors are encouraged to choose the method that best suits the needs of their students and the current compute infrastructure you have available.

The course material will provide instructions throughout for usage of each of the following three options:

Praxis AI

This course is offered via a cloud-based service called [Praxis AI](#). Instructors and students who do not have computational resources onsite for students. Praxis AI integrates educational material with computational resources, conference and collaboration services. The course requires a student fee, equivalent for a text book, for access to these services and provides predictive costs for offering the course. Once the course is completed students can continue to access course content via this site as a free reference.

Advantages

- Compute infrastructure is already available.
- You can upscale the amount of computing power as needed.
- Course material and compute are available within the same systems.
- Praxis AI is a great solution for instructors that do not have ready access to compute.

Limitations

- A fee is required, but at a rate similar to most textbooks.
- The base fee should cover the costs for teaching this course with the sample data but may not be sufficient for an entire genome.

Workstations or Compute Cluster

For instructors and students with access to a high-performance workstations or an institutional compute cluster, we have created [Docker](#) images that contain all of the necessary software required for this class. The Computational infrastructure must have a Docker service or a [Singularity](#) service installed to use these images.

Note: A docker image is a minimal self-contained UNIX/Linux based-operating system pre-configured with a desired set of software.

For learners that have access to an institutional compute cluster you may need to request installation of Singularity if it is not already present.

Note: Instructors: while there are multiple job schedulers available for compute clusters, this course currently only provides instructions for the SLURM scheduler. If your cluster does not use SLURM you may have to adapt the course material to use the scheduler that your cluster provides.

Students: If you are self-training and you have access to a cluster but it does not use SLURM you will need to do a bit of searching to find the corresponding commands for submitting and managing jobs on your cluster.

Advantages

- The software needed for the course is ready to go. No installation is required.
- You can use your own compute infrastructure.
- Students receive training on compute infrastructure they may be using for other purposes in the future.
- No extra costs. Use the compute you already have.
- Everyone uses the same software versions in the same environment despite working on different machines.

Limitations

- Unlike Praxis AI, the course material is not embedded in an interface along with the computational access. Students must connect separately to their respective infrastructure.
- You must ensure that the compute power is sufficient for the course.
- A bit more time is required to train students to use Singularity and Docker.

Set it up Yourself

While Praxis AI and or the pre-configured Docker images make learning easiest, we recognize that some folks prefer to learn the nitty-gritty of bioinformatics software installation. Within the lessons we will provide instructions.

Advantages

- Students learn how to install software.

Limitations

- It is more time consuming
- Students must have the ability to install software on the machine on which they work.

1.1.3 Software Requirements

This course requires a variety of software packages. To ensure students have as similar results to others, this course will use specific versions of those software. The software, version and their purpose are provided below.

Software	Version	Purpose
R	4.1.3	A free software environment for statistical computing and graphics.
FastQC	0.11.9	A quality control tool for high throughput sequence data.
MultiQC	1.13a	MultiQC summarizes the output from numerous bioinformatics tools into a single report.
SRA Tools	2.11.0	A collection of tools and libraries for using data in the INSDC Sequence Read Archives.
FastP	0.23.2	Provides fast all-in-one preprocessing for FastQ file.
SamTools	1.15.1	A suite of programs for interacting with high-throughput sequencing data.
NOVOPlasty	4.3.1	A de novo assembler and heteroplasmy/variance caller for short circular genomes.
BWA	0.7.17	A software package for mapping low-divergent sequences against a large reference genome.
Jellyfish	2.2.10	A tool for fast, memory-efficient counting of k-mers in DNA.
Hifiasm	0.16.1	Hifiasm is a fast haplotype-resolved de novo assembler for PacBio HiFi reads.
Mummer	3.23	Ultra-fast alignment of large-scale DNA and protein sequences.
BedTools	2.30.0	A swiss-army knife of tools for a wide-range of genomics analysis tasks.
RepeatMasker	4.1.2.p1	Screens DNA sequences for interspersed repeats and low complexity DNA sequences
Braker2	2.1.6	Gene structural annotation.
BUSCO	4.1.2	Assesses genome assembly and annotation completeness using single-copy orthologs.
EDTA	2.0.1	Automated whole-genome de-novo TE annotation and benchmarking.
hic_qc	git commit 6881c33	Performs QC Checks for Hi-C libraries using reads in a BAM file aligned to the genome assembly. The version of this software is a commit to the source repository made on June 27, 2022.

1.1.4 Sample Data

Warning: Details on this page not complete

1.1.5 Course Resources

Warning: Details on this page not complete

- [ACTG Slack Workspace](#). This course provides a Slack workspace where students in active, concurrent courses can interact by asking questions, sharing insight and working together to complete the course and assemble and annotate a genome.
- [ACTG Youtube Channel](#). Lecture videos from the lessons of this course are housed on this YouTube channel.

- **ACTG DockerHub Repository** The docker images used for this course are housed in a public DockerHub repository.
- **ACTG GitHub Repository** The source code markup for this online document is maintained on this public GitHub repository. Instructors who wish to contribute may clone this repository, edit content and push their changes as pull requests.

1.1.6 History

Warning: Details on this page not complete

Active instruction of this course occurred for the following genomes

Course Versions

As this course is updated new versions will be assigned.

- Version 0.1b (current)
- Version 0.1a The first implementation of this course. First offered in 2021 at [Auburn University](#) by Alex Harkess.

Assembled Genomes

The following genomes have been (or are being) assembled by students of this course.

Malus x domestica WA38 (Cosmic Crisp © apple)

Institution	Washington State University , Pullman WA.
Instructors	Huiting Zhang, Stephen Ficklin
Course Number	HORT 503
When	Fall Semester
Course Version	v0.1b

Pyrus communis Anjou Pear

Institution	Auburn University , Auburn AL.
Instructors	Alex Harkess
Course Number	CSES 7120
When	Fall Semester
Course Version	v0.1b

Official Offerings in 2021

Quercus virginiana Toomers (Live Oak)

Institution	Auburn University , Auburn AL.
Instructors	Alex Harkess
Course Number	CSES 7120
When	Fall Semester
Course Version	v0.1a

1.1.7 How to Contribute

This is an open-source developed course. Others are welcome to contribute updates to the material.

Warning: Add instructions here for how someone can contribute content to this course. What are the rules, how is it reviewed, etc.

About the “Restructured Text” Format

The material for this course is written in an easy to read and write [Restructured Text](#) markdown format. To provide edits or comments please review the [Users Guide](#) for instructions about this format.

1.1.8 Need Help?

Warning: Add information here about how instructors or students can get help if the instructions provided here don't work.

1.2 Course Setup

This course will require both computational infrastructure and software to complete. Choose from one of the infrastructure types for this course described in the following sections. Depending on the selection you make, you may or may not have to install software. Instructions for installing the necessary software are provided.

1.2.1 Computational Infrastructure

The following provides instructions for setup of this course on computing resources. Choose the method that best suits your needs or those of your students.

If you have an instructor for this course, the instructor will advise you on the appropriate option to use and these instructions will serve as a reference for you.

Setup for Praxis AI (Cloud-Based)

Setup on a SLURM Cluster

Setup for a Stand-Alone Workstation Using Docker

Setup for a Stand-Alone Workstation Without Docker

1.2.2 Software Installation and Usage

The following sections provide instructions for how to prepare your computational infrastructure for this course. Choose the method that is most appropriate for your computational infrastructure.

If you have an instructor for this course, the instructor will advise you on the appropriate option to use and these instructions will serve as a reference for you.

Note: To ensure that all students have as similar experiences as possible, these instructions specify software versions. If you install your own software be sure to use the version numbers specified for the course to ensure your results match as closely to those shown.

Software Installation: Praxis AI

If you are using Praxis AI for this course you will be instructed to install individual software as you need it using conda. Conda makes software installation easy. You'll use a similar set of commands to install everything. See the example for how to find a package and install it on the [Software Installation: Conda](#) page.

Software Installation: Conda

Conda Installation

Conda is a package manager which is amazingly powerful and simple to use. If conda is not already on your compute infrastructure, and if you have permission to install software, then the easiest way to get conda is to install [Anaconda](#). You can find a variety of Anaconda installation instructions for different computational platforms [here](#)



If you do not have permission to install Anaconda (or other Conda provider) then work with your systems administrator to get conda available.

Quick How-To

There is an entire collection of biology-related software that has been deposited into a “channel” of **conda** called bioconda. Check out all the available software packages you can install at the [bioconda package repository](#) — more than 7,000 and growing.

As a quick example for how to install software using Conda. Search in the bioconda repository for a program called **fastqc**. The website shows us exactly how to install the program:

bioconda / **packages** / **fastqc** 0.11.9

  12





A quality control tool for high throughput sequence data.

Conda

Files

Labels




Badges

 License: [GPL >=3](#)
 Home: <http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>
 382050 total downloads
 Last upload: 4 months and 15 days ago

Installers

Info: This package contains files in non-standard [labels](#).

conda install ?

 linux-64 v0.11.8
 osx-64 v0.11.8
 noarch v0.11.9

To install this package with conda run one of the following:

```
conda install -c bioconda fastqc
conda install -c bioconda/label/broken fastqc
conda install -c bioconda/label/cf201901 fastqc
```

To install it run the `conda install` command as shown on the site:

```
conda install -c bioconda fastqc
```

You'll probably get a message asking if you want to install some other dependencies (other programs that fastqc relies on). It will look like this:

Proceed ([y]/n)?

Note: Often when you see messages like this in UNIX-based operating system, the brackets around [y] mean that if you just press enter, it will assume you mean "yes". In other words, [y] is the default assumed response.

Did it work? Run fastqc with the -h (help) flag and see:

```
fastqc -h
```


Base Software Installation

This course uses a variety of software packages listed on the [Software Requirements](#) page. Rather than install them all we will use an environment file to install them with conda. To do this, create a new file named `base.environment.yml`. And add the following contents:

```
channels:
- conda-forge
- bioconda
- defaults
dependencies:
- conda-forge::r-base=4.1.3
- bioconda::fastqc=0.11.9
- bioconda::multiqc=1.13a
- bioconda::sra-tools=2.11.0
- bioconda::fastp=0.23.2
- bioconda::samtools=1.15.1
- bioconda::bwa=0.7.17
- bioconda::jellyfish=2.2.10
- bioconda::hifiasm=0.16.1
- bioconda::mummer=3.23
- bioconda::bedtools=2.30.0
- bioconda::repeatmasker=4.1.2.p1
```

The contents above are in [YAML format](#) and are instructions that can be used by conda to perform bulk installation of software. The following explains the meaning of the elements of the file:

- **name:** tells conda what the name of the environment is. You will use this name to access the software after installation.
- **channels:** tells conda what online repositories to use to find software
- **dependencies:** lists the software that should be installed. The list includes the name of the channel (e.g. bioconda), the software name, and a specific version to install.

You can install all of these software into the “base” environment with this command:

```
conda env update --quiet -n base -f /base.environment.yml
```

Additional Software Installation

Some of the software tools are not part of the “base” environment because they work best in their own self-contained environment. You can install the remaining software in this way:

Braker2

Step 1: Create the following YAML file named *braker.environment.yml*

```
name: braker
channels:
- conda-forge
- bioconda
- defaults
```

(continues on next page)

(continued from previous page)

```
dependencies:
- anaconda::perl
- bioconda::perl-app-cpanminus
- bioconda::perl-hash-merge
- bioconda::perl-parallel-forkmanager
- bioconda::perl-scalar-util-numeric
- bioconda::perl-yaml
- bioconda::perl-class-data-inheritable
- bioconda::perl-exception-class
- bioconda::perl-test-pod
- anaconda::biopython
- bioconda::perl-file-which
- bioconda::perl-mce
- bioconda::perl-threaded
- bioconda::perl-list-util
- bioconda::perl-math-utils
- bioconda::cdbtools
- bioconda::braker2=2.1.6
```

Step 2: Create the new braker environment

```
conda env create --quiet -f braker.environment.yml
```

Step 3: When you want to run braker run the following to enable the environment and then run any of the braker programs:

```
conda activate braker
braker.pl
```

You can use any of the software in the base environment by switching back:

```
conda activate base
```

EDTA

Step 1: Create the following YAML file named *edta.environment.yml*

```
name: edta
channels:
- conda-forge
- bioconda
- defaults
dependencies:
- bioconda::edta=2.0.1
```

Step 2: Create the new braker environment

```
conda env create --quiet -f edta.environment.yml
```

Step 3: When you want to run EDTA run the following to enable the environment and then run any of the EDTA programs:

```
conda activate edta
EDTA.pl
```

You can use any of the software in the base environment by switching back:

```
conda activate base
```

BUSCO

Step 1: Create the following YAML file named *busco.environment.yml*

```
name: busco
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - bioconda::busco=4.1.2
```

Step 2: Create the new BUSCO environment

```
conda env create --quiet -f busco.environment.yml
```

Step 3: When you want to run BUSCO run the following to enable the environment and then run any of the BUSCO programs:

```
conda activate busco
busco
```

You can use any of the software in the base environment by switching back:

```
conda activate base
```

Software Usage: Docker

This course provides a pre-configured [Docker](#) image containing all of the software. This means you do not have to install any software on your computational infrastructure! You can use this image if you have either Docker or [Singularity](#) installed. If you have permission to install software on your infrastructure you may need to install either Docker or Singularity. If not, work with your systems administrator.

Note: Docker requires root (or administrative) access to the machine. Thus it is often not available on an institutional compute cluster. Singularity, however, does not and is often used in cases where root access is not given to end-users.

Docker or Singularity Installation

- If you want to install Docker you can find installation instructions [here](#).
- If you want to install Singularity you can find installation instructions [here](#):

How to Run Software with Docker

If you use Docker, anytime software is referenced in this course you can run it by following this example:

```
docker run -v ${PWD}:/work -u $(id -u ${USER}):$(id -g ${USER}) \
systemsgenetics/actg-wgaa:0.1 <command>
```

Note: The backslash character \ at the end of the first line tells the UNIX command-line that your instruction spans more than one line. This is not necessary but makes it easier to read and cut-and-paste from documentation!

The following is the meaning of each component in that command:

- The `-v ${PWD}:/work` argument instructs Docker to include the current directory on your current machine as a new directory inside of the image and available at the path `/work`.
- The Docker image is named `systemsgenetics/actg-wgaa:0.1` and Docker will download it if you've never used that image before. So be patient the first time you run the command.
- The `-u $(id -u ${USER}):$(id -g ${USER})` argument instructs Docker to run any commands in the image as your local account on the UNIX system you are using.
- Replace the `<command>` placeholder with the exact command you want to run.

As an example, you can print out the version of R installed with the following:

```
docker run -v ${PWD}:/work -u $(id -u ${USER}):$(id -g ${USER}) \
systemsgenetics/actg-wgaa:0.1 R --version
```

How to Run Software with Singularity

If you use singularity, anytime software is referenced in this course you can run it using Singularity by following this example:

```
singularity exec -B ${PWD}:/work docker://systemsgenetics/actg-wgaa:0.1 <command>
```

The following is the meaning of each component in that command:

- The `-v ${PWD}:/work` argument instructs Docker to include the current directory on your current machine as a new directory inside of the image and available at the path `/work`.
- The Docker image is named `systemsgenetics/actg-wgaa:0.1` and Singularity will download it if you've never used that image before. So be patient the first time you run the command. So be patient the first time you run the command.
- Replace the `<command>` placeholder with the exact command you want to run.

As an example, you can print out the version of R installed with the following:

```
singularity exec -B ${PWD}:/work docker://systemsgenetics/actg-wgaa:0.1 R --version
```

How to Run Docker in Interactive Mode

The instructions in the previous sections described how to run software in a Docker image directly on the command-line. However, if you like, you can enter inside of the image and use the command-line terminal as if it were a stand-alone machine.

Interactive Mode with Docker

To run software for this course in Docker in interactive mode use the following command to enter the image and then change into the work folder.

```
docker run -it -v ${PWD}:/work -u $(id -u ${USER}):$(id -g ${USER}) \
systemsgenetics/actg-wgaa:0.1 /bin/bash

cd work
```

Interactive with Singularity

To run software for this course in Singularity in interactive mode use the following command to enter the image and then change into the work folder.

```
singularity shell -B ${PWD}:/work docker://systemsgenetics/actg-wgaa:0.1

cd work
```

1.3 Module 1: Plant Genomics

1.3.1 Lesson 1: Introduction to Plant Genomics

1.1 Instructions

1.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Linux Survival](#)
- [Linux Cheat Sheet](#)

1.1 Lesson

Learning Objectives

Vocabulary

Learning Material

Error: EMBED LECTURE VIDEO HERE

Error: ADD TEXT TRANSCRIPTION OF VIDEO HERE

1.1 Lab Exercises

Overview

In this lab, we will get familiar with our new Virtual Machine (VM) terminal. If you've got unix and command line experience, great! If you don't, or it's been a while, don't stress. Think of this VM like a personal sandbox. It's yours, it never goes away, and you can't break anything (too badly).

We will do four major things in this lab:

- Get familiar with your new VM
- Learn some basic unix commands: ls, cd, mkdir, pwd
- Download a genome from TAIR
- Explore that genome with grep

Give it a go, be patient, and ask questions.

"Roads were made for journeys, not destinations" - Confucius

Task A: Get comfortable on the command line

Step 1. Where am I?

Help! I'm lost! Where am I? The UNIX command `pwd` is your lighthouse.

```
pwd
```

If this tree-like directory structure doesn't make sense, stop and review Module 1 of [LinuxSurvival](#)

Use `ls` to list the files in your current directory:

```
ls
```

Add a "flag" to `ls` to see more information about every file. `-l` stands for "long format".

```
ls -l
```

What are all the flags you can use for a given command? Read the manual for every UNIX command by using the man command:

```
man ls
```

Step 2. Make a new directory for this lab

We use `mkdir` to make a new directory (folder). The usage is two parts:

```
mkdir <dirname>
```

Replace `<dirname>` with what you want to call this directory. Make sure it is one word, no spaces. I'll use "Lab1" but organize your life however you'd like to.

```
mkdir Lab1
```

There's a nice trick we can use to speed up our command line life, called tab completion. The tab key is your best friend in UNIX; it is similar to how Google will try and autocomplete text for you while you're typing into the search bar. If you start typing a filename in UNIX, and press the tab key, UNIX will try to complete the filename or path for you as long as it is unique.

We want to change directories into Lab1 now using the `cd` command, but we also want to be lazy. We *could* type out the full command:

```
cd Lab1
```

Or, we could just type:

```
cd La
```

and then press the tab key to complete the word. Try it, and press enter to execute the `cd` command.

Did it work? Use `pwd` to see where you are.

This trick works with just about anything you're typing, like programs, filenames, scripts, and commands.

Task B: Download the *Arabidopsis thaliana* genome from TAIR

Arabidopsis is a powerful model for plant biology. It is not perfect, and is not useful in every situation. After all, there are >300,000 species of land plants on the planet, so how could one species possibly be useful to understanding another?

Step 1. Download the genome for *Arabidopsis thaliana*

The unix command `wget` allows us to fetch data from servers. Not every UNIX command means something, but `wget`'s name is derived from World Wide Web + get = `wget`. Here's how we use it:

```
wget https://www.arabidopsis.org/download_files/Genes/TAIR10_genome_release/TAIR10_
↳ chromosome_files/TAIR10_chr_all.fas
```

That's it, just two parts: `wget [path-to-what-we-want-to-fetch]`



Fig. 1: Image source: Plantlet.org, Credit: Eric Belfield

Step 2. Let's see what the genome looks like

Use the command `less` to open up the FASTA file:

```
less TAIR10_chr_all.fas
```

This is what FASTA format looks like. FASTA format contains two major parts:

1. A header that starts with “>” and includes information about
2. The sequence on the next line(s). Sometimes the header can have information about the chromosome number (as you see here). Other genomes are not so perfect, and might be in hundreds or thousands of pieces.

Just like in Microsoft Word, you can use another UNIX program to find words or characters. This is really helpful if we just want to look at every line that has a FASTA header with the “>” character.

```
grep ">" TAIR10_chr_all.fas
```

The Arabidopsis genome is incredibly high quality, since people have been improving it for nearly 20 years. You should see FASTA headers for 5 nuclear chromosomes, one chloroplast genome, and one mitochondrial genome.

Step 3. View gene annotation sequences in a FASTA file

Use your new set of UNIX vocabulary to download the peptide sequences for Arabidopsis. Here's the link:

```
https://www.arabidopsis.org/download_files/Sequences/Araport11_blastsets/Araport11_genes.  
↪202106.pep.fasta.gz
```

This file ends in “.gz”. This means that it is compressed using a program called `gzip`. This is a very common and nifty compression tool, just like .zip files on Windows and MacOS. To decompress this file, all we need to do is:

```
gzip -d filename
```

The `-d` flag means “decompress”. What if we want to compress something?

```
gzip filename
```

Mastering Content

Step 1

Count the number of genes in the Arabidopsis peptide fasta file.

Hint: You know how to use `grep` now. Is there a flag you can add to `grep` that will count things for you? Use `man` and/or Google. If you get stuck, rely on your colleagues, friends, and classmates in the discussion forum — this is real life, after all.

Step 2

Plants have canonical repeat motifs at their telomeres, usually “CCCTAAA” for most monocots and eudicots (side note: monocots in the Asparagales order often have “CCCTAA” telomere repeats, like humans).

Count the number of times that the string “CCCTAAA” occurs in the genome fasta file. Is this a robust way to measure of the length of telomeres in Arabidopsis?

1.3.2 Lesson 2: Introduction to Biological Computing

1.2 Instructions

1.2 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Conda and Bioconda](#)
- [Conda CheatSheet](#)

1.2 Lesson

Learning Objectives

Vocabulary

Learning Material

Error: EMBED LECTURE VIDEO HERE

Error: ADD TEXT TRANSCRIPTION OF VIDEO HERE

1.2 Lab Exercises

Overview

In this lab, we will learn how to Conda, a package manager which makes installing and running software very simple.

We will do three major things in this lab:

- Download and install **fastqc** with **conda**
- Download some Illumina data from SRA
- Run **fastqc** on the raw Illumina data

Give it a go, be patient, and ask questions.

“I am not discouraged, because every wrong attempt discarded is another step forward.” - Thomas Edison

Task A: Download and run FASTQC

Step 1. Use Conda to install a package

Conda is amazingly powerful and simple to use. There is an entire collection of biology-related software that has been deposited into a “channel” of **conda** called bioconda. Check out all the available software packages you can install at the [bioconda package repository](#) — more than 7,000 and growing.

Search for a program called **fastqc**. The website shows us exactly how to install the program:

The screenshot shows the bioconda website for the **fastqc** package (version 0.11.9). The page has a header with the bioconda logo and navigation links (home, star, 12). Below the header, it says "A quality control tool for high throughput sequence data." There are tabs for Conda, Files, Labels, and Badges. The Conda tab is selected, showing details like License: GPL >=3, Home: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/, 382050 total downloads, and Last upload: 4 months and 15 days ago. Below this is an "Installers" section with an info box stating "This package contains files in non-standard labels." It lists installers for linux-64 (v0.11.8), osx-64 (v0.11.8), and noarch (v0.11.9). At the bottom, it says "To install this package with conda run one of the following:" and provides three commands: `conda install -c bioconda fastqc`, `conda install -c bioconda/label/broken fastqc`, and `conda install -c bioconda/label/cf201901 fastqc`.

I usually just google something like “conda fastqc” and it’s always the first result. Now install it, just like the website says:

```
conda install -c bioconda fastqc
```

You’ll probably get a message asking if you want to install some other dependencies (other programs that fastqc relies on). It will look like this:

Proceed ([y]/n)?

Whenever you see messages like this in unix, the brackets around [y] mean that if you just press enter, it will assume you mean “yes”. In other words, [y] is the default assumed response.

Did it work? Run fastqc with the -h (help) flag and see:

PraxisAI

```
fastqc -h
```

Stand-alone

```
fastqc -h
```

Singularity

```
singularity exec -B ${PWD} docker://systemsgenetics/actg-wgaa:0.1 \
fastqc -h
```

Docker

```
docker run -v ${PWD} -u $(id -u ${USER}):$(id -g ${USER}) systemsgenetics/actg-wgaa:0.1 \
fastqc -h
```

Help

Here you find, on several different tabs, the command-line instruction to execute this step of the lab on your computational infrastructure. Depending on how this course has been setup the instruction will vary. Please see the [Computational Requirements](#) page for information. If you are unsure which instruction to use contact your instructor.

Step 2. Download some Illumina data

How do we store sequencing data? NCBI's [Sequence Read Archive \(SRA\)](#) is the dominant repository for sequencing data. It is free to use in every sense: free to upload data, free to download data, free to explore. Let's start at the main SRA page. I got here just by Googling "NCBI SRA".

Search for data from one of my favorite plant species, *Spirodela polyrhiza*, otherwise known as a duckweed. Find one of the entries that says "WGS duckweed". WGS means "Whole Genome Shotgun", as in randomly sequenced DNA from the genome. I picked this one: [https://www.ncbi.nlm.nih.gov/sra/SRX9007723\[accn\]](https://www.ncbi.nlm.nih.gov/sra/SRX9007723[accn])

Look through the whole SRA page; there is a lot of metadata attached to this sample.

SRX9007723: WGS duckweed

1 ILLUMINA (Illumina HiSeq 2000) run: 29M spots, 5.8G bases, 2.4Gb downloads

Design: Sequencing at McGill innovation centre

Submitted by: University of Toronto

Study: Duckweed mutation accumulation lines (*Spirodela polyrhiza*)

[PRJNA659313](#) • [SRP278761](#) • [All experiments](#) • [All runs](#)

[show Abstract](#)

Sample:

[SAMN15903372](#) • [SRS7259414](#) • [All experiments](#) • [All runs](#)

Organism: [Spirodela polyrhiza](#)

Library:

Name: 10

Instrument: Illumina HiSeq 2000

Strategy: WGS

Source: GENOMIC

Selection: RANDOM

Layout: PAIRED

Runs: 1 run, 29M spots, 5.8G bases, [2.4Gb](#)

Run	# of Spots	# of Bases	Size	Published
SRR12517164	29,038,715	5.8G	2.4Gb	2020-09-01

We know what machine the data was sequenced on (HiSeq 2000), that this is WGS Whole Genome Shotgun (as opposed to e.g. amplicon sequencing or RNA-seq), that this comes from Genomic DNA, and that the data are paired-end (meaning two reads per spot on the flow cell). Click on the SRR Run for more info and a preview of the data.

For Illumina sequencing, paired-end means that each DNA molecule was sequenced from both ends, producing two reads per spot/molecule. We will cover this more in the coming weeks, but here's a visualization of the DNA fragment (grey), sequence read 1 (orange), and sequence read 2 (blue).

If you want to learn more, watch this short 5 minute video on Illumina Sequencing-by-Synthesis

The data we really need is the SRR number that specifies the run. Luckily, NCBI has written some software tools called **sra-tool** that allow us to quickly download data from SRA once we know this SRR number.

Use **conda** to install **sra-tools** on your own, then make a new directory for this lab. Name it whatever you want, but stay consistent so that your labs are organized and your home directory is not super cluttered. If you can't remember how to make a new directory, go back to the UNIX cheat sheet in the [Lesson 1 Resources](#).

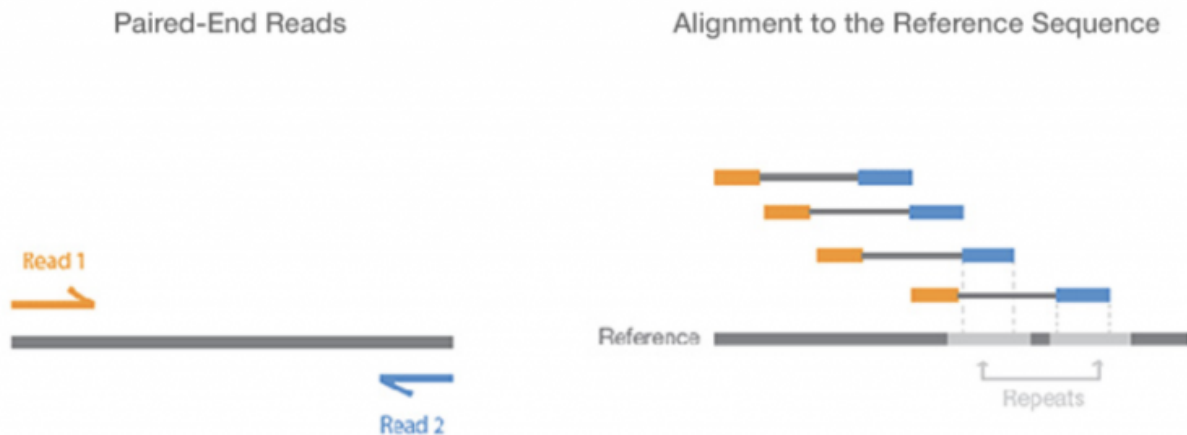
Usually we would download the entire dataset. For this lab, we'll just download 20 million read pairs from this dataset to save time. Check out the options for **fastq-dump** using the **-h** flag. This admittedly is not the best documented software, and some of the options are pretty confusing. For data that is paired-end, we need to add the **-split-files** flag.

To download this paired-end Illumina data, copy/paste the SRR number into the **fastq-dump** command:

PraxisAI

```
fastq-dump -X 200000000 --split-files SRR12517164
```

Stand-alone



Paired-end sequencing enables both ends of the DNA fragment to be sequenced. Because the distance between each paired read is known, alignment algorithms can use this information to map the reads over repetitive regions more precisely. This results in much better alignment of the reads, especially across difficult-to-sequence, repetitive regions of the genome.

Fig. 2: Image Source: [Illumina Website](#)

```
fastq-dump -X 200000000 --split-files SRR12517164
```

Singularity

```
singularity exec -B ${PWD} docker://systemsgenetics/actg-wgaa:0.1 \
  fastq-dump -X 200000000 --split-files SRR12517164
```

Docker

```
docker run -v ${PWD} -u $(id -u ${USER}):$(id -g ${USER}) systemsgenetics/actg-wgaa:0.1 \
  fastq-dump -X 200000000 --split-files SRR12517164
```

Help

Here you find, on several different tabs, the command-line instruction to execute this step of the lab on your computational infrastructure. Depending on how this course has been setup the instruction will vary. Please see the [Computational Requirements](#) page for information. If you are unsure which instruction to use contact your instructor.

Great! Well, mostly. We're twiddling our thumbs now since this program is running and we can't use the command line. Let's shove this job into "the background" so we can use our command line again. Press "Control + Z" to pause the job, and then push the job into the background using `bg`.

```
bg
```

Now we've got our command line back. We can see what jobs are running in the background using `jobs`:

```
jobs
```

```
(base) student@landingvm:~$ jobs
[1]+  Running                  fastq-dump --split-files SRR12517164 &
(base) student@landingvm:~$
```

See how it displays the **fastq-dump** command you entered? This job is now running “in the background”. The ampersand at the end (&) is a nifty thing. We could have saved ourselves some time by running the **fastq-dump** command with an ampersand & at the end, which would automatically start the job in the background.

Data transfer from SRA is not blazing fast, though. Check on the progress of your data transfer using:

```
ls -lhrt
```

You can mix and match multiple flags onto UNIX commands. Let’s break this one down:

ls = list all the files in my current directory

- **-l** = long format (show permissions, date last touched)
- **-h** = human readable file sizes. I like this option because it shows me 2G instead of 2000000 for the file size. K=kilo, M=mega, G=giga, T=tera.
- **-t** = sort the files by the time of their last modification
- **-r** = reverse the order, putting the “newest” files at the bottom. These last two options, **-rt**, make it really quick to see how much of your file has been downloaded. It’s especially nice when you have a lot of files in one directory.

Step 3: Look at our fastq files

We have two files that end in **.fastq** in our directory. They differ in a small but important way: **_1.fastq** and **_2.fastq**. These two files belong to the same sequencing run, and represent read1 (**_1.fastq**) and the read2 (**_2.fastq**) for every single sequenced molecule. We’ll talk more about fastq format soon, but go ahead and look at the files. You can quickly look at the first few lines of a file using **head**.

```
head SRR12517164_1.fastq
```

Illumina describes the fastq file as:

For each cluster that passes filter, a single sequence is written to the corresponding sample’s R1 FASTQ file, and, for a paired-end run, a single sequence is also written to the sample’s R2 FASTQ file. Each entry in a FASTQ files consists of 4 lines:

1. A sequence identifier with information about the sequencing run and the cluster. The exact contents of this line vary by based on the BCL to FASTQ conversion software used.
2. The sequence (the base calls; A, C, T, G and N).
3. A separator, which is simply a plus (+) sign.
4. The base call **quality scores**. These are Phred +33 encoded, using **ASCII** characters to represent the numerical quality scores.

Now we’ve got data and we’ve got fastqc installed. Let’s run **fastqc**.

Task B: Run FASTQC and assess the quality of some Illumina shotgun data

FASTQC is a simple program that allows us to objectively measure some statistics about a sequencing run. From the [FASTQC github page](#):

“FastQC is a program designed to spot potential problems in high throughput sequencing datasets. It runs a set of analyses on one or more raw sequence files in fastq or bam format and produces a report which summarizes the results.”

Step 1: Check out the help options for fastqc

PraxisAI

```
fastqc -h
```

Stand-alone

```
fastqc -h
```

Singularity

```
singularity exec -B ${PWD} docker://systemsgenetics/actg-wgaa:0.1 \
fastqc -h
```

Docker

```
docker run -v ${PWD} -u $(id -u ${USER}):$(id -g ${USER}) systemsgenetics/actg-wgaa:0.1 \
fastqc -h
```

Help

Here you find, on several different tabs, the command-line instruction to execute this step of the lab on your computational infrastructure. Depending on how this course has been setup the instruction will vary. Please see the [Computational Requirements](#) page for information. If you are unsure which instruction to use contact your instructor.

FastQC looks pretty straightforward to run, right? From the help menu, all we need to run this program is to list our sequence files.

PraxisAI

```
fastqc seqfile1 seqfile2 .. seqfileN
```

Stand-alone

```
fastqc seqfile1 seqfile2 .. seqfileN
```

Singularity

```
singularity exec -B ${PWD} docker://systemsgenetics/actg-wgaa:0.1 \
fastqc seqfile1 seqfile2 .. seqfileN
```

Docker

```
docker run -v ${PWD} -u $(id -u ${USER}):$(id -g ${USER}) systemsgenetics/actg-wgaa:0.1 \
fastqc seqfile1 seqfile2 .. seqfileN
```


Help

Here you find, on several different tabs, the command-line instruction to execute this step of the lab on your computational infrastructure. Depending on how this course has been setup the instruction will vary. Please see the [Computational Requirements](#) page for information. If you are unsure which instruction to use contact your instructor.

Give it a shot — run **fastqc** on both of your fastq files.

Step 2: Download the results

PraxisAI is nifty because it also has a way to download data built-in. I marked two arrows here on how to download data from this server to your own local computer.

Download both of the `*fastqc.zip` files to your own computer (right click, download), unzip them and open them up. We'll talk about these together in class.

Mastering Content

Step 1: Conda environments

A good tip with **conda** is to keep your default (base) environment clean, and to create new environments that contain your installed software. You can make as many environments as you'd like. For example, I have one called "pb-assembly" that contains all software related to PacBio genome assembly, annotation, and quality control. I have another environment called "chloroplast" that contains all software I need related to chloroplast genome assembly and annotation.

Your tasks are to:

1. Create a new conda environment called "toomers"
2. Activate the new environment
3. List all of your current environments
4. Switch your environment back to default (base)
5. Switch your environment back to toomers

Step 2: Messy data

The duckweed whole genome shotgun data we investigated with **fastqc** looks really clean, meaning it has high quality scores along the length of both reads, and very little adapter contamination, among other things. What about something a little messier?

Here is the SRA page for small RNA (sRNA) reads from garden asparagus (*Asparagus officinalis*). These are single-end, 50 nt long reads. Small RNAs are typically 18-25 nt pieces of RNA. What happens when the molecule you're sequencing is shorter than the read length of the machine?

[https://www.ncbi.nlm.nih.gov/sra/SRX8241476\[accn\]](https://www.ncbi.nlm.nih.gov/sra/SRX8241476[accn])

Run **fastqc** on this *Asparagus officinalis* sRNA data and see for yourself, then let's talk about this in class together. Give [this guide](#) on fastqc output a read-through.

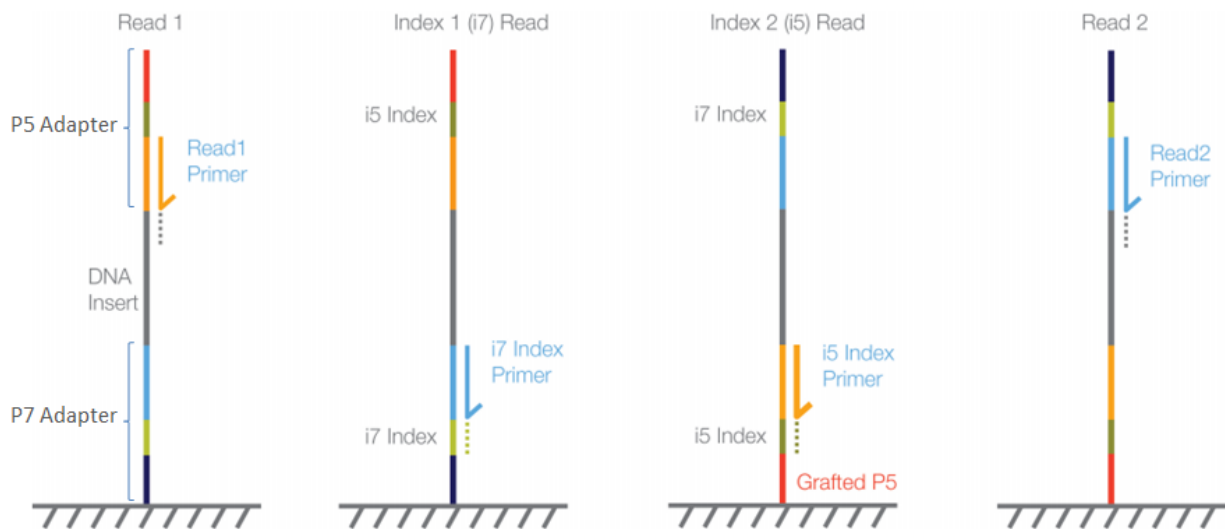


Fig. 3: Image Source: Illumina Website

Step 3: Compression

Right now we have lots of .fastq files sitting around, taking up space. Use the **gzip** compression algorithm to compress all of them.

```
ls *.fastq
gzip *.fastq
```

The asterisk * is a wildcard. See how it works by using `ls *.fastq`. It lists every file that ends in .fastq. Nifty! Unix is all about being lazy (other people call this “efficiency”).

1.3.3 Lesson 3: Compute Clusters and Programming Languages

1.3 Instructions

1.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- Ten simple rules for biologists learning to program
- An Introduction to Computing Clusters
- For bioinformatics, which language should I learn first

1.3 Lesson

Learning Objectives

Vocabulary

Learning Material

1.3 Lab Exercises

Overview

In this lab, we will learn how to organize our project, then write and run a loop in bash.

We will do four major things in this lab:

- Organize our data and the project
- Make symbolic links to raw data
- Write a bash loop
- Run **fastqc** in a bash loop

Give it a go, be patient, and ask questions.

“Some people feel the rain. Others just get wet.” Bob Dylan

Task A: Organize your directories and clean up

Step 1. Project organization

First, let’s talk a little about data management and organization. Do you have a lot of files in your home directory? Is it cluttered? Clean up! Delete things you don’t need anymore (using the `rm` command, carefully. Remember: Once it’s gone, it’s gone for good).

Today we’re going to start our first analysis on the Toomer’s Oak data by touching the raw Illumina data, so let’s go over some best practices in project management. Keep everything organized! Create a new directory called “toomers-genome” in your home directory, as well as four directories within, to represent the 4 major data types we’ll be generating.

```
# This quickly gets you to your home directory
cd ~

mkdir toomers-genome
cd toomers-genome

# We can make multiple directories at once with the mkdir
mkdir shotgun-dna rna-seq pacbio hi-c
```

Note: The pound sign / hashtag is the universal symbol for leaving a comment in a piece of code. Interpreted languages (like bash, perl, python) ignore any line that starts with #. Annotate your scripts, and leave yourself notes, using this symbol! treat your code like your lab notebook.

Change directory into `shotgun-dna`. Then make two more directories called `raw-data` and `fastqc`. Your directory structure should look like this:

```
/home/student/toomers-genome/
├── hi-c
├── pacbio
├── rna-seq
├── shotgun-dna
│   ├── fastqc
│   └── raw-data
```

I used the **tree** command to make this directory tree.

If you're struggling to make directories and move around, or if this tree-like structure of directories doesn't make sense, be sure to review modules 1 and 2 of [Linux Survival](#).

Step 2: Create symbolic links to the data

We don't want to mess with the raw data in the shared directory, so we can create a symbolic link (or softlink) to the data. A soft link is a special kind of file that points to another file, much like a shortcut in Windows or a Macintosh alias. After you've made the symbolic link, you can perform an operation on or execute "myfile", just as you could with the source_file. You can use normal file management commands (for example, `cp`, `rm`) on the symbolic link.

To create a symbolic link in Unix, we type:

```
ln -s source_file myfile
```

- `source_file`: what we want to make a link to
- `myfile`: the name of the softlinked file you want to make in your current directory. I would suggest keeping this the same as the `source_file` name.

Our shared class data is held in `/scratch/`. We never want to mess with this data directly — but we can create softlinks to it! There are two `.fastq.gz` files in there right now, representing ~100X coverage of an Illumina whole genome shotgun, paired-end 150 nt sequencing library.

```
cd ~/toomers-genome/shotgun-dna/raw-data
ln -s /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R1.fastq.gz JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R1.fastq.gz
ln -s /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R2.fastq.gz JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R2.fastq.gz
```

Now we're organized, and we've got softlinks to our raw data.

1.3.4 Lesson 4: Writing a Scientific Manuscript

1.4 Instructions

1.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Oak genomics is providing its worth](#)
- [Pecan genome](#)
- [How to write an abstract](#)

1.4 Lesson

Learning Objectives

Vocabulary

Learning Material

1.4 Lab Exercises

Overview

In this lab, we will learn how to write parts of a manuscript as we go along. In cooking, we say “clean as you cook”, and you will not be left with a giant pile of dishes at the end. The same thing applies for genomics. Keep your eye on the prize: hypothesizing, experimenting, performing, and publishing the highest quality science you can, that tells a well-supported story based on the information you have.

There are also a lot of annoying things that we have to deal with: aggregating citations, aggregating results from major analyses.

We will do four major things in this lab:

- Learn to work with Google Docs
- Learn how to add citations using Paperpile
- Create the skeleton of our genome manuscript
- Fill in basic stats about our data using MultiQC

“If everything was perfect, you would never learn and you would never grow.” -Beyoncé

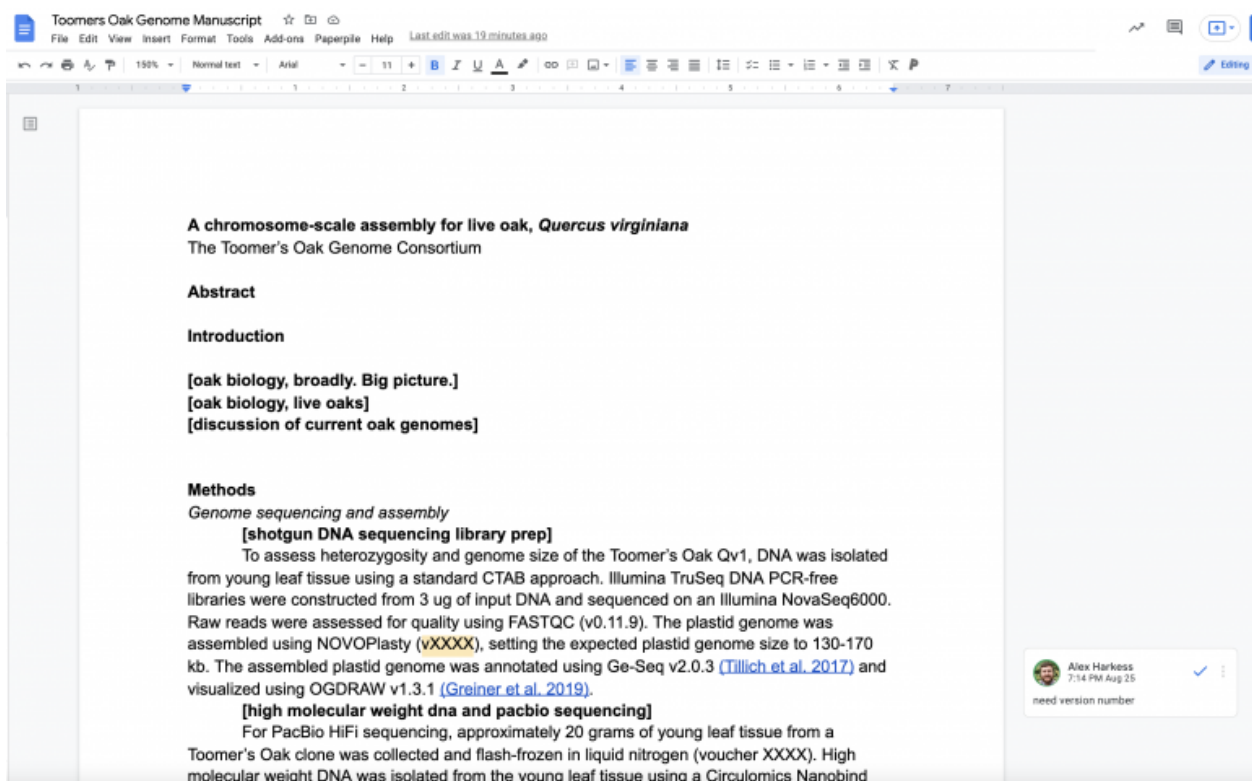
Task A

Step 1: Get comfortable with Google Docs and Paperpile

The end goal of our course is to write a manuscript that details the genome assembly and a variety of analyses for the Toomer's Oak. Writing a manuscript has changed a lot in the last decade. Back then, you used to email a Microsoft Word document back and forth between authors. Thankfully, those days are gone.

There are several ways to collaborate online. Microsoft Teams, Dropbox, Overleaf, and my personal favorite: Google Docs. A key reason I am fan of Google Docs is its ability for multiple people to work on the same document at once, the amount of control you as an author have, and the ability to integrate with third-party plug-ins that make your life easier.

Check with your instructor for the Google Doc link to the manuscript. Below is an example screenshot from a previous class.



Poke around the Google Doc. The two main features we'll use in this class are 1) making a comment (Insert -> Comment) and 2) making a suggestion (View -> Mode -> Suggesting). Make sure you know how to do both.

Step 2: Insert a citation with PaperPile

My #1 advice with writing a scientific manuscript on Google Docs is, just like with coding, don't be a glutton for punishment. There are tools that exist to make your life easier. **USE THEM.** Managing citations is one of those annoyances. Imagine you have a manuscript ready to submit, and you've painstakingly curated all of the 75 citations in the text. You used superscripts (e.g. "Plant Genomes can be quite large³"). Your adviser makes a suggestion and wants you to add an extra citation in the middle of the manuscript. Do you A) scream, or B) re-do all of the citations by hand?

Just like we Conda as a package manager to help us install software, there are citation managers we can use to help us

insert and edit citations using large manuscript databases (e.g. PubMed, Google Scholar), and automatically populate reference lists that meet formatting criteria for different journals. Some common examples are [Mendeley](#), [Zotero](#), and [PaperPile](#). For this course, we'll use Paperpile – it integrates with Google Docs, has an Ipad app, and works on every browser.

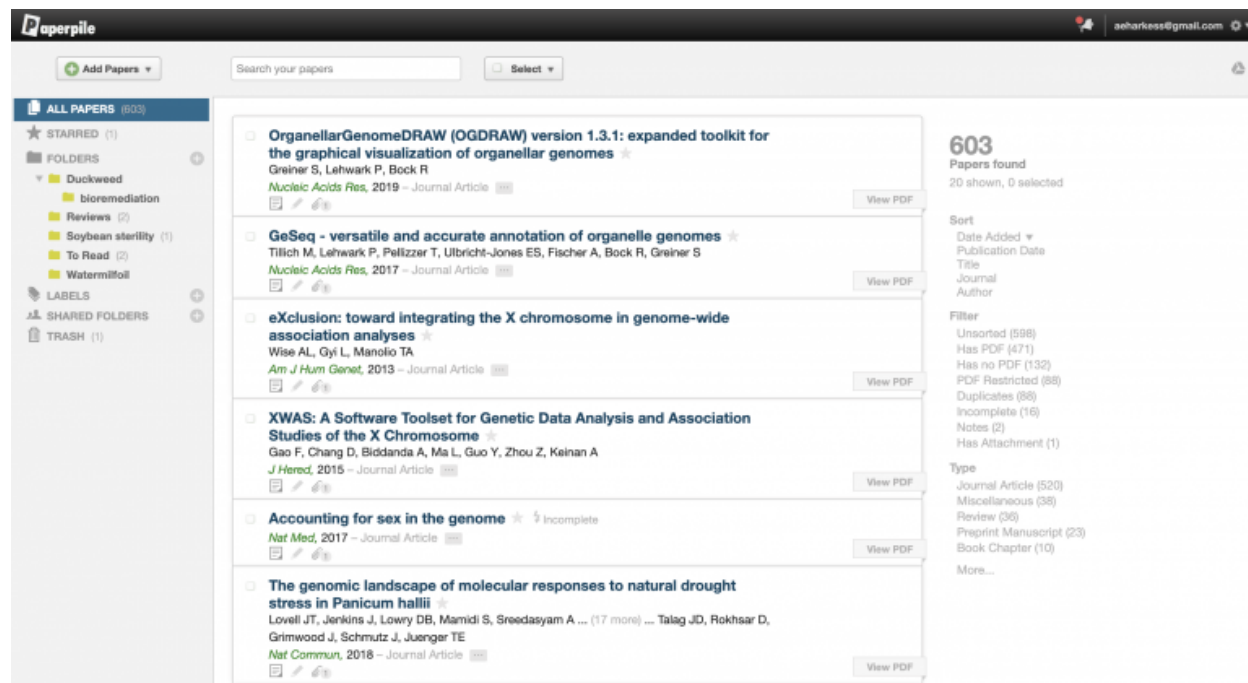


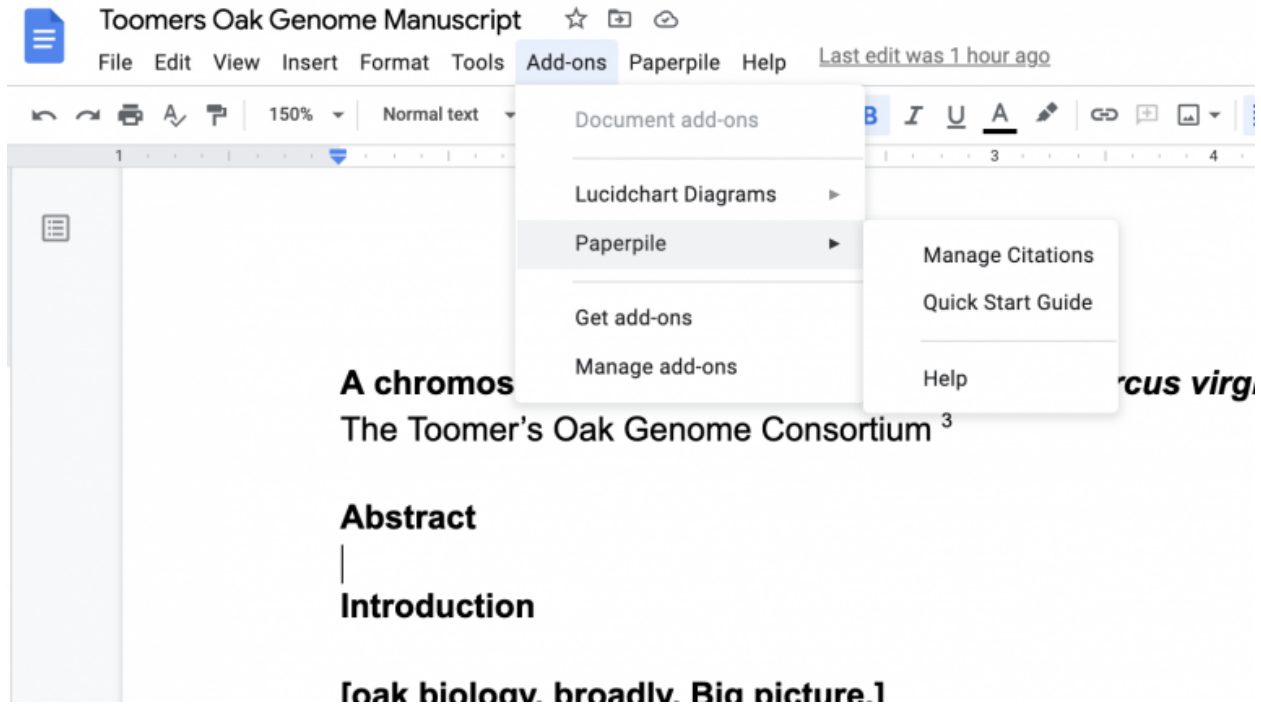
Fig. 4: My PaperPile database with 603 papers in it. I should read more papers!

If you haven't already, go ahead and start your [free trial](#) for PaperPile. Download and install the [Google Doc plugin](#).

Step 3: Insert a citation with PaperPile

PaperPile is powerful and useful, but just like every citation manager, it's not perfect. Citation managers rely on a community of users to curate and fix incorrect citations, especially for older or niche manuscripts that aren't always contained within automatically populated databases.

Inserting citations in PaperPile is quick and easy. There are two ways to do it. First, you can open PaperPile as a sidebar in Google Docs by going to the Add-ons tab and clicking "Manage Citations".



That opens up a panel to the right side where you can search for manuscripts using the title, DOI number or just general search terms. Give it a shot — search for a citation for a paper you recently read, and try to insert it somewhere in the document.

You can also add citations by using the Paperpile menu option. Paperpile >> Insert Citation.

When you're ready to insert a list of formatted citations at the end of the document, we use "Format citations". Whenever we update citations in the text, or add new citations, we can refresh the final citation list at the end of the manuscript by using "Format citations". We can also pick our journal style using "Citation style".

This is all part of "cleaning as we cook". As you write, fill in the citation, and you'll save yourself tons of time.

Task B: Cleaning up after ourselves

Last lab we discovered a few major issues with our data. In particular, a failed transfer of our fastq files truncated the end.

Step 1: Rerun fastqc on the complete Toomers WGS dataset

Return to the previous lab if you have any trouble. Make sure to leave the results in `~/toomers-genome/shotgun-dna/fastqc`

Here are some tricks you can use to shove jobs into the background and keep them running silently. `nohup` is a command that means "no hang up". Output that would normally go to the terminal goes to a file called `nohup.out`, if it has not already been redirected. Add in the ampersand (&) at the end to make sure the job goes into the background.

```
nohup fastqc -t 4 filename1 filename2 &
```

Remember how we can run **fastq** multi-threaded?

You can always check on the status of a job in the background by typing **top**. It is a task manager program, found in many Unix-like operating systems, that displays information about CPU and memory utilization. Read more about [top](#) and how to interpret the CPU and memory usage of your jobs.

```
top
```

Press “q” to get out.

Mastering Content

MultiQC is a software tool that aggregates the results of many common bioinformatic analyses. As always, our goal is to let computational tools do as much work as possible for us, especially for the annoying things: for example, how many reads did we sequence, how many reads are clean, what is the alignment rate of every RNA-seq library we sequenced, etc etc etc.

From the website: MultiQC is a reporting tool that parses summary statistics from results and log files generated by other bioinformatics tools. MultiQC doesn't run other tools for you – it's designed to be placed at the end of analysis pipelines or to be run manually when you've finished running your tools.

When you launch MultiQC, it recursively searches through any provided file paths and finds files that it recognises. It parses relevant information from these and generates a single stand-alone HTML report file. It also saves a directory of data files with all parsed data for further downstream use.

Read the manual on how to install and run it on ~/toomers-genome. We will run MultiQC throughout the semester as we run more programs to update major analyses.

Take note of a particularly important bit here — MultiQC uses python 3.7. MultiQC suggests that you make a new Conda environment that runs python 3.7. Nifty!

```
conda create --name py3.7 python=3.7 conda activate py3.7
conda active py3.7
conda install -c bioconda -c conda-forge multiqc
```

1.4 Module 2: Planning a Genome Project

1.4.1 Lesson 1: Isolating DNA and RNA

2.1 Instructions

2.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [A simple plant high-molecular-weight DNA extraction method](#)
- [Spectrophotometry for DNA purity](#)

2.1 Lesson

Learning Objectives

Vocabulary

Learning Material

Error: EMBED LECTURE VIDEO HERE

Error: ADD TEXT TRANSCRIPTION OF VIDEO HERE

2.1 Lab Exercises

Overview

In this lab, we will learn how to clean your raw Illumina data of adapters and poor quality sequence.

We will do four major things in this lab:

- Learn what Illumina adapter sequences are
- Learn how Illumina quality scores work on the NovaSeq6000
- Run fastp to clean your data
- Calculate sequencing coverage for a sample

“If everything was perfect, you would never learn and you would never grow.” -Beyoncé

Task A

Step 1: Learn the structure of an Illumina sequence run

Illumina sequencing is based on the chemistry of SBS – “sequencing-by-synthesis”. In the next lecture you’ll learn more about the intricacies of SBS and how the molecules sit on the machine’s flow cell. In short, SBS chemistry uses four fluorescently labeled nucleotides to sequence up to billions of clusters on the flow cell surface in parallel, much like Sanger Sequencing. During each sequencing cycle, a single labeled deoxynucleoside triphosphate (dNTP) is added to the nucleic acid chain, one base at a time. A’s are added, then imaged, T’s are added, then imaged, C’s are added, then imaged, then G’s are added. and imaged. After all 4 bases have been added, all molecules on the flow cell should be advanced by one nucleotide of length, or 1 sequencing cycle. The dNTPs contain a reversible blocking group that serves as a terminator for polymerization, so after each dNTP incorporation, the fluorescent dye is imaged to identify the base and then enzymatically cleaved to allow incorporation of the next nucleotide. Since all four reversible terminator-bound dNTPs (A, C, T, G) are present as single, separate molecules, natural competition minimizes incorporation bias, which can be problematic with serial nucleotide incorporation chemistry used in Sanger sequencing. Base calls are made directly from signal intensity measurements during each cycle, greatly reducing raw error rates compared to other technologies.

Check out this short, 5 minute video for a quick primer on Illumina sequencing.

For more than a decade, 4 different dye terminator colors were used (T = green, G = blue, C = Red, A = yellow), just like Sanger sequencing. This is called 4-color sequencing. Illumina realized they could speed up sequencing by

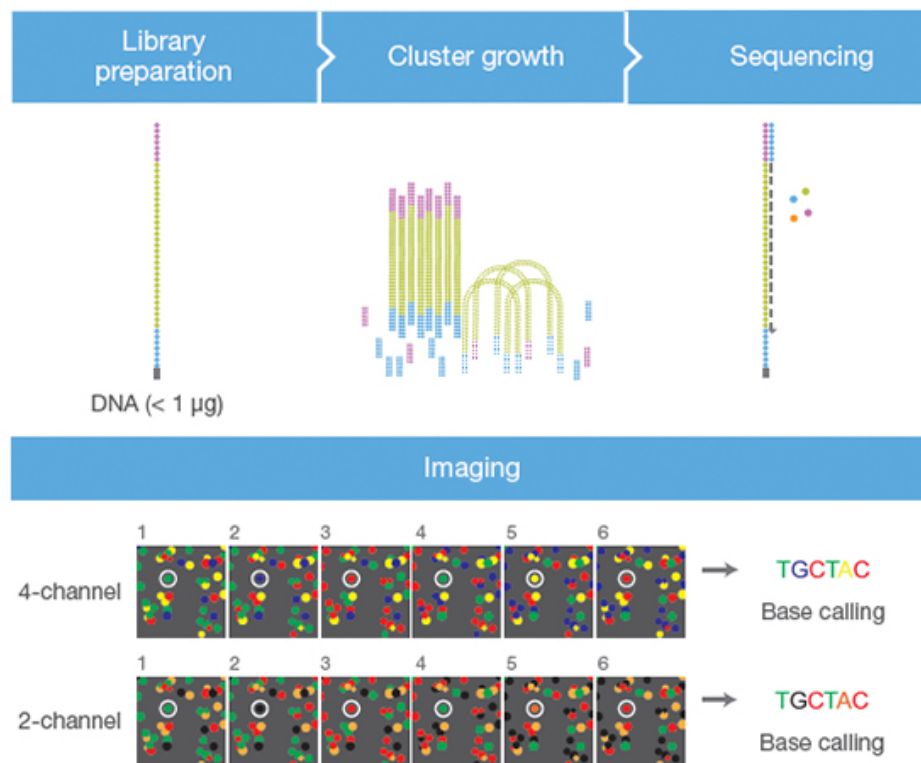


Fig. 5: Image Source: Illumina Website

eliminating two of the colors altogether, reducing the sequencing time by 50%. This is called two-color sequencing. There are only two colored fluorophores used now:

If a base lights up green, it's a T. If a base lights up red, it's a C. If a base lights up yellow, it's A — meaning they mixed red and green fluorophores together for the “A” cycle. If it's G, there's no fluorophore, meaning it's a dark cycle and no light was emitted. The axis of the figure below show the intensity of each base, measured in green versus red intensity.

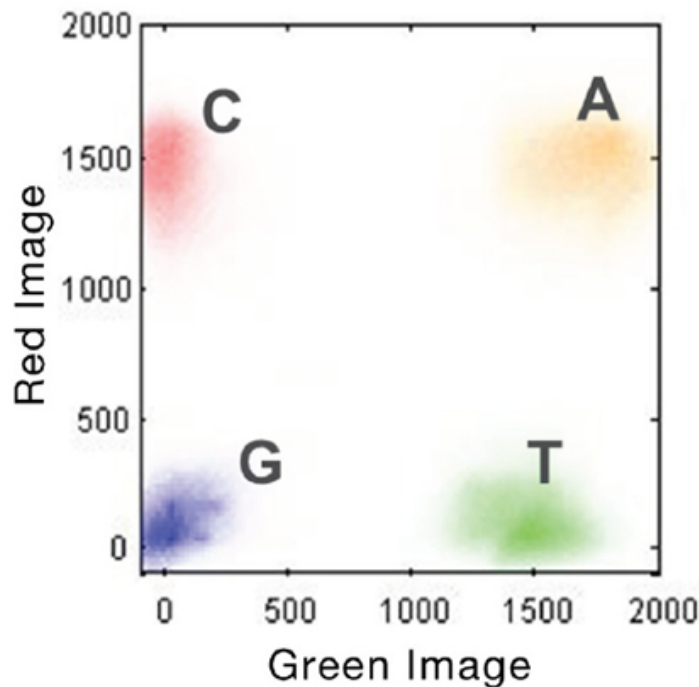


Fig. 6: Image Source [Illumina Website](#)

Fig. 7: Image Source [EC Seq Bioinformatics Website](#)

Do you have two colors or four colors in Illumina?

Step 2: Learn how quality scores in Illumina work

No sequencing technology is perfect (yet), and errors occur at some frequency. It's important to understand the source of these kinds of errors, because 1) they impact the downstream analyses you want to do, and 2) so you can account for them. Illumina has the lowest per-base error rate of any modern sequencing platform, which is why it is often employed for variant calling during resequencing.

The reasons for errors during the base calling process are diverse. In most cases the emitted light signal of a cluster is disturbed. In this context, it is important to know that the detected light signal is always a sum of single signals from thousands of molecules within one cluster. Typical reasons for a polluted cluster light signal can be phasing (see [quality decrease over illumina reads](#)), overlapping clusters and not uniform clusters because of an error in the cluster generation (bridge amplification) step. This can often lead to a gradual degradation of signal over the course of sequencing a molecule. Because the probability of errors fluctuates and differs from cluster to cluster and from cycle to cycle it is necessary and useful to indicate a quality for each called and recorded base expressed in a score.

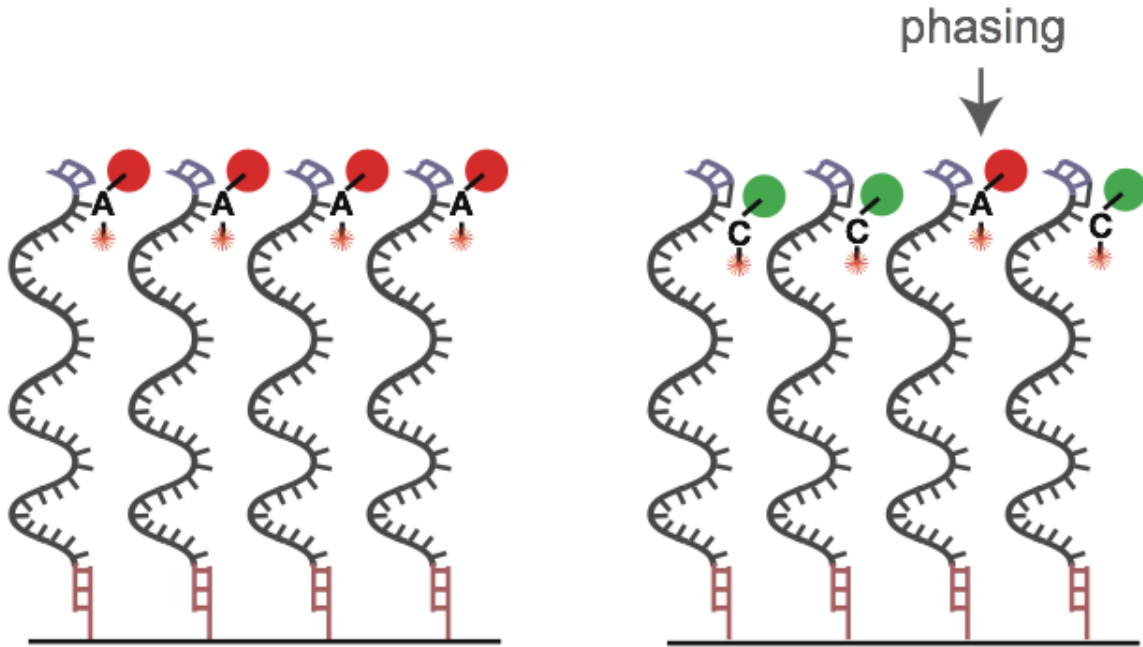


Fig. 8: On the left, a perfect sequencing run with many molecules acting the same within a single cluster. On the right, an example of “phasing”, where the dNTP blocker nucleotide isn’t cleaved, polluting the light signal in the cluster. Image Source [EC Seq Bioinformatics Website](#)

In the last few years, especially with the introduction of Illumina’s bigger machines like the NovaSeq6000, the data produced by the machine is becoming unwieldy. One source of data bloating was that every single nucleotide of every single read had a quality score attached to it. The quality score (Q score) with the attached probability of error (P_error) was given a different ASCII code:

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	33 !	11	0.07943	44 ,	22	0.00631	55 7	33	0.00050	66 B
1	0.79433	34 "	12	0.06310	45 -	23	0.00501	56 8	34	0.00040	67 C
2	0.63096	35 #	13	0.05012	46 .	24	0.00398	57 9	35	0.00032	68 D
3	0.50119	36 \$	14	0.03981	47 /	25	0.00316	58 :	36	0.00025	69 E
4	0.39811	37 %	15	0.03162	48 0	26	0.00251	59 ;	37	0.00020	70 F
5	0.31623	38 &	16	0.02512	49 1	27	0.00200	60 <	38	0.00016	71 G
6	0.25119	39 '	17	0.01995	50 2	28	0.00158	61 =	39	0.00013	72 H
7	0.19953	40 (18	0.01585	51 3	29	0.00126	62 >	40	0.00010	73 I
8	0.15849	41)	19	0.01259	52 4	30	0.00100	63 ?	41	0.00008	74 J
9	0.12589	42 *	20	0.01000	53 5	31	0.00079	64 @	42	0.00006	75 K
10	0.10000	43 +	21	0.00794	54 6	32	0.00063	65 A			

Fig. 9: Phred33 offset ASCII table.

Illumina’s solution to this issue is to not report the quality score of every single nucleotide. Instead, quality scores are **binned** into 4 or 8 categories of qualities. See the table below for an example of how quality scores can be binned into 8 Q-score categories. In the example below, if a base has a quality score of 7, it gets changed to a “6”. This saves space because when compression algorithms or formats (like gzip, bzip, .bam) compress data, repeated stretches of the same byte (like a quality score bin value) can be very efficiently compressed, resulting in smaller file sizes. In practice, it hasn’t really mattered much to us; binning actually saves us computational time when we trim the data and very few people *really* needed to know the Q score of every single base.

Q-Score Bins	Example of Empirically Mapped Q-Scores*
N (no call)	N (no call)
2–9	6
10–19	15
20–24	22
25–29	27
30–34	33
35–39	37
≥ 40	40

By replacing the quality scores between 19 and 25 with a new score of 22, data storage space can be conserved.

*The mapped quality score for each bin (except “N”) is subject to change depending on individual Q-tables.

Task B

Many programs have been written to “clean” Illumina sequencing data. Some common examples are [trimmomatic](#), [TrimGalore!](#), and my personal favorite, [Fastp](#). Fastp is exceptionally fast, and the defaults are excellent. It will automatically trim your fastq dataset with reasonable defaults, including automatically identifying and trimming the adapter sequences that might be present in your data.

The structure of an Illumina-ready molecule for sequencing is below. The “insert” is your biological sequence (e.g. a piece of DNA or RNA), flanked on both sides by “adapter” sequence that is required for binding to the flow cell. The first read (R1) initiates sequencing at Primer 1, and reads through the insert sequence on the top strand. Depending on the length of the insert, and the chosen sequencing read length, sometimes you can sequencing into the adapter on the other side of the molecule (shown by the red dots). These adapter sequences at the 3 ends of reads need to be removed. They are not true biological sequence!

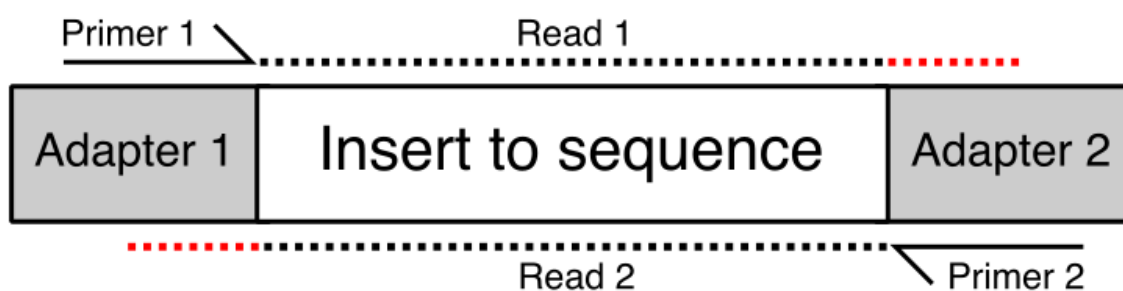


Fig. 10: Image Source: [QCFail.com Website](#)

QC Fail Sequencing » Read-through adapters can appear at the ends of sequencing reads These issues are visible in the fastqc plots, like the example below, which shows the Illumina Universal Adapter being present in a high frequency of molecules starting ~35 nucleotides. The insert must be very short here:

Similarly, remember how chemistry issues like “phasing” can lead to signal degradation over time? Quality scores often start to drop as the sequencing moves towards the 3 end of each molecule. This is normal, and we can detect this in fastqc plots. Below is an example:

- Why does the per base sequence quality decrease over the read in Illumina?
- Does the Toomer’s Oak data display this trend? Let’s look at the fastqc plots you made.

Install fastp

Read the github page, and install fastp using Conda. Make sure you’re in your “toomers” environment (or whatever you decided to name it).

There are many ways to run fastp to output cleaned reads. You can 1) stream the reads directly to the standard out, so that you can pipe them into another program (e.g. a read aligner like BWA or bowtie), 2) write the cleaned reads to a separate file (which takes up space), or 3) output nothing except for the cleaned read statistics. For this lab, we’ll do #3. In the very near future future, since fastp is so quick to run, we will use the “streaming” option #1, and make use of pipes. This saves us a lot of storage — do we need to keep a copy of the raw data, plus a copy of the cleaned data? Not really*

Note: There are caveats here we will talk about in lab.

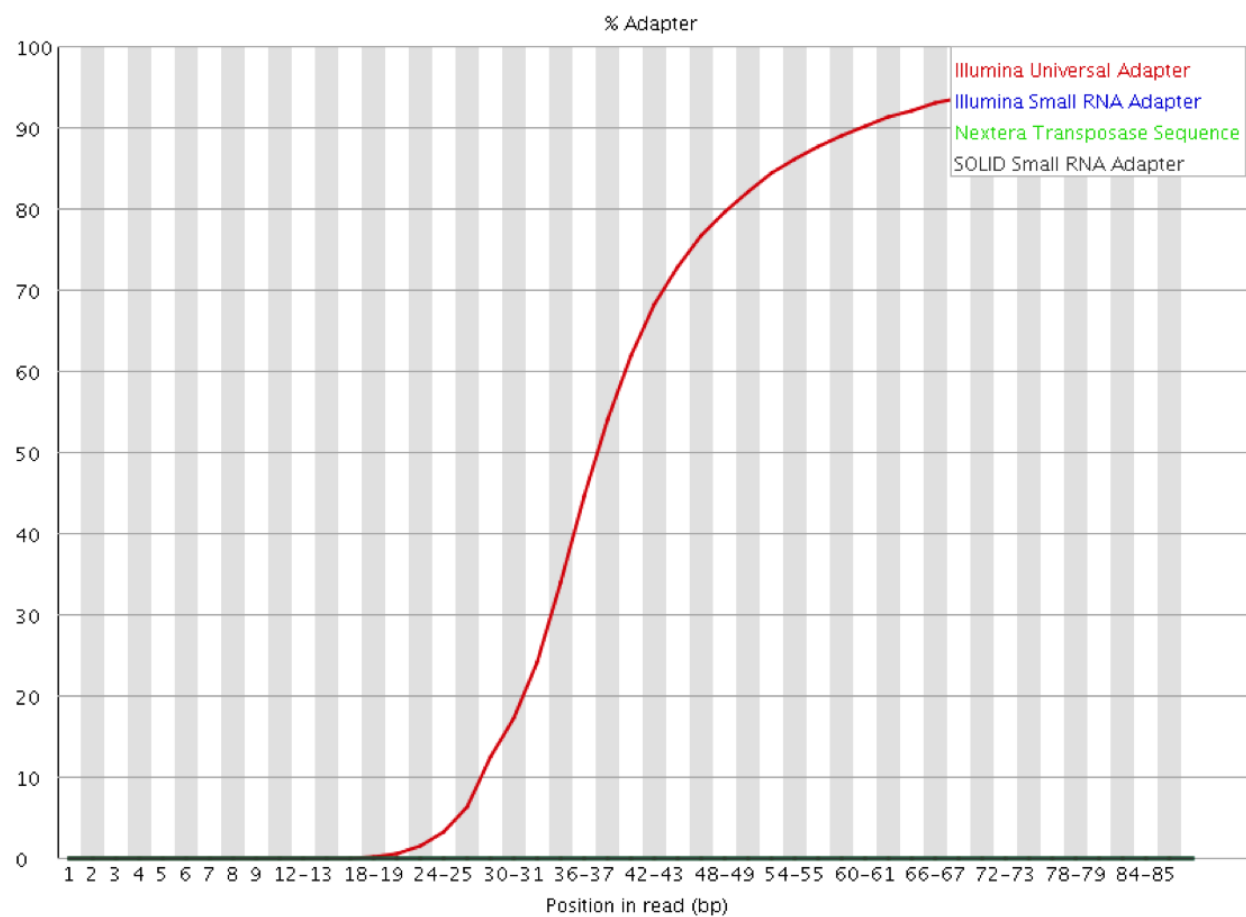


Fig. 11: Image Source: QCFail.com Website

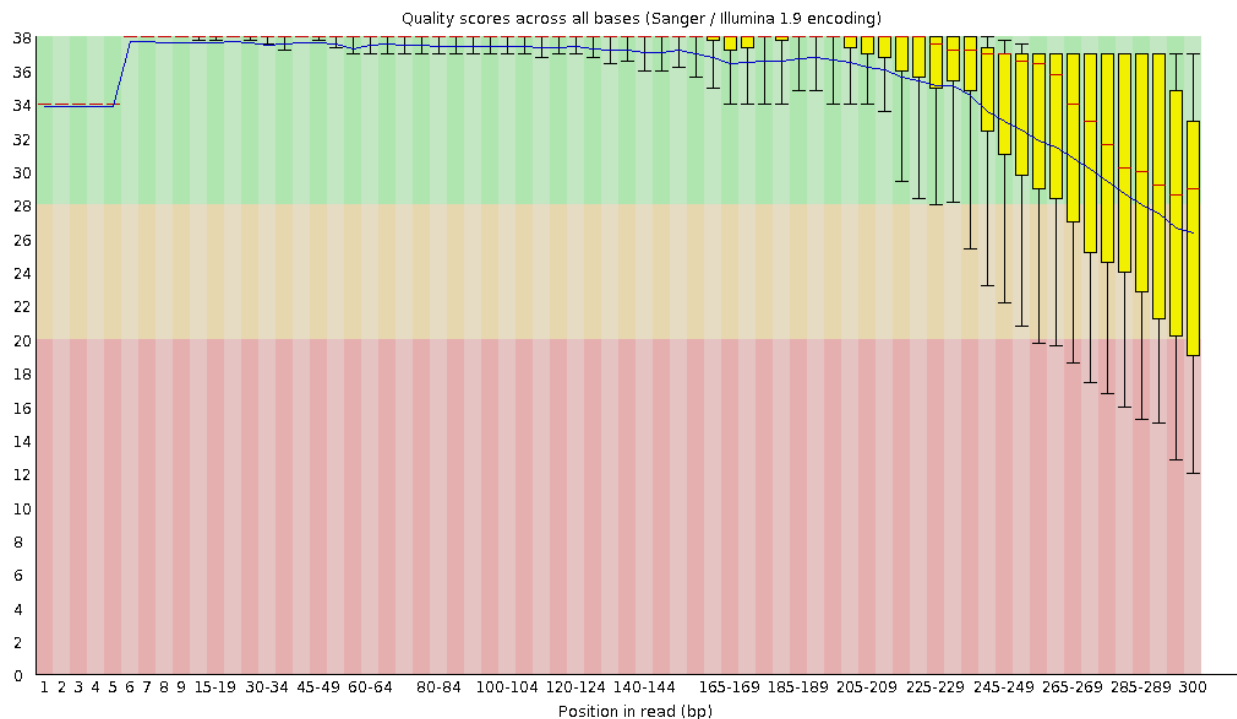


Fig. 12: Image Source [EC Seq Bioinformatics Website](#)

Use the same raw Illumina whole genome shotgun data that you used in the last lab. read1 is the file that ends in R1, and read2 is the file that ends in R2. Insert the correct path to the reads, either the raw data from /scratch or your softlinked files in your own directory. The simplest way to run fastp to only generate a quality report of our data is:

```
fastp -i read1 -I read2
# yeah, it's that easy.
```

Use the ampersand (&) to start this job in the background. Ask google or your classmates if you can't remember how. This job will take a few hours.

After the run has finished, run MultiQC in the ~/toomers-genome/ directory to aggregate your **fastqc** and **fastp** results.

Mastering Content

What depth of coverage did I sequence to?

A question we often ask — “Did I sequence deeply enough?”.

Next-generation shotgun sequencing approaches require sequencing every base in a sample several times for two reasons:

- You need multiple observations per base to come to a reliable base call.
- Reads are not distributed evenly over an entire genome, simply because the reads will sample the genome in a random and independent manner. Therefore many bases will be covered by fewer reads than the average coverage, while other bases will be covered by more reads than average. You need to account for this in your planning.

This is expressed by the coverage metric, which is the number of times a genome has been sequenced (the depth of

sequencing). For applications where you aim to sequence only a defined subset of an entire genome, like targeted resequencing or RNA sequencing, coverage means the amount of times you sequence that subset. For example, for targeted resequencing, coverage means the number of times the targeted subset of the genome is sequenced. In this case, we want to know the sequencing coverage of the whole genome; in other words, how many times did we sequence each nucleotide of the oak tree, on average?

The general equation for computing coverage is: $C = LN / G$ - C stands for coverage - G is the haploid genome length - L is the read length - N is the number of reads

Assume that the diploid genome size of our Toomers Oak is 1.5 Gigabases. What coverage of Illumina read depth did we sequence to?

1.4.2 Lesson 2: Data Types in Genomics

2.2 Instructions

2.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Illumina Sequencing Technology](#)
- [PAC Bio Sequencing Primer](#)

2.2 Lesson

Learning Objectives

Vocabulary

Learning Material

2.2 Lab Exercises

Overview

When you work on a genome project, you'll be generating several kinds of raw data. In this lab, we will explore three diverse data types in genomics: fasta, fastq, bam, and gff. There are many more, but we'll focus on these for now.

We will do four major things in this lab:

- Dissect the headers of an Illumina fastq file (.fastq format)
- Align short reads to your Toomer's chloroplast genome (.fasta + .bam)
- Explore the annotation (.gff) and alignments (.bam) in IGV

"If everything was perfect, you would never learn and you would never grow." -Beyoncé

Task A

Step 1: Dissect a fastq file

By now, you’ve interacted with fastq files in the class in different ways. Today let’s really dive into what a fastq file tell us.

A FASTQ file normally uses four lines per sequence.

- Line 1 begins with a ‘@’ character and is followed by a sequence identifier and an optional description (like a FASTA title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a ‘+’ character and is optionally followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

Here’s a random read I chose from our fastq file:

```
@A00406:205:HGTFWDSX2:4:1101:14470:1157 1:N:0:GGTACCTT+GACGTCTT
ATTTAATACTTCGTTACGTTAGATCTTAGTTACTGACTGGCGTTAAATAAGCTAAATGATATTGTTTGGTTTCTTTTTTAATTTTGAAATTAAGAAA
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
↪FFF,:F:FFFF
```

Let’s dissect the fastq entry, starting with the first line (a.k.a. the header):

@A00406	The unique instrument ID
205	The run ID
HGTFWDSX2	The flowcell ID
4	The flowcell lane (4 lanes / NovaSeq flowcell)
1101	Tile number within the flowcell lane
14470	X-coordinate of the cluster within the tile
1157	Y-coordinate of the cluster within the tile
1	For paired-end reads, is this read 1 or read 2
N	Y if the read was filtered by the machine for poor quality, N if the read passed. see below
0	You can spike in PhiX to a sequence run as a control sequence. If this value ever reads “18”, this means the read is matching to a control. Otherwise, 0.
GGTAC-CTT+GACGTCTT	This is the barcode for this library. In our case, we performed dual indexing, where each molecule has a unique i5 and i7 index sequence on both ends. The i5 and i7 8-nucleotide index sequences are separated with a plus sign. i7 is the 3 adapter, and i5 is the 5 adapter. i7 is read first, then i5.

Illumina sequencers perform an internal quality filtering procedure called **chastity filter**, and reads that pass this filter are called **PF** for **pass-filter**. According to Illumina, **chastity** is defined as the ratio of the brightest base intensity divided by the sum of the brightest and second brightest base intensities. Clusters of reads pass the filter if no more than 1 base call has a chastity value below 0.6 in the first 25 cycles. This filtration process removes the least reliable clusters from the image analysis results. (Source: [GATK](#))

Note: What to look out for: When you get new data, always use

```
zcat <fastq.gz> | head
```

and check out the first few lines of the fastq file. Check the header for the correct barcode for a few samples. Sequencing centers aren't perfect, "stuff" happens, and just smart to spot check your data.

Discussion question: What do you notice about the barcodes in the first few entries of our fastq files?

Task B: Align reads to your chloroplast genome

Let's run a very common bioinformatic task, and learn more about fasta, fastq, and bam files along the way. We are going to align reads to a reference genome. In this case, the reads will be the PE150 WGS reads we've been working with, and the reference will be the Toomer's Oak chloroplast fasta you assembled and annotated. Alignment is sometimes called "mapping", e.g. "aligning reads to a genome" is the same as "mapping reads to a genome". Here's a broad overview of what we'll be doing:

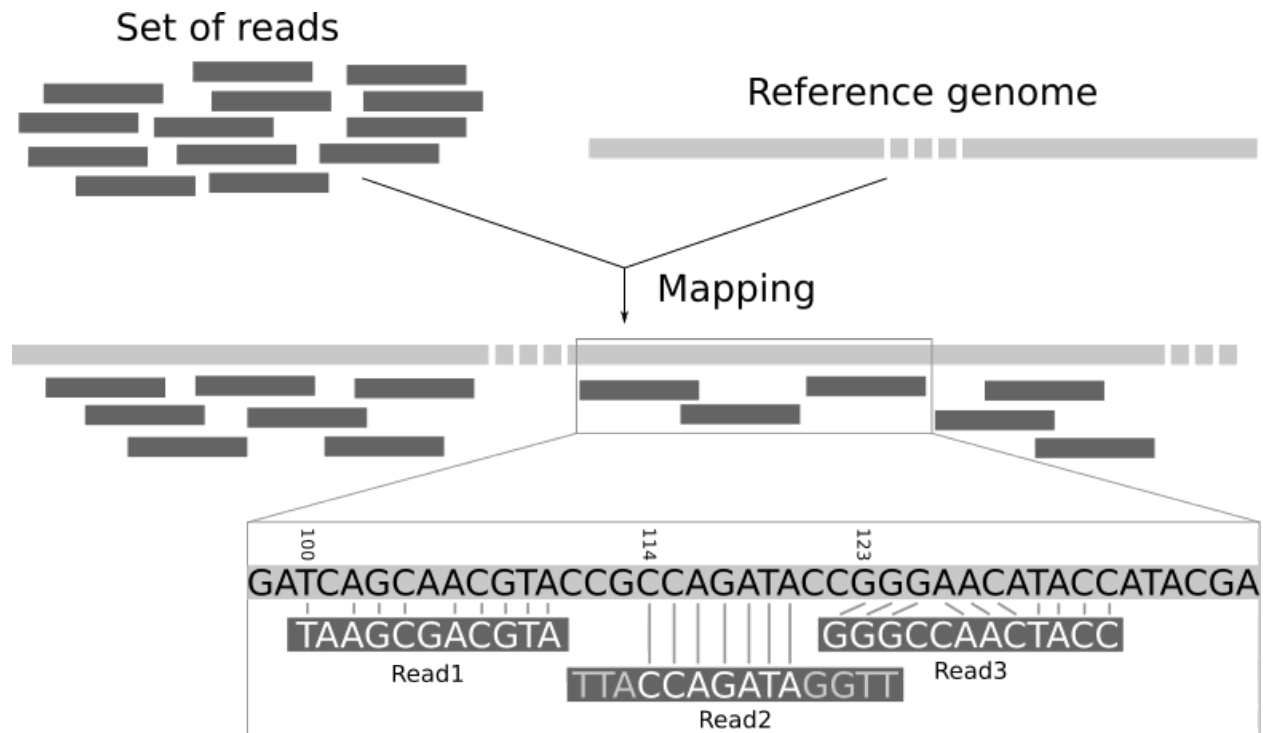


Fig. 13: Image Source: [Galaxy Project Website](#)

Fastp took our two paired-end read files as input, cleaned them, and it can stream the output (cleaned reads) to another program. How do we deal with the fact that 2 read files went into fastp, but only one stream can exit as output? There is a flavor of fastq files called "interleaved", where Read1 and Read2 are zippered together, like this:

Read1:

```
>SLXA-EAS1_89:1:1:672:654/1
GCTACGGAATAAAACCAGGAACAACAGACCCAGCA
>SLXA-EAS1_89:1:1:657:649/1
GCAGAAAATGGGAGTGAAAATCTCCGATGAGCAGC
>SLXA-EAS1_89:1:1:708:653/1
GAGAGAGCAGTGGGCGAGGTTGGGACATGTCATGA
```

Read2:

```
>SLXA-EAS1_89:1:1:672:654/2
ATTAACAACAAAGGGTAAAAGGCATCATGGCTTCA
>SLXA-EAS1_89:1:1:657:649/2
TGATGCGACGACGCACCTCGTTGTTACGCACTTCA
>SLXA-EAS1_89:1:1:708:653/2
CTGTGGATAACATGGTGTAAAGATCCTGTTTATTTT
```

Interleaved:

```
>SLXA-EAS1_89:1:1:672:654/1
GCTACGGAATAAAACCAGGAACAACAGACCCAGCA
>SLXA-EAS1_89:1:1:672:654/2
ATTAACAACAAAGGGTAAAAGGCATCATGGCTTCA
>SLXA-EAS1_89:1:1:657:649/1
GCAGAAAATGGGAGTGAAAATCTCCGATGAGCAGC
>SLXA-EAS1_89:1:1:657:649/2
TGATGCGACGACGCACCTCGTTGTTACGCACTTCA
```

BWA, the short read aligner we will use next, can recognize interleaved files.

Step 1: Install BWA

BWA is the Burrow-Wheeler Alignment (BWA) program. This is a powerful and fast aligner that works with both short read (Illumina) and long-read (PacBio, Nanopore) data. Check out the [Github page](#).

First, Install bwa and samtools using Conda. We'll use BWA to align reads, then samtools to filter those reads and produce a .bam file that records all of the read alignments and their locations on the reference.

Second, build a bwa index from your Option_1_toomers-cp.fasta assembly (or whatever you named it). BWA requires building an index for your reference genome to allow it to more efficiently search the genome during sequence alignment:

```
bwa index Option_1_toomers-cp.fasta
```

Third, check out all the [options for bwa mem](#), the aligner we'll use. I've highlighted an important bit about interleaved files:

```
mem      bwa mem [-aCHMP] [-t nThreads] [-k minSeedLen] [-w bandWidth] [-d zDropoff] [-r
seedSplitRatio] [-c maxOcc] [-A matchScore] [-B mmPenalty] [-O gapOpenPen] [-E
gapExtPen] [-L clipPen] [-U unpairPen] [-R RGLine] [-v verboseLevel] db.prefix
reads.fq [mates.fq]
```

Align 70bp-1Mbp query sequences with the BWA-MEM algorithm. Briefly, the algorithm works by seeding alignments with maximal exact matches (MEMs) and then extending seeds with the affine-gap Smith-Waterman algorithm (SW).

If *mates.fq* file is absent and option *-p* is not set, this command regards input reads are single-end. If *mates.fq* is present, this command assumes the *i*-th read in *reads.fq* and the *i*-th read in *mates.fq* constitute a read pair. If *-p* is used, the command assumes the *2i*-th and the *(2i+1)*-th read in *reads.fq* constitute a read pair (such input file is said to be interleaved). In this case, *mates.fq* is ignored. In the paired-end mode, the mem command will infer the read orientation and the insert size distribution from a batch of reads.

The BWA-MEM algorithm performs local alignment. It may produce multiple primary alignments for different part of a query sequence. This is a crucial feature for long sequences. However, some tools such as Picard's markDuplicates does not work with split alignments. One may consider to use option *-M* to flag shorter split hits as secondary.

OPTIONS:

- t INT* Number of threads [1]
- k INT* Minimum seed length. Matches shorter than *INT* will be missed. The alignment speed is usually insensitive to this value unless it significantly deviates 20. [19]
- w INT* Band width. Essentially, gaps longer than *INT* will not be found. Note that the maximum gap length is also affected by the scoring matrix and the hit length, not solely determined by this option. [100]
- d INT* Off-diagonal X-dropoff (Z-dropoff). Stop extension when the difference between the best and the current extension score is above $|i-j|*A+INT$, where *i* and *j* are the current positions of the query and reference, respectively, and *A* is the matching score. Z-dropoff is similar to BLAST's X-dropoff except that it doesn't penalize gaps in one of the sequences in the alignment. Z-dropoff not only avoids unnecessary extension, but also reduces poor alignments inside a long good alignment. [100]
- r FLOAT* Trigger re-seeding for a MEM longer than *minSeedLen*FLOAT*. This is a key heuristic parameter for tuning the performance. Larger value yields fewer seeds, which leads to faster alignment speed but lower accuracy. [1.5]
- c INT* Discard a MEM if it has more than *INT* occurrence in the genome. This is an insensitive parameter. [10000]
- P* In the paired-end mode, perform SW to rescue missing hits only but do not try to find hits that fit a proper pair.
- A INT* Matching score. [1]

Next, we want to set up a series of pipes to stream our data from fastp >> bwa >> .bam alignment file. I noticed something funny, though — when I checked our version of fastp, it says 0.12.4. But the Github version says it's on 0.22.0 ! We really need the *-stdout* option, since that's how we'll stream the cleaned reads into BWA for alignment. But the older version of fastp doesn't have that option! I wonder if there's some incompatibilities we didn't know about, since we're all using a new VM with perhaps different underlying software than we're used to...

Here's how we can install a specific version of something on Conda

```
conda install -c bioconda fastp=0.22.0
```

```
(toomers) student@landingvm:~/toomers-genome/shotgun-dna/chloroplast$ conda install -c bioconda fastp=0.22.0
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: -
Found conflicts! Looking for incompatible packages.
This can take several minutes. Press CTRL-C to abort.
failed

UnsatisfiableError: The following specifications were found to be incompatible with each other:

Output in format: Requested package -> Available versions

Package libgcc-ng conflicts for:
fastp=0.22.0 -> libgcc-ng[version='>=9.4.0']
fastp=0.22.0 -> zlib[version='>=1.2.11,<1.3.0a0'] -> libgcc-ng[version='>=7.2.0|>=7.3.0']

Package zlib conflicts for:
fastp=0.22.0 -> zlib[version='>=1.2.11,<1.3.0a0']
python=2.7 -> zlib[version='>=1.2.11,<1.3.0a0']The following specifications were found to be incompatible with your system:

- feature:/linux-64::__glibc==2.31=0
- feature:|@/linux-64::__glibc==2.31=0
- python=2.7 -> libgcc-ng[version='>=7.3.0'] -> __glibc[version='>=2.17']

Your installed version is: 2.31
```

Failed! Okay. Let's problem solve. We *could* try and install updated versions of **libgcc** and **zlib**, but these are more complex compilers that many programs rely on. Is there another way to run the latest version of fastp without having to install anything?

Remember the difference between interpreted code and compiled code? Sometimes developers will provide the **pre-compiled binaries**. The fastp developers do! We can download and run precompiled binaries without installing anything. Problem solved.

Before you download the binaries, let's create a new folder called "bin" inside of toomers-genome, where we will keep any scripts and programs we accumulate throughout the course. Follow their instructions and download fastp binaries into your toomers-genome/bin/ directory. Here's what my entire directory organization looks like now:

```

├── bin
│   └── fastp
├── hi-c
├── pacbio
├── rna-seq
└── shotgun-dna
    ├── chloroplast
    │   ├── assemblathon_stats.pl
    │   ├── Contigs_1_toomers-cp.fasta
    │   ├── contigs_tmp_toomers-cp.txt
    │   ├── log_toomers-cp.txt
    │   ├── Merged_contigs_toomers-cp.txt
    │   ├── nohup.out
    │   ├── novoplasty.cfg
    │   ├── Option_1_toomers-cp.fasta
    │   ├── Option_1_toomers-cp.fasta.amb
    │   ├── Option_1_toomers-cp.fasta.ann
    │   ├── Option_1_toomers-cp.fasta.bwt
    │   ├── Option_1_toomers-cp.fasta.pac
    │   ├── Option_1_toomers-cp.fasta.sa
    │   ├── Option_2_toomers-cp.fasta
    │   └── rbcl.fa
    ├── fastp
    │   ├── fastp.html
    │   └── fastp.json
    ├── fastqc
    └── raw-data
        ├── JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R1.fastq.gz -> /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L
        ├── 4_R1.fastq.gz
        │   └── JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R2.fastq.gz -> /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L
        └── 4_R2.fastq.gz
```


Step 2: String together a set of pipes

Let's test `fastp` and see if it can stream the output to the **stdout** (standard out) so that we can pipe it into BWA for alignment. [The github page](#) told me everything I needed to know. Here's how I usually test things like this:

output to STDOUT

`fastp` supports streaming the passing-filter reads to STDOUT, so that it can be passed to other compressors like `bzip2`, or be passed to aligners like `bwa` and `bowtie2`.

- specify `--stdout` to enable this mode to stream output to STDOUT
- for PE data, the output will be interleaved FASTQ, which means the output will contain records like `record1-R1 -> record1-R2 -> record2-R1 -> record2-R2 -> record3-R1 -> record3-R2 ...`

```
~/toomers-genome/bin/fastp \  
-i /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R1.fastq.gz \  
-I /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R2.fastq.gz \  
--stdout | head
```

Success.

Note: Notice in the command-line above a `\` character is used. This allows us to split and run a long command across multiple lines which makes it easier to read. You can do this with any command, just be sure there are no spaces after the `\` character. It must always be followed by a new line.

Okay, now let's string it all together and pipe the output of `fastp` (cleaned reads) as the input for `bwa` (to align reads to the chloroplast), and then output a `.bam` file.

```
~/toomers-genome/bin/fastp \  
-i /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R1.fastq.gz \  
-I /scratch/JLGI_PCRfree_1_1_GGTACCTT_Quercus_virginiana_Toomers_I1126_L4_R2.fastq.gz \  
--stdout | bwa mem -t 3 -p Option_1_toomers-cp.fasta - | samtools view -F4 -Sb >  
↪ chloro_alignment.bam
```

There are three parts to this command, let's break them down:

1. First, we called `fastp` to clean our raw paired-end fastq data. We used the `-stdout` flag to output the cleaned reads to the **STDOUT**, meaning they can be streamed into another program.
2. We piped the output into BWA. We specified the `-p` option so `bwa` knew that the fastq data was interleaved, then we gave it out indexed chloroplast fasta assembly, and we told `bwa` that the input files were coming from the STDIN through a pipe by using `"-"` as the input file.
3. The output of BWA is an uncompressed **SAM** file with a `.sam` extension that contains information about every read that `bwa` processed, whether or not it aligned, and reports the location it maps to in the reference genome. Check out the linked guide to learn more about `.sam` format, and its compressed counterpart, `.bam`. These `.sam` files are BIG since they're uncompressed. And they include information about EVERY read, even the ones that don't align to the reference. Remember, we're aligning total DNA reads against a chloroplast, so only ~5% of our reads will align to the chloroplast reference. We can use some flags of `samtools` to filter out unmapped reads (`-F4`) and that we are feeding it a SAM file but want to output a compressed BAM file (`-Sb`). We'll spend a full day on `samtools` soon, but here's a quick primer.

4. Output the filtered .bam file to a new file called chloro_alignment.bam. Call it whatever you want!

Mastering Content

String it all together and visualize your .bam file in IGV

How do I visualize these .bam format read alignments against my reference chloroplast genome? IGV is a powerful alignment viewer. Download and install it on your computer. Download your chloroplast fasta assembly, the .bam alignment file, and the .bai index file to your computer.

```
# Sort your bam file to make it easier to process by putting the reads in order
# along each fasta entry. Output a sorted .bam file to "chloro_alignment.sort.bam"
# using 4 threads (-@ 4).
samtools sort -o chloro_alignment.sort.bam -@ 4 chloro_alignment.bam

# Index your bam file so IGV can read it
samtools index chloro_alignment.sort.bam

# Get some basic stats on how many reads aligned
samtools flagstat chloro_alignment.sort.bam
```

In IGV, load your reference chloroplast. It can be loaded by clicking on *Genomes >> Load Genome from File*.

Then you can drag and drop your .bam file into the main window and it will load the alignments. Here's a [great video from IGV](#) to get you started:

If you haven't already, go back to your GE-SEQ annotation of the chloroplast and download the .gff file of annotations. Drag and drop that gff into the main window to load your gene annotations and explore. It will take a while to load because there are so many reads. I wonder if there's a way to downsample our .bam file to reduce the number of reads... (can you find a solution?)

Poke around IGV and we'll talk about it in class and over group chat throughout the week. Have fun!

1.4.3 Lesson 3: Measuring Genome Complexity

2.3 Instructions

2.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Kew C-Value database](#)

2.3 Lesson

Learning Objectives

Vocabulary

Learning Material

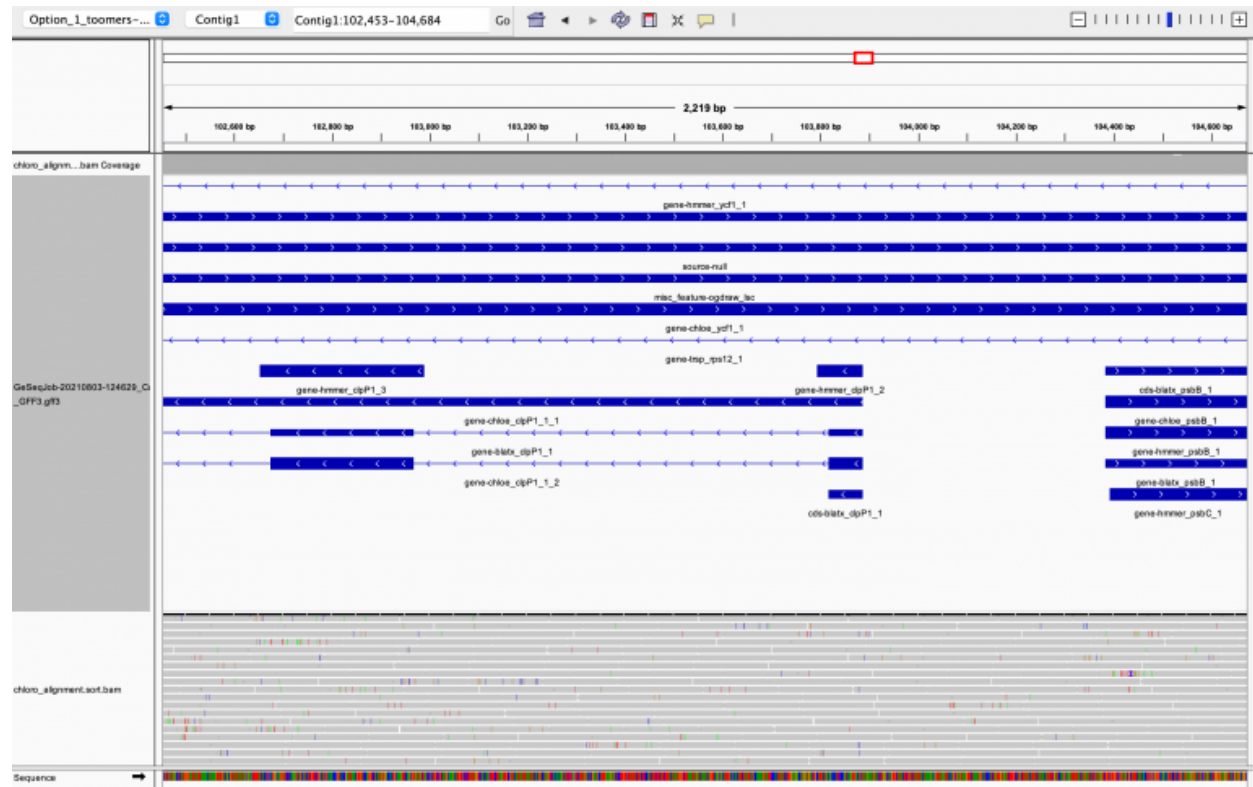


Fig. 14: IGV Screenshot

2.3 Lab Exercises

Overview

In this lab, we will learn how to use raw Illumina sequencing reads to genome complexity: genome size, heterozygosity size, and ploidy.

We will do three major things in this lab:

- Learn what kmers are
- Generate kmer frequencies with Jellyfish
- Run Jellyfish on our dataset

“If everything was perfect, you would never learn and you would never grow.” -Beyoncé

Task A:

Step 1: What is a k-mer?

Today we'll explore the power of k-mers and their application to genome sequencing.

K-mers are all possible substrings length k of a sequence string. A sequence string can refer to an Illumina read, a PacBio read, a gene assembly, a genome assembly, or any other string of nucleotides. We can search for kmers of any length k .

For a given sequence string length L , there are $L - K + 1$ possible k -mers in that string. In the example below, we have a sequence of length $(L) = 7$: ACTGGCT. To find all possible k -mers of length 3 (3-mers) in that string:

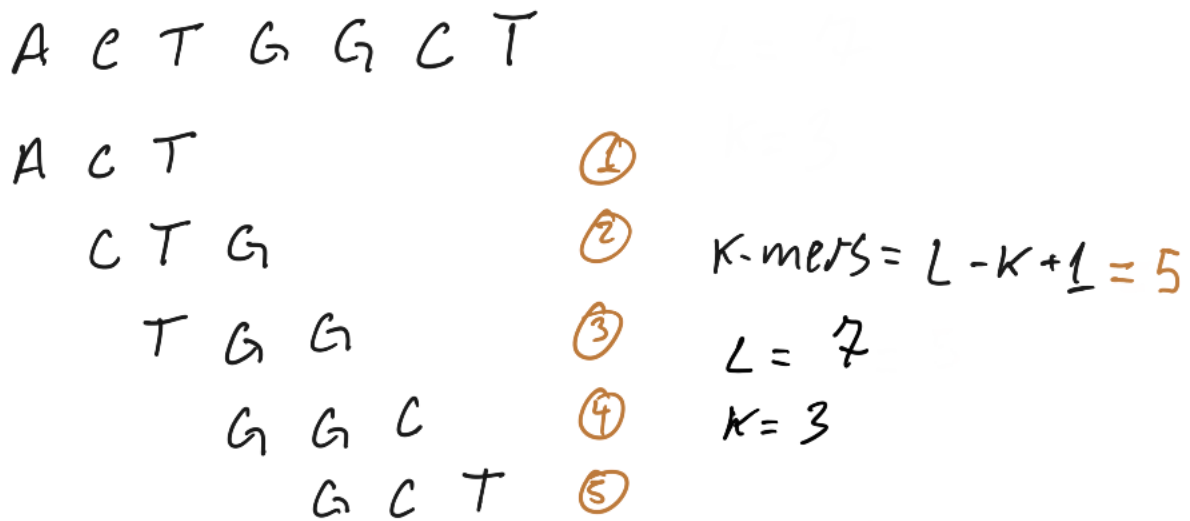


Fig. 15: Image Source: [K-mer analysis with python](#)

In this example, all possible k -mers were unique and appeared only once. When k -mers appear more than once, we can count their frequency, e.g. 3-mer “ACT” appears 33 times in the string.

Step 2: Calculate all possible 21-mers in our dataset

First, make a new directory in shotgun-data/ called kmer/. My tree directory looks like this now:

```

├── bin
│   └── fastp
├── hi-c
├── pacbio
├── rna-seq
├── shotgun-dna
│   ├── chloroplast
│   │   ├── assemblathon_stats.pl
│   │   ├── chloro_alignment.bam
│   │   ├── chloro_alignment.sort.bam
│   │   ├── chloro_alignment.sort.bam.bai
│   │   ├── Contigs_1_toomers-cp.fasta
│   │   ├── contigs_tmp_toomers-cp.txt
│   │   ├── fastp.html
│   │   ├── fastp.json
│   │   ├── log_toomers-cp.txt
│   │   ├── Merged_contigs_toomers-cp.txt
│   │   ├── nohup.out
│   │   ├── novoplasty.cfg
│   │   ├── Option_1_toomers-cp.fasta
│   │   ├── Option_1_toomers-cp.fasta.amb
│   │   ├── Option_1_toomers-cp.fasta.ann
│   │   ├── Option_1_toomers-cp.fasta.bwt
│   │   ├── Option_1_toomers-cp.fasta.pac
│   │   ├── Option_1_toomers-cp.fasta.sa
│   │   ├── Option_2_toomers-cp.fasta
│   │   └── rbcl.fa
│   ├── fastp
│   │   ├── fastp.html
│   │   └── fastp.json
│   ├── fastqc
│   ├── kmer
│   └── raw-data

```

Next, install Jellyfish using Conda.

I left a reduced dataset of just 10,000 lines of your Toomer's Oak data in /scratch/ for us to play around with:

/scratch/test-kmer.fastq

Check out the help page for jellyfish count using

```
jellyfish count -help
```

Usage: jellyfish count [options] file:path+

Count k-mers in fasta or fastq files

Options (default value in (), *required):

-m, --mer-len=uint32	*Length of mer
-s, --size=uint64	*Initial hash size
-t, --threads=uint32	Number of threads (1)
--sam=PATH	SAM/BAM/CRAM formatted input file
-F, --Files=uint32	Number files open simultaneously (1)
-g, --generator=path	File of commands generating fast[ag]
-G, --Generators=uint32	Number of generators run simultaneously (1)
-S, --shell=string	Shell used to run generator commands (\$SHELL or /bin/sh)
-o, --output=string	Output file (mer_counts.jf)
-c, --counter-len=Length in bits	Length bits of counting field (7)
--out-counter-len=Length in bytes	Length in bytes of counter field in output (4)
-C, --canonical	Count both strand, canonical representation (false)
--bc=peath	Bloom counter to filter out singleton mers
--bf-size=uint64	Use bloom filter to count high-frequency mers
--bf-fp=double	False positive rate of bloom filter (0.01)
--if=path	Count only k-mers in this files
-Q, --min-qual-char=string	Any base with quality below this character is changed to N
--quality-start=int32	ASCII for quality values (64)
--min-quality=int32	Minimum quality. A base with lesser quality becomes an N
-p, --reprobes=uint32	Maximum number of reprobes (126)
--text	Dump in text format (false)
--disk	Disk operation. Do not do size doubling (false)
-L, --lower-count=uint64	Don't output k-mer with count < lower-count
-U, --upper-count=uint64	Don't output k-mer with count > upper-count
--timing=Timing file	Print timing information
--usage	Usage
-h, --help	This message
--full-help	Detailed help
-V, --version	Version

There are just a couple options we need to invoke, -m (what kmer size do we want, -s (how much memory do we want to use to store the kmers), and take note that we can also multi-thread it with -t. We use -C to count k-mers on both strands of DNA (top and bottom).

Here's how to run jellyfish to count all possible 4-mers in the test data:

```
jellyfish count -C -m 4 -s 100000000 /scratch/test-kmer.fastq > test.jf
```

The output is a compressed file called "mer_counts.jf" that is not human-readable. But we can query this file in many ways. For example

```
# Get some stats on the k-mers, including how many occur only once, how many
# distinct k-mers exist, and how many total k-mers exist.
jellyfish stats mer_counts.jf

# dump a fasta-like file with all the kmer's and their counts
jellyfish dump mer_counts.jf

# Count the frequency of a specific 4-mer, e.g. ATTG
jellyfish query mer_counts.jf ATTG
```

On your own: Generate a kmer count of this test dataset for k=7 and count the number of k-mers ATTCGAG.

Task B

Next, we will use Jellyfish and GenomeScope to build a kmer spectra.

A K-mer spectra is a graphical representation of a dataset showing how many short fixed length words (k-mers) appear a certain number of times. The frequency of occurrence is plotted on the x-axis and the number of k-mers on the y-axis. The k-mer spectra is composed of distributions representing groups of motifs at different frequencies in the sample, plus biases. Given not too many biases, the shape of the distributions provides a useful set of properties describing the biological sample, the sequencing process and the amount of useful data in the dataset.

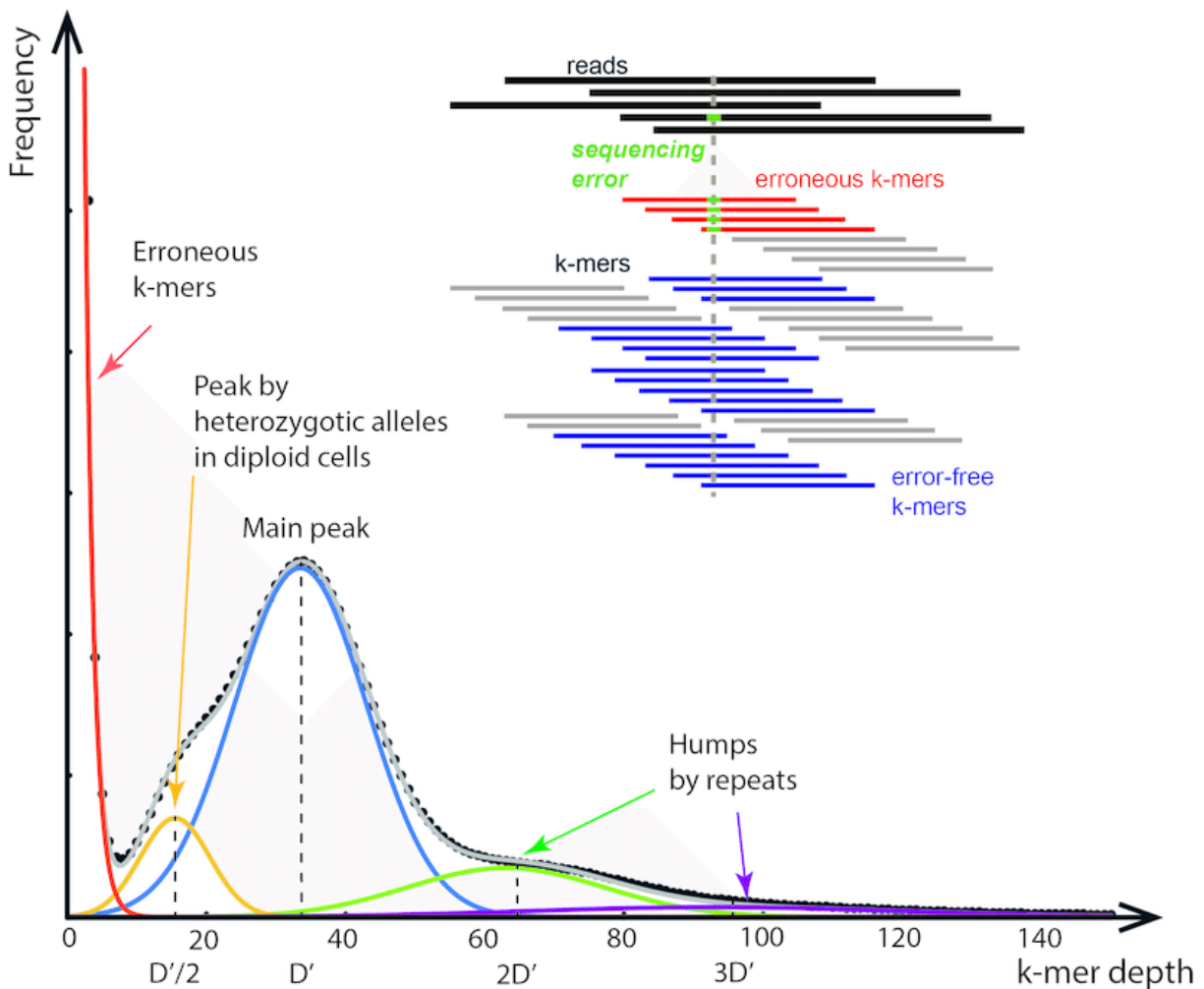
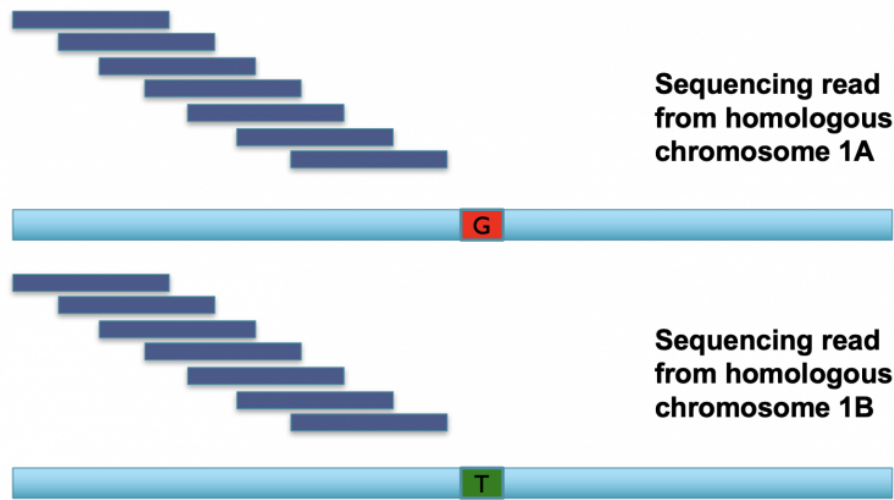


Fig. 16: K-mer histogram. The x-axis refers to the k-mer depth $D(k)$, which... Image Source: Jang-il Sohn, Jin-Wu Nam, *The present and future of de novo whole-genome assembly*, Briefings in Bioinformatics, Volume 19, Issue 1, January 2018, Pages 23–40, <https://doi.org/10.1093/bib/bbw096>

Sequencing errors occur randomly in Illumina sequencing. These will be represented in a kmer spectra as a high frequency (high on y-axis) of k-mers that occur just a few times (low on X-axis).

Let's review what heterozygosity looks like. In this example we have a diploid organism with two homologous chromosomes (A and B) for chromosome 1. If we have two sequencing reads that hit both A and B alleles, and we break up those reads into k-mers (the dark blue blocks), it looks like this:



The k-mers that span most of these two reads are identical, meaning we have two copies of every k-mer across most of the read, representing the shared parts of both alleles (aka the “haploid” representation of the genome). Once we find k-mers that span the mutation, however, we have k-mers that are unique to each allele (the “diploid” representation of the genome). Consequently, these diploid k-mers are present at 1/2 coverage relative to the rest of the k-mers in the read.

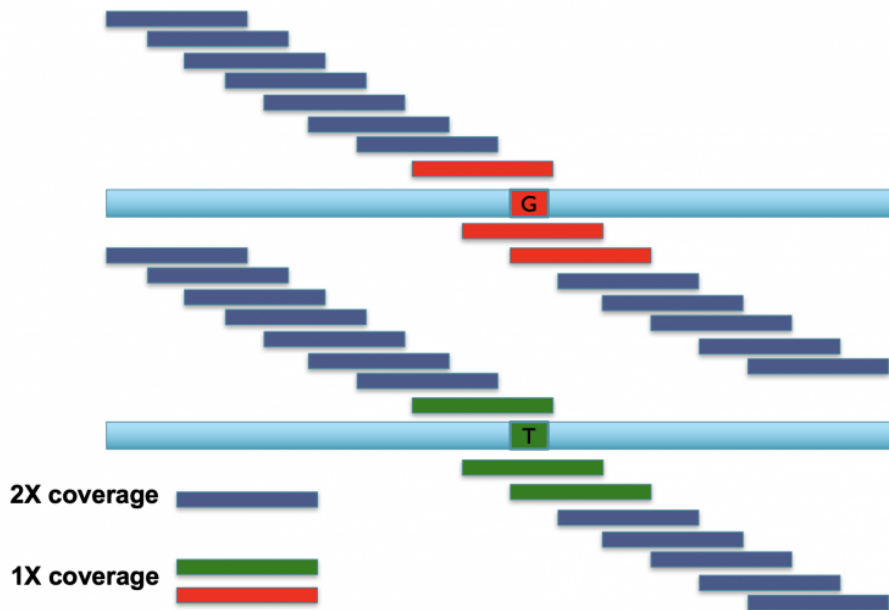
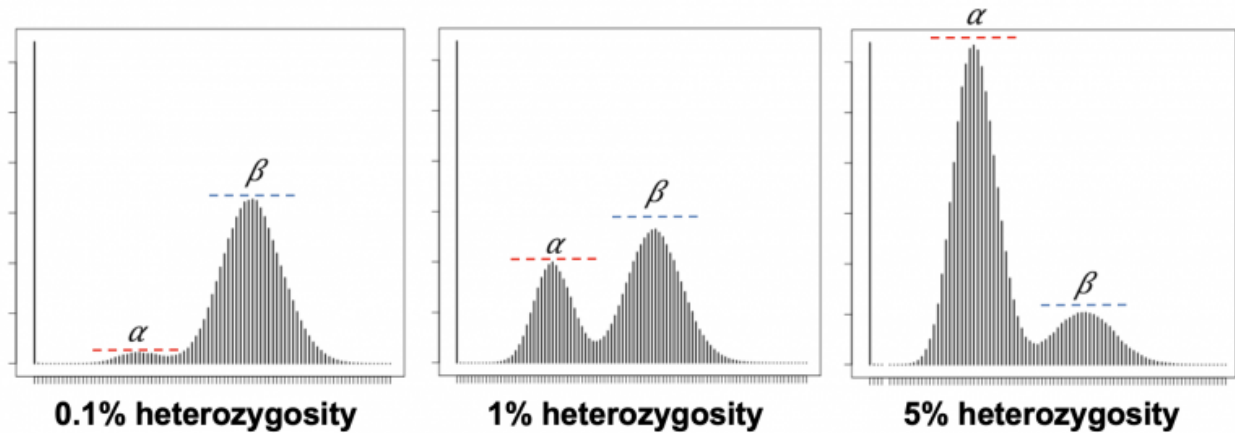


Fig. 17: Image Source: Mike Schatz

When you do this repeatedly across a diploid genome with shotgun Illumina reads, these kmer coverages can be used to calculate the heterozygosity of an organism. The heterozygous k-mers (a) are at 50% depth (the peak on the left) compared to the homozygous k-mers (the peak on the right). Comparing the relative heights of the diploid/heterozygous peak on the left, to the homozygous/haploid peak on the right, allows you to calculate heterozygosity. Increasing heterozygosity means that the left peak gets higher. X axis is coverage/depth of kmers, and the Y axis is the number of unique kmers at that given coverage/depth.



These data also tell us genome size, but we'll discuss that next lab.

Run Jellyfish as GenomeScope describes

GenomeScope can calculate heterozygosity for us, using shotgun sequencing reads. It tells us exactly how it wants to be run:

Instructions

Upload results from running Jellyfish. Example: [inputk21.hist](#)

Instructions for running Jellyfish:

1. Download and install jellyfish from: <http://www.genome.umd.edu/jellyfish.html#Release>
2. Count kmers using jellyfish:

```
$ jellyfish count -C -m 21 -s 1000000000 -t 10 *.fastq -o reads.jf
```

Note you should adjust the memory (-s) and threads (-t) parameter according to your server. This example will use 10 threads and 1GB of RAM. The kmer length (-m) may need to be scaled if you have low coverage or a high error rate. You should always use "canonical kmers" (-C)

3. Export the kmer count histogram

```
$ jellyfish histo -t 10 reads.jf > reads.histo
```

Again the thread count (-t) should be scaled according to your server.

4. Upload reads.histo to GenomeScope

Note: High copy-number DNA such as chloroplasts can confuse the model. Set a max kmer coverage to avoid this. Default is -1 meaning no filter.

Now, run Jellyfish count on the raw Toomer's Illumina data, except remember that we only have access to 4 threads, so change -t to 4:


```
jellyfish count -t 4 -C -m 21 -s 5000000000 /scratch/*.fastq.gz -o reads.jf
```

Mastering Content

If you run Jellyfish like this, you'll get an error like this one:

```
(toomers) student@landingvm:~/toomers-genome/shotgun-dna/kmer$ jellyfish count -t 4 -C -m 21 -s 1000000000 -t 10 /scratch/
*.fq.gz -o reads.jf
terminate called after throwing an instance of 'std::runtime_error'
  what(): Can't open file '/scratch/*.fq.gz'
Aborted (core dumped)
```

Jellyfish can't open a .fastq.gz file? Interesting. On your own and with your classmates, try and troubleshoot this issue.

Hint: I wonder if the [help page](#) has some clues for us.

1.4.4 Lesson 4: Plotting Heterozygosity and Size

2.4 Instructions

2.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

2.4 Lesson

Learning Objectives

Vocabulary

Learning Material

2.4 Lab Exercises

Overview

In this lab, we will learn how to use the Jellyfish kmer counting output to plot complexity: genome size, heterozygosity size, and ploidy.

We will do three major things in this lab:

- Generate a kmer histogram
- Run GenomeScope
- Interpret the output and make an assessment for the future of our project

“If everything was perfect, you would never learn and you would never grow.” -Beyoncé

Task A

Step 1: Finish running Jellyfish

By now, your Jellyfish k-mer counting run should be finished. There's one more step in order for us to run GenomeScope – to export the kmer count histogram:

Instructions

Upload results from running Jellyfish. Example: [inputk21.hist](#)

Instructions for running Jellyfish:

1. Download and install jellyfish from: <http://www.genome.umd.edu/jellyfish.html#Release>
2. Count kmers using jellyfish:

```
$ jellyfish count -C -m 21 -s 1000000000 -t 10 *.fastq -o reads.jf
```

Note you should adjust the memory (-s) and threads (-t) parameter according to your server. This example will use 10 threads and 1GB of RAM. The kmer length (-m) may need to be scaled if you have low coverage or a high error rate. You should always use "canonical kmers" (-C)

3. Export the kmer count histogram

```
$ jellyfish histo -t 10 reads.jf > reads.histo
```

Again the thread count (-t) should be scaled according to your server.

4. Upload reads.histo to GenomeScope

Note: High copy-number DNA such as chloroplasts can confuse the model. Set a max kmer coverage to avoid this. Default is -1 meaning no filter.

```
# we only have 4 threads, so change -t to 4
jellyfish histo -t 4 reads.jf > reads.histo
```

if your jellyfish run hasn't finished, I have left a copy of "reads.jf" in /scratch/

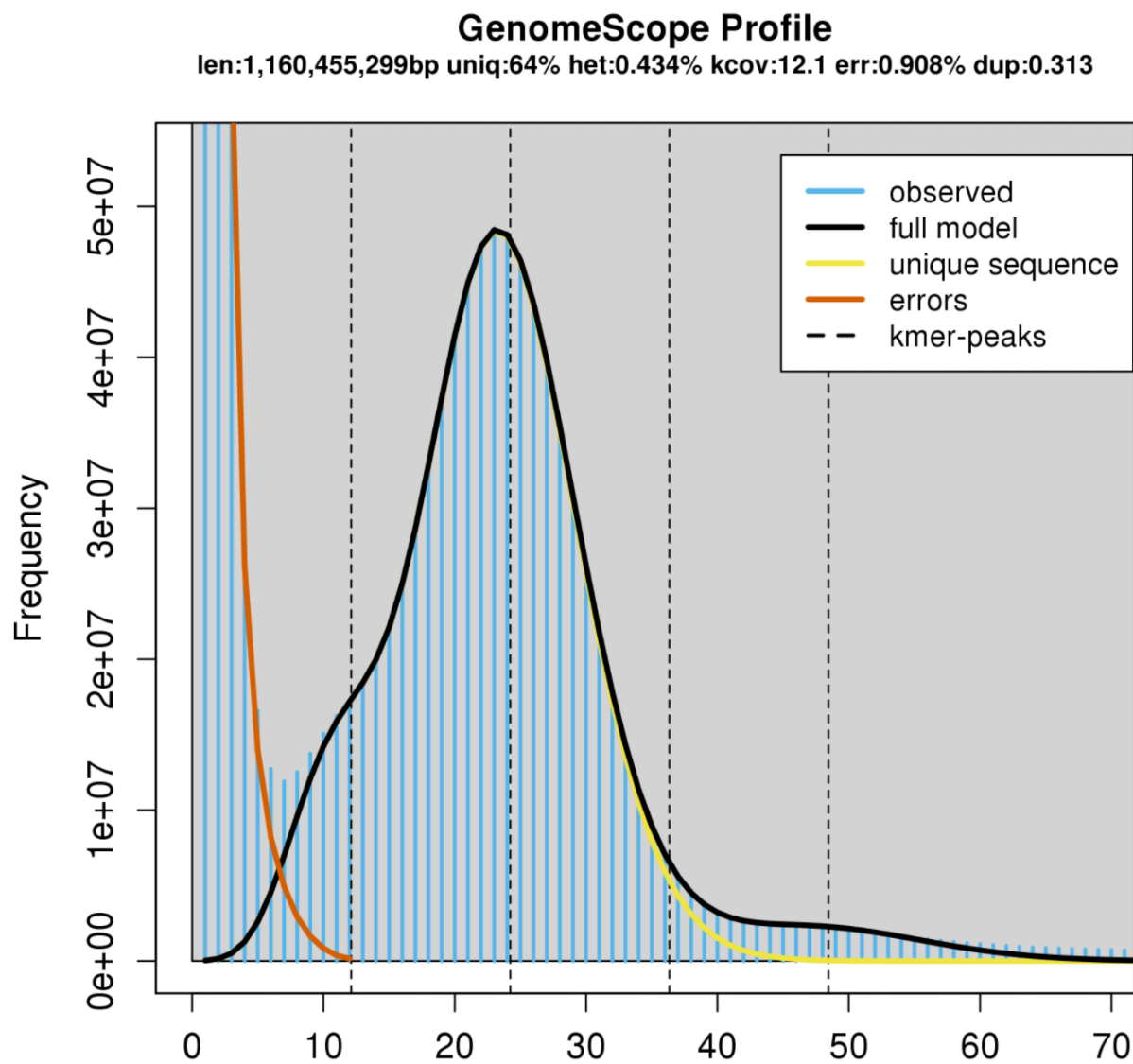
Note: There are some caveats that are important to remember. Extremely large (haploid size >>10GB) and/or very repetitive genomes may benefit from larger kmer lengths to increase the number of unique k-mers. Accurate inferences requires a minimum amount of coverage, at least 25x coverage of the haploid genome or greater, otherwise the model fit will be poor or not converge. We have much higher coverage for our Toomer's Oak genome, so we're all set.

Task B

Next, we will use Jellyfish and [GenomeScope](#) to build a kmer spectra. Let's dissect a few of these plots, first.

This is an example GenomeScope plot for a relatively low heterozygosity individual:

The big peak at 25 in the graph above is in fact the homozygous portions of the genome that account for the identical 21-mers from both strands of the DNA. The dotted line corresponds to the predicted center of that peak. The small shoulder to the left of the peak corresponds to the heterozygous portions of the genome that accounts for different 21-mers from each strand of the DNA. The two dotted lines to the right of the main peak (at coverage = 25) are the duplicated heterozygous regions and duplicated homozygous regions and correspond to two smaller peaks. The **shape** of these peaks are affected by the **sequencing errors** and **PCR duplicates**.

Fig. 18: Image Source: [Bioinformatics Workbook](#)

The terms in the plot are defined as:

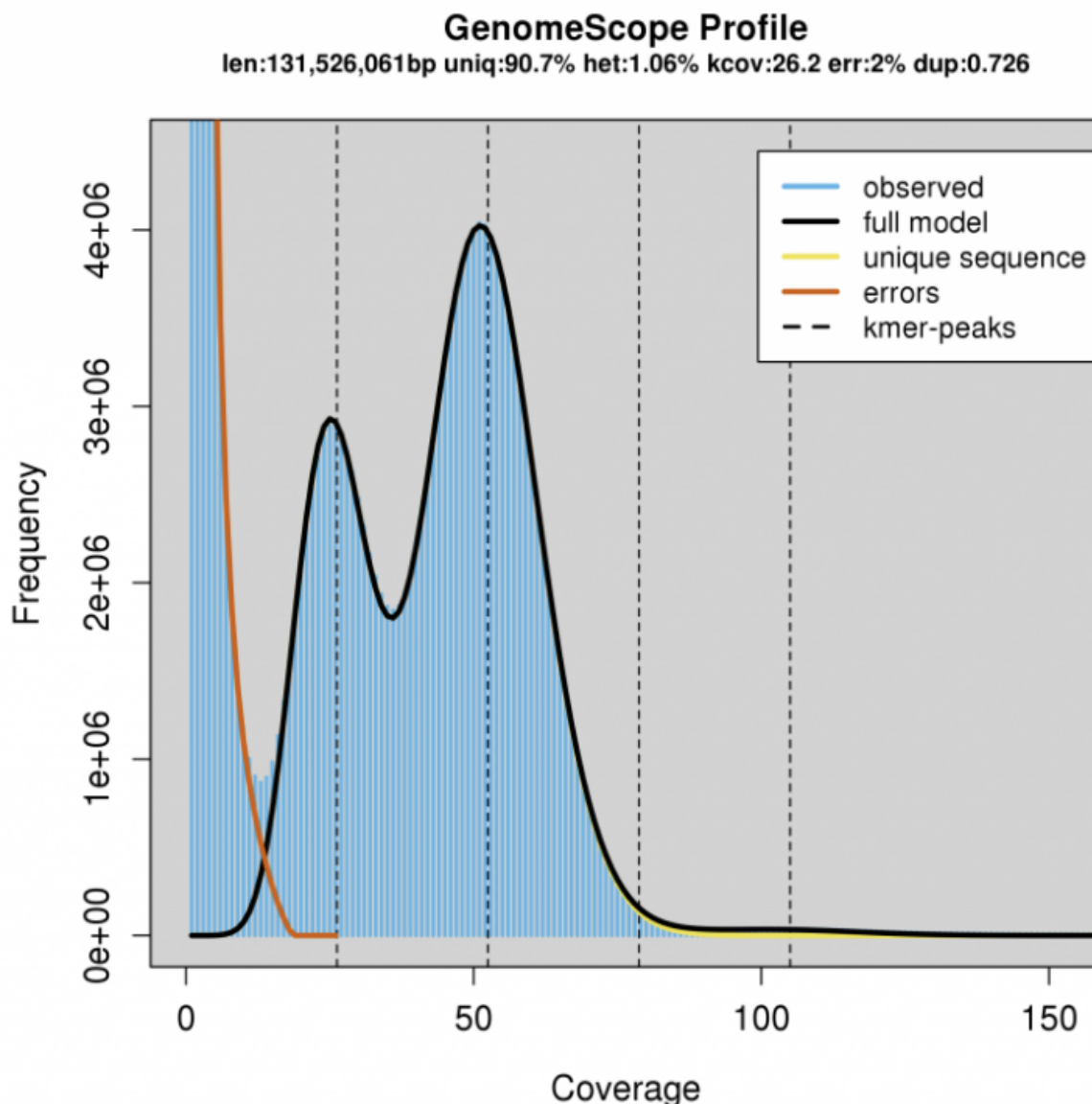
- **len**: inferred total genome length
- **uniq**: percent of the genome that is unique (not repetitive)
- **het**: overall rate of heterozygosity
- **kcov**: mean kmer coverage for heterozygous bases. note the top of the peak will not intersect the kcov line because of the over dispersion in real data
- **err**: error rate of the reads: average rate of read duplications

Calculating genome size: A genome size estimate is revised by summing the total number of k-mers, except presumptive sequencing errors identified as the far left part of the graph, and dividing by the 2*, the estimated coverage for homozygous k-mers. GenomeScope did this math for you, and presents it at **len** at the top.

In other words: subtract out the sequence error kmers, and divide the remaining total kmers by the haploid peak coverage.

On the other hand, here is an example of a relatively heterozygous individual (1.06%):

Estimating repeat content in a genome: Subtract 100-“uniq”. This will be the estimated % repetitive element content in the genome.



Note the high diploid peak at ~25X coverage, compared to the haploid peak at 50X coverage.

We can expect ~1% of sites in this genome to be heterozygous. This will create some unique differences in the way these two genomes will assemble with PacBio HiFi reads.

As you learned in the sequencing technology lesson, PacBio HiFi reads are highly accurate (>99%) and long (~20-30kb). In the “low heterozygosity” example, we would likely assemble an “unphased” assembly, meaning that haplotypes from the maternal and paternal chromosomes would be smashed together into a chimera.

On the other hand, for sufficiently heterozygous individuals, we can fully phase the maternal and paternal haplotypes of a diploid organism. In other words, we can fully assemble each chromosome pair separately. This is what the actual assembly graphs look like for “haplotype-resolved assembly construction”:

Run Jellyfish as GenomeScope describes

GenomeScope can be easily run by dragging/dropping your histogram file into the input box, change read length to 150, and click Submit.

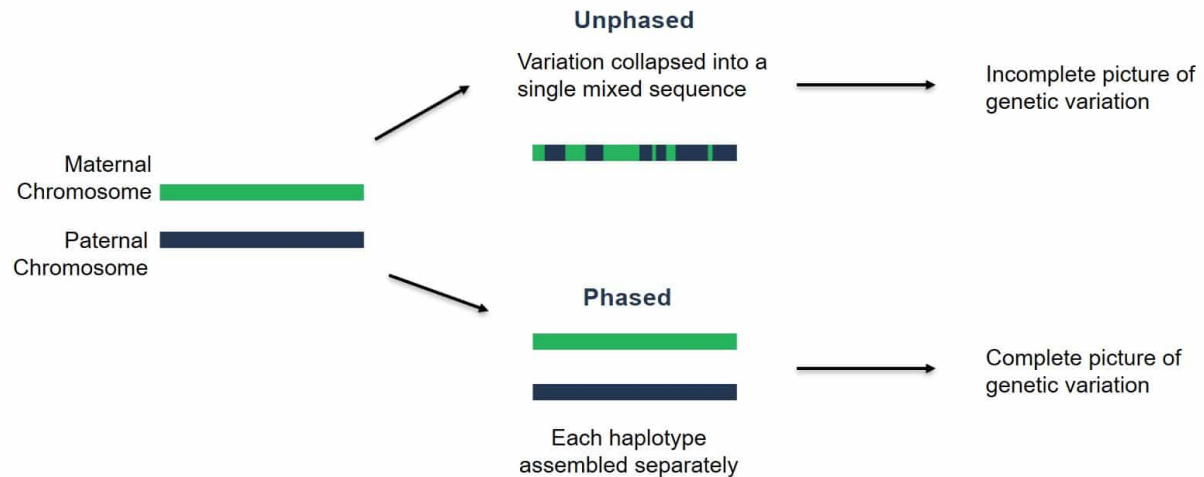


Fig. 19: Image Source: PacBio Website

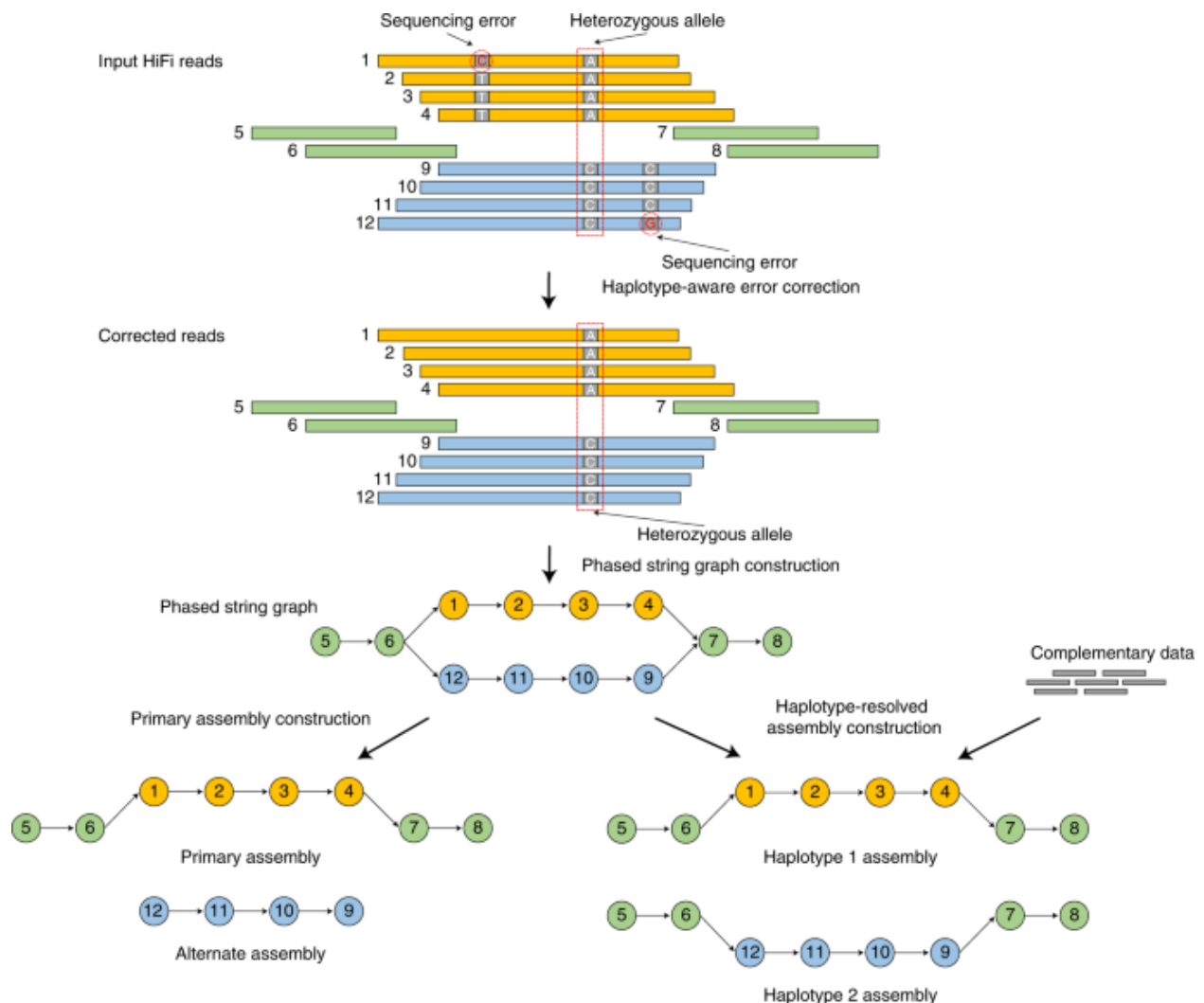


Fig. 20: Image Source: Cheng, H., Concepcion, G.T., Feng, X. et al. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nat Methods 18, 170–175 (2021). <https://doi.org/10.1038/s41592-020-01056-5>

Click or drop .histo file here to upload

Description my sample

Kmer length 21

Read length 150

Max kmer coverage 1000

Submit

Mastering Content

Based on your new genome size estimation, calculate the estimated coverage of Illumina PE150 reads that we sequenced. Edit the appropriate section in the manuscript. Fill in Supplemental Table 1 that includes information about the number of raw read pairs, and the number of trimmed read pairs after you ran fastp.

1.5 Module 3: Genome Assembly

1.5.1 Lesson 1: Assembly Algorithms

3.1 Instructions

3.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

3.1 Lesson

Learning Objectives

Vocabulary

Learning Material

3.1 Lab Exercises

Overview

In this lab, we will learn the basics of working with PacBio HiFi (CCS) reads.

We will do three major things in this lab:

- Explore the format of raw PacBio HiFi data
- Sanity check our data using NCBI BLAST

“If everything was perfect, you would never learn and you would never grow.” -Beyoncé

Task A

Step 1: Explore the format and do a sanity check

Let’s review the basics of PacBio HiFi sequencing, otherwise known as CCS (Circular Consensus Sequencing). If you need a quick refresher on PacBio HiFi sequencing, here’s a quick 2 minute clip. The premise of PacBio HiFi is that PacBio sequencing has an inherent ~8% error rate per molecule. However, if you keep rolling the circular template around the ZMW, you can generate 10-20+ passes (subreads) of that single molecule. The subreads can then be aligned and used to build a consensus HiFi read that is >99% accurate, depending on how many subread passes it contains.

In this class, we generated two PacBio HiFi flow cells worth of data. Each flow cell produces a `.fastq.gz` file. The raw PacBio HiFi data for Toomer’s Oak is in `/scratch` with two files:

- `m64103_210818_191603.fastq.gz`
- `m64103_210825_210414.fastq.gz`

First, explore the data a little bit. This is just a regular fastq file — four lines, except the lines are much longer than an Illumina fastq file. As a reminder, to explore just the first read:

```
zcat /scratch/m64103_210818_191603.fastq.gz | head -n 4
```

I like to do sanity checks on my data. Is this really the species I *think* it is? **BLAST is a simple way to check**. BLAST is an alignment and search algorithm that is widely used: it will become part of your everyday toolkit. NCBI uses BLAST in a way that allows you to search against the entirety of the collection; that is, you can search any sequence against every nucleotide in their database (which is a lot). Copy a chunk of sequence, e.g. 10 lines or more, and copy/paste it into the white box saying “Enter Query Sequence”. Change the “Program Selection: Optimize For...” to “somewhat similar sequences (blastn)”. Then press BLAST.

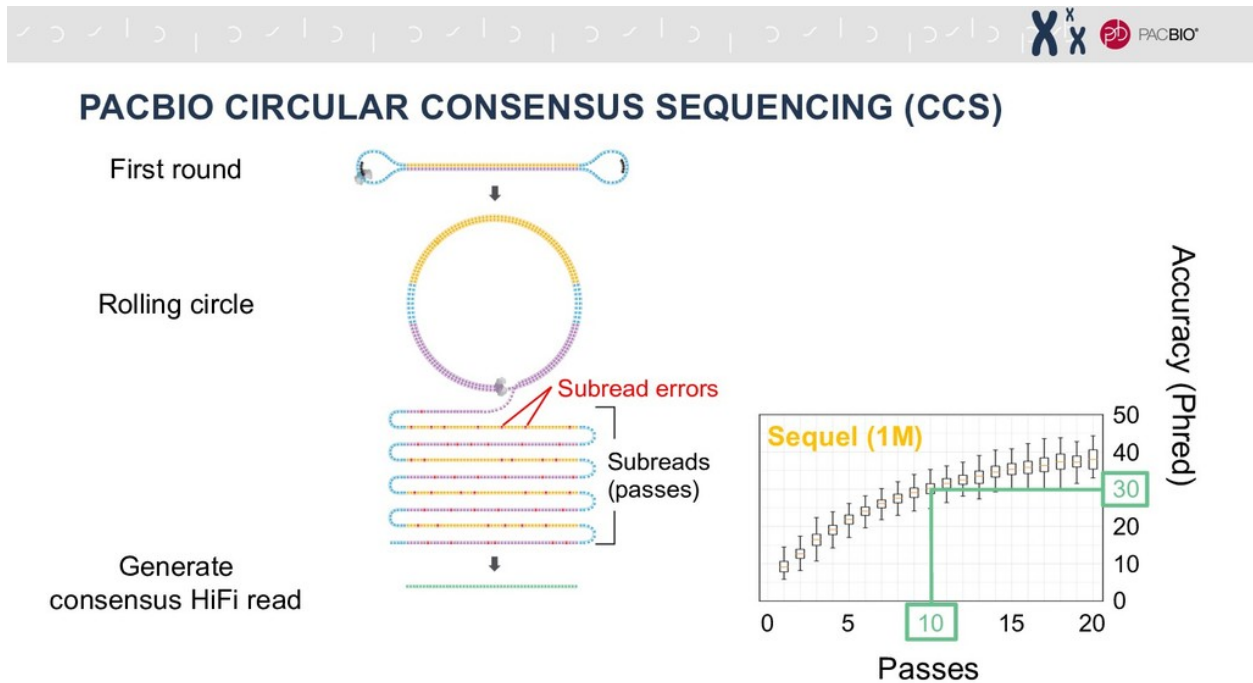


Fig. 21: Advantages of HiFi reads for variant discovery and genome assembly - Speaker Deck

U.S. National Library of Medicine
 National Center for Biotechnology Information

COVID-19 Information
[Public health information \(CDC\)](#) | [Research information \(NIH\)](#) | [SARS-CoV-2 data \(NCBI\)](#) | [Prevention and treatment information \(HHS\)](#)

BLAST® » blastn suite

Standard Nucleotide BLAST

blastn
blastp
blastx
tblastn
tblastx

Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s) [?](#) [Clear](#)

Query subrange [?](#)

From

To

Or, upload file

No file chosen [?](#)

Job Title

Enter a descriptive title for your BLAST search [?](#)

☐ Align two or more sequences [?](#)

Choose Search Set

Database

☒ Standard databases (nr etc.):
 ☐ rRNA/ITS databases
 ☐ Genomic + transcript databases
 ☐ Betacoronavirus

[?](#)

Organism

Optional

☐ exclude

Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown [?](#)

Exclude

Optional

☐ Models (XM/XP) ☐ Uncultured/environmental sample sequences

1.5. Module 3: Genome Assembly

65

Phew! Most of the top hits are *Quercus* species or related.

BLAST® » blastn suite » results for RID-MH712FRR016

[Home](#)
[Recent Results](#)
[Saved Strategies](#)
[Help](#)

[← Edit Search](#)
[Save Search](#)
[Search Summary ▾](#)

[How to read this report?](#)
[BLAST Help Videos](#)
[Back to Traditional Results Page](#)

Job Title

Nucleotide Sequence

RID

MH712FRR016 Search expires on 09-21 22:35 pm [Download All ▾](#)

Program

BLASTN [Citation ▾](#)

Database

nt [See details ▾](#)

Query ID

lc|Query_47889

Description

None

Molecule type

dna

Query Length

4151

Other reports

[Distance tree of results](#)
[MSA viewer](#)
[?](#)

Filter Results

Organism

only top 20 will appear

☐ exclude

Type common name, binomial, taxid or group name

[+ Add organism](#)

Percent Identity

to

E value

to

Query Coverage

to

[Filter](#)
[Reset](#)

Descriptions

Graphic Summary

Alignments

Taxonomy

Sequences producing significant alignments

[Download ▾](#)
[New Select columns ▾](#)
[Show 100 ▾](#)
[?](#)

☒ select all 100 sequences selected

[GenBank](#)
[Graphics](#)
[Distance tree of results](#)
[New MSA Viewer](#)

	Description ▾	Scientific Name ▾	Max Score ▾	Total Score ▾	Query Cover ▾	E value ▾	Per. Ident ▾	Acc. Len ▾	Accession
<input checked="" type="checkbox"/>	PREDICTED: Quercus suber cyclin-D1-1-like (LOC112034684). mRNA	Quercus suber	324	324	5%	6e-83	91.81%	1527	XM_024067505.1
<input checked="" type="checkbox"/>	PREDICTED: Quercus lobata cyclin-D2-1-like (LOC115967482). mRNA	Quercus lobata	306	306	4%	2e-77	93.69%	1523	XM_031086580.1
<input checked="" type="checkbox"/>	Fagus sylvatica genome assembly, chromosome: 7	Fagus sylvatica	284	602	18%	5e-71	85.88%	41494007	QU015767.1
<input checked="" type="checkbox"/>	Quercus robur BAC genome sequence, isolate 14B23	Quercus robur	263	510	5%	2e-64	85.48%	119231	LT799015.1
<input checked="" type="checkbox"/>	Corylus avellana genome assembly, chromosome: ca4	Corylus avellana	233	400	12%	8e-56	78.43%	36845065	LR899426.1
<input checked="" type="checkbox"/>	Carpinus viminea chromosome 3	Carpinus viminea	222	436	14%	5e-52	77.02%	46964676	CP054669.1
<input checked="" type="checkbox"/>	Quercus robur BAC genome sequence, isolate 87F12	Quercus robur	215	575	11%	2e-50	83.17%	135760	LT799034.1
<input checked="" type="checkbox"/>	PREDICTED: Quercus suber uncharacterized LOC112031182 (LOC112031182). transcript variant X2, nc...	Quercus suber	208	208	5%	3e-48	80.17%	2077	XR_002884957.1

Mastering Content

Your first job: Use any tool to 1) calculate the total number of bases sequenced, and 2) based on your GenomeScope genome size estimate, calculate the PacBio sequencing coverage. Use google and your classmates. Remember: you have total freedom. Install a useful piece of software if you find one. This is the essence of computational biology.

Insert your answer as a comment into the Google Doc: How many GB of sequencing data did we generate, and what X coverage of our estimated genome size did we sequence?

Bonus: Can you find a way to plot the read length distribution? Is there a piece of software that already exists?

1.5.2 Lesson 2: Building a Draft Genome

3.2 Instructions

3.2 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

3.2 Lesson

Learning Objectives

Vocabulary

Learning Material

3.2 Lab Exercises

Overview

In this lab, we will learn the basics of the hifiasm assembler, and launch an assembly with our PacBio HiFi data.

We will do two major things in this lab:

- Discuss the basics of haplotype-aware genome assembly
- Launch a hifiasm assembly

“Be for real, don’t be a stranger” - Spice Girls

Task A:

Step 1: What is haplotype-aware assembly?

Diploid genomes contain two copies of every chromosome. We call each of these copies a haplotype. Our new goal for genome assembly is to produce a haplotype-resolved assembly. Take the example below: two chromosome haplotypes, one from the maternal contribution and the other from the paternal contribution. Over the last decade, many genome assemblies have been produced that smash the two haplotypes together, switching between maternal/paternal haplotypes in a single chromosome representation of the assembly. In other words, these chromosome assemblies are incomplete.

Nowadays, we have better data that is longer and more accurate. Given sufficient heterozygosity in a sample, we can phase the chromosome haplotypes. In other words, we can produce two genome assemblies for every diploid genome.

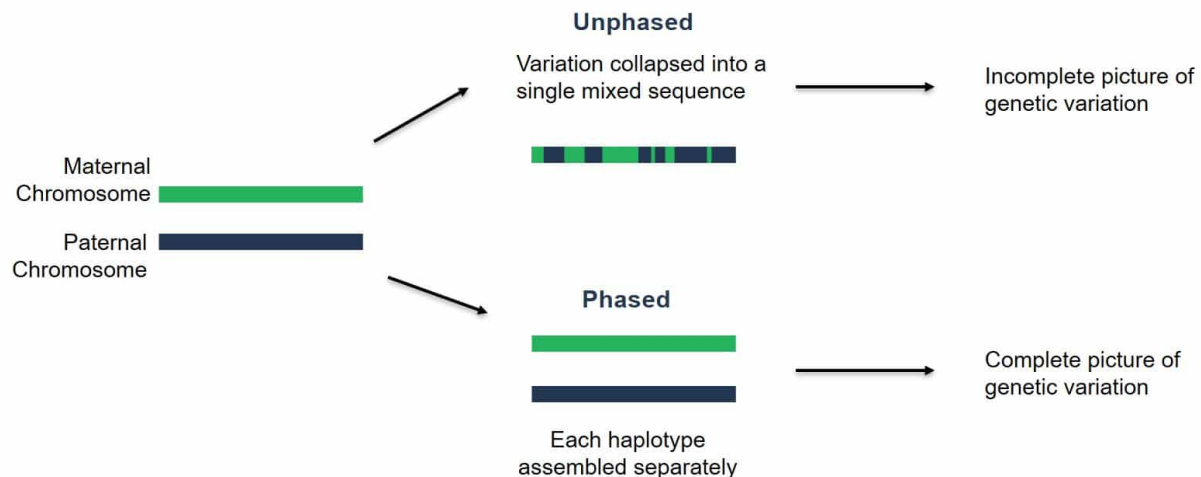


Fig. 22: Image Source: [PacBio Website](#)

One issue is that without some additional information from the two parents of a diploid individual, we don't often know which haplotype comes from the maternal versus paternal lineage. That is, unless you can also sequence the genomes of the parents! Here's an example:

Okay, let's review the basics of PacBio hifiasm assembler. Hifiasm leverages both the long PacBio reads, and k-mers derived from those reads, to identify 1) errors in the PacBio data, and 2) putative heterozygous allele sites in the data. First, it effectively recapitulates what you did with GenomeScope: it builds a k-mer distribution from the raw reads to identify a homozygous and heterozygous peak and their coverages.

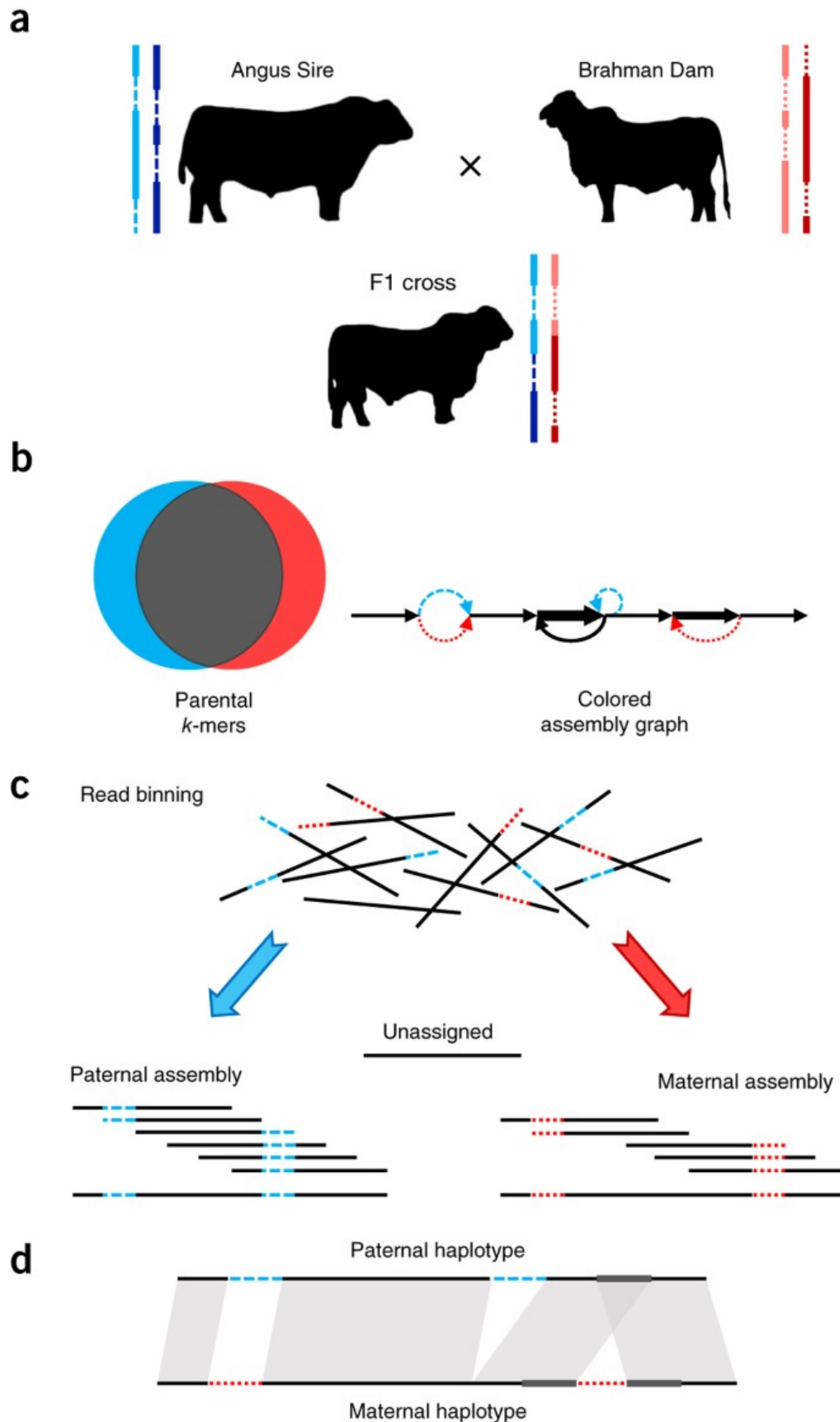


Fig. 23: (a) Two parents constitute four haplotypes, including shared sequences in both parents (solid lines) and sequences unique to one parent (dashed lines). The offspring inherits a recombined haplotype from each parent (blue, paternal; red, maternal). (b) Short-read sequencing of the parents identifies unique length- k subsequences (k -mers), which can be used to infer the origin of heterozygous alleles in the offspring's diploid genome. (c) Trio binning simplifies assembly by first partitioning long reads from the offspring into paternal and maternal sets on the basis of these

```

M:ha_analyze_count] lowest: count[7] = 2869
M:ha_analyze_count] highest: count[42] = 982180
M:ha_hist_line] 1: ***** 1538421
M:ha_hist_line] 2: ***** 150746
M:ha_hist_line] 3: ** 24452
M:ha_hist_line] 4: * 8120
M:ha_hist_line] 5: 4609
M:ha_hist_line] 6: 3521
M:ha_hist_line] 7: 2469
M:ha_hist_line] 8: 3804
M:ha_hist_line] 9: 2797
M:ha_hist_line] 10: 2853
M:ha_hist_line] 11: 3629
M:ha_hist_line] 12: 4388
M:ha_hist_line] 13: * 5835
M:ha_hist_line] 14: * 7198
M:ha_hist_line] 15: * 9482
M:ha_hist_line] 16: * 2212
M:ha_hist_line] 17: ** 15966
M:ha_hist_line] 18: ** 20388
M:ha_hist_line] 19: *** 25419
M:ha_hist_line] 20: *** 32438
M:ha_hist_line] 21: *** 43386
M:ha_hist_line] 22: ***** 54867
M:ha_hist_line] 23: ***** (9791
M:ha_hist_line] 24: ***** 88946
M:ha_hist_line] 25: ***** 109457
M:ha_hist_line] 26: ***** 138784
M:ha_hist_line] 27: ***** 169322
M:ha_hist_line] 28: ***** 207327
M:ha_hist_line] 29: ***** 247532
M:ha_hist_line] 30: ***** 295538
M:ha_hist_line] 31: ***** 349859
M:ha_hist_line] 32: ***** 413295
M:ha_hist_line] 33: ***** 479258
M:ha_hist_line] 34: ***** 551656
M:ha_hist_line] 35: ***** 627935
M:ha_hist_line] 36: ***** 703218
M:ha_hist_line] 37: ***** 779912
M:ha_hist_line] 38: ***** 842168
M:ha_hist_line] 39: ***** 901890
M:ha_hist_line] 40: ***** 959193
M:ha_hist_line] 41: ***** 978961
M:ha_hist_line] 42: ***** 982180
M:ha_hist_line] 43: ***** 981387
M:ha_hist_line] 44: ***** 990648
M:ha_hist_line] 45: ***** 913433
M:ha_hist_line] 46: ***** 862795
M:ha_hist_line] 47: ***** 801753
M:ha_hist_line] 48: ***** 734744
M:ha_hist_line] 49: ***** 668369
M:ha_hist_line] 50: ***** 589961
M:ha_hist_line] 51: ***** 517582
M:ha_hist_line] 52: ***** 444698
M:ha_hist_line] 53: ***** 378998
M:ha_hist_line] 54: ***** 320529
M:ha_hist_line] 55: ***** 268270
M:ha_hist_line] 56: ***** 225684
M:ha_hist_line] 57: ***** 186385
M:ha_hist_line] 58: ***** 156980
M:ha_hist_line] 59: ***** 133021
M:ha_hist_line] 60: ***** 117426
M:ha_hist_line] 61: ***** 106320
M:ha_hist_line] 62: ***** 99988
M:ha_hist_line] 63: ***** 95835
M:ha_hist_line] 64: ***** 94881
M:ha_hist_line] 65: ***** 98337
M:ha_hist_line] 66: ***** 104326
M:ha_hist_line] 67: ***** 108796
M:ha_hist_line] 68: ***** 118380
M:ha_hist_line] 69: ***** 127824
M:ha_hist_line] 70: ***** 137938
M:ha_hist_line] 71: ***** 149793
M:ha_hist_line] 72: ***** 162446
M:ha_hist_line] 73: ***** 174992
M:ha_hist_line] 74: ***** 189295
M:ha_hist_line] 75: ***** 203187
M:ha_hist_line] 76: ***** 216653
M:ha_hist_line] 77: ***** 229978
M:ha_hist_line] 78: ***** 240515
M:ha_hist_line] 79: ***** 253384
M:ha_hist_line] 80: ***** 263535
M:ha_hist_line] 81: ***** 273823
M:ha_hist_line] 82: ***** 284086
M:ha_hist_line] 83: ***** 294620
M:ha_hist_line] 84: ***** 298936
M:ha_hist_line] 85: ***** 298837
M:ha_hist_line] 86: ***** 297758
M:ha_hist_line] 87: ***** 294328
M:ha_hist_line] 88: ***** 287735
M:ha_hist_line] 89: ***** 281648
M:ha_hist_line] 90: ***** 273587
M:ha_hist_line] 91: ***** 261908
M:ha_hist_line] 92: ***** 247226
M:ha_hist_line] 93: ***** 234483
M:ha_hist_line] 94: ***** 219288
M:ha_hist_line] 95: ***** 205768
M:ha_hist_line] 96: ***** 189174
M:ha_hist_line] 97: ***** 172269
M:ha_hist_line] 98: ***** 158441
M:ha_hist_line] 99: ***** 142831
M:ha_hist_line] 100: ***** 126957
M:ha_hist_line] 101: ***** 110993
M:ha_hist_line] 102: ***** 10018
M:ha_hist_line] 103: ***** 87054
M:ha_hist_line] 104: ***** 77573
M:ha_hist_line] 105: ***** 66620
M:ha_hist_line] 106: ***** 57433
M:ha_hist_line] 107: ***** 48906
M:ha_hist_line] 108: **** 42493
M:ha_hist_line] 109: *** 36698
M:ha_hist_line] 110: *** 31234
M:ha_hist_line] 111: *** 26086
M:ha_hist_line] 112: ** 22797
M:ha_hist_line] 113: ** 19768
M:ha_hist_line] 114: ** 17905
M:ha_hist_line] 115: ** 16074
M:ha_hist_line] 116: * 14644
M:ha_hist_line] 117: * 13017
M:ha_hist_line] 118: * 12238
M:ha_hist_line] 119: * 11281
M:ha_hist_line] 120: * 10792
M:ha_hist_line] 121: * 10359
M:ha_hist_line] 122: * 10179
M:ha_hist_line] 123: * 10283
M:ha_hist_line] 124: * 10205
M:ha_hist_line] 125: * 10237
M:ha_hist_line] 126: * 10070
M:ha_hist_line] 127: * 9695
M:ha_hist_line] 128: * 9749
M:ha_hist_line] 129: * 9753
M:ha_hist_line] 130: * 9557
M:ha_hist_line] 131: * 9472
M:ha_hist_line] 132: * 9229
M:ha_hist_line] 133: * 9195
M:ha_hist_line] 134: * 9231

```

Then, hifiasm builds an OLC graph that finds a path through heterozygous alleles — they look like bubbles in the graph below. Hifiasm can produce two kinds of assemblies — primary + alt (left), or a phased haplotype 1 + haplotype 2 assembly. The “primary” assembly is the best path for one haplotype per contig. The “alt” is the alternative haplotype.

In this class, we generated two PacBio HiFi flow cells worth of data. Given our computational resources on the Virtual Machines (4 threads, 32 GB RAM) we can only use so much data as input. So I’ve subsetting the data to be smaller, so that it can still run on our VMs. I subsetting the data to 20 Gigabases (Gb)

```
# data in /scratch # toomers.20G.subset.fastq.gz
```

Check out the hifiasm github page. Install the software on your own in your ~/toomers-genome/bin/ directory (or whatever you named it). Here’s the example for how to assemble heterozygous genomes:

```
# Assemble heterozygous genomes with built-in duplication purging
hifiasm -o HG002.asm -t32 HG002-file1.fq.gz HG002-file2.fq.gz
```

Before running hifiasm, read the tutorial first: <https://hifiasm.readthedocs.io/en/latest/pa-assembly.html#pa-assembly>

HiFi-only Assembly

A typical hifiasm command line looks like:

```
hifiasm -o NA12878.asm -t 32 NA12878.fq.gz
```

where `NA12878.fq.gz` provides the input reads, `-t` sets the number of CPUs in use and `-o` specifies the prefix of output files. Input sequences should be FASTA or FASTQ format, uncompressed or compressed with gzip (.gz). The quality scores of reads in FASTQ are ignored by hifiasm. Hifiasm outputs assemblies in GFA format.

Remember that you only have 4 threads, so adjust `-t` accordingly. Launch your job using the `toomers.20G.subset.fastq.gz` HiFi reads.

Mastering Content

After your assembly finishes:

1. The assembly file that we’ll focus on is the primary contig assembly. But it’s in a .gfa format. . . Find a solution on google to convert “toomers.subset.gfa.bp.p_ctg.gfa” to a .fasta file.
2. Then use the `assemblathon_stats.pl` script to calculate basic statistics about your assembly. What is the total size, what is the `contig N50`?

So I have an Assembly... Now What?

Step 1: Understand the output

Hifiasm outputs a handful of files:

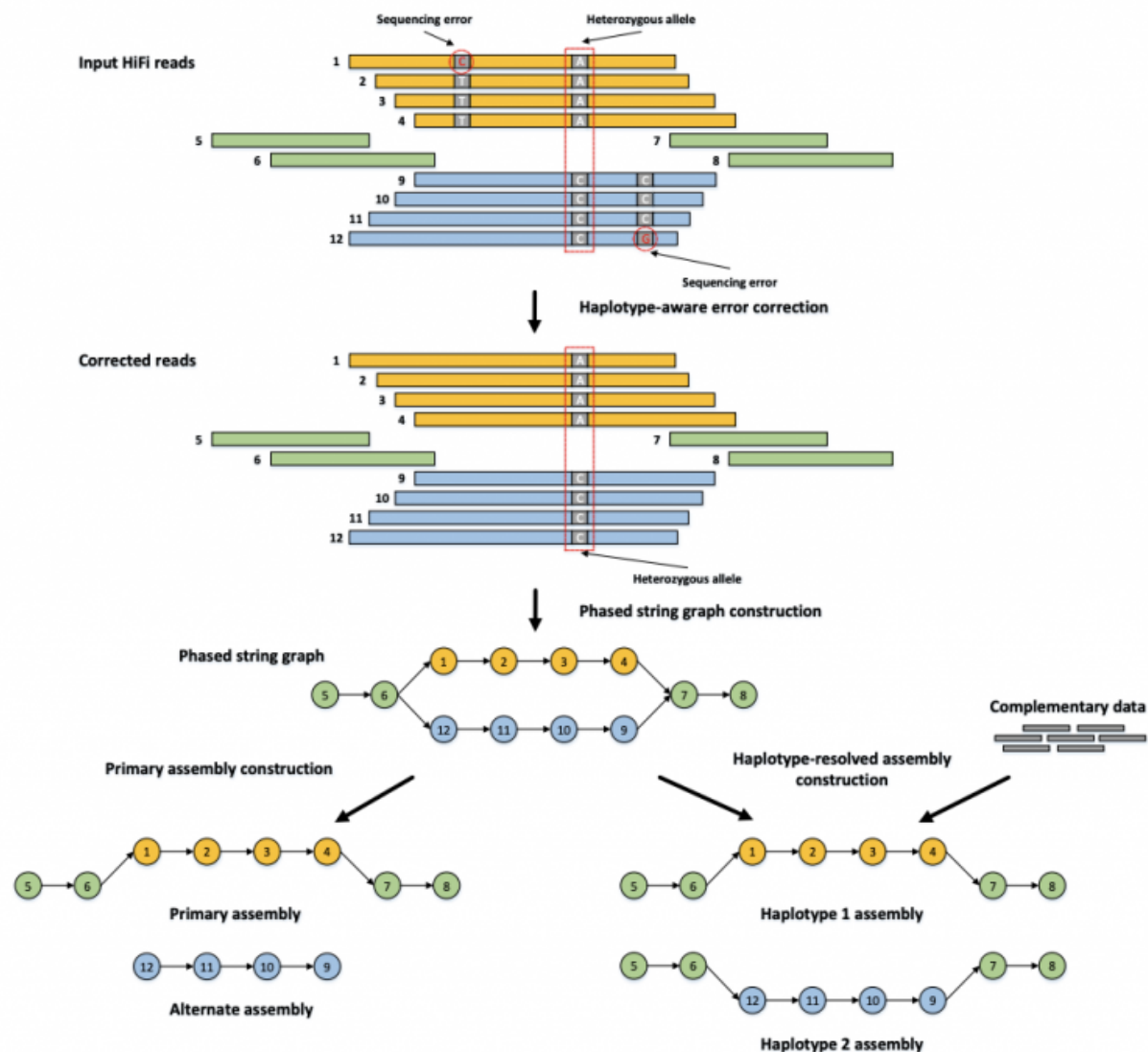


Fig. 24: Orange and blue bars represent the reads with heterozygous alleles carrying local phasing information, while green bars come from the homozygous regions without any heterozygous alleles. In the phased string graph, a vertex corresponds to the HiFi read with the same ID, and an edge between two vertices indicates that their corresponding reads are overlapped with each other. Hifiasm first performs haplotype-aware error correction to correct sequence errors but keep heterozygous alleles, and then builds a phased assembly graph with local phasing information from the corrected reads. Only the reads coming from the same haplotype are connected in the phased assembly graph. With complementary data providing global phasing information, hifiasm generates a completely phased assembly for each haplotype from the graph. Hifiasm also can generate an unphased primary assembly only with HiFi reads. This unphased primary assembly represents phased blocks (regions) that are resolvable with HiFi reads, but does not preserve phasing information between two phased blocks. Image Source: Cheng, H., Concepcion, G.T., Feng, X. et al. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nat Methods* 18, 170–175 (2021). <https://doi.org/10.1038/s41592-020-01056-5>

Output files

In general, hifiasm generates the following assembly graphs in the GFA format:

- ``prefix`.r_utg.gfa`: haplotype-resolved raw unitig graph. This graph keeps all haplotype information.
- ``prefix`.p_utg.gfa`: haplotype-resolved processed unitig graph without small bubbles. Small bubbles might be caused by somatic mutations or noise in data, which are not the real haplotype information. Hifiasm automatically pops such small bubbles based on coverage. The option `--hom-cov` affects the result. See [homozygous coverage setting](#) for more details. In addition, the option `-p` forcibly pops bubbles.
- ``prefix`.p_ctg.gfa`: assembly graph of primary contigs. This graph includes a complete assembly with long stretches of phased blocks.
- ``prefix`.a_ctg.gfa`: assembly graph of alternate contigs. This graph consists of all contigs that are discarded in primary contig graph.
- ``prefix`.hap*.p_ctg.gfa`: phased contig graph. This graph keeps the phased contigs.

Let's go over some of this terminology, first. The [PacBio manual](#) is quite helpful here:

Consensus	Given a collection of overlapping reads, that do not precisely match along their overlaps, a consensus sequence for the collection is one for which the sum of the differences between the consensus sequence and each one of the reads is minimal.
Contig	A maximal set of reads in a layout which in aggregate cover a contiguous interval.
Degenerate	A unitig that could not be combined into any scaffold. It is like a singleton but it has more than one read. Degenerates sometimes contain high-copy plasmid sequence. Degenerates can reflect biological phenomena that undermine the assumptions of Celera Assembler's mathematical model.
Fragment	Either a guide or a read. Unfortunately this term has a long history of different uses. For instance, one may actually be talking about inserts. Usually the intended meaning is clear from context, but when it isn't and it is important to understand the precise meaning, be sure to ask for clarification.
Scaffold	A maximal set of contigs in a layout that are connected together by mate-links.
Singleton	A read that could not be assembled. Singletons can represent contamination, unique sequence with no overlap due to the fluctuation of random coverage, or sequence with so many overlaps it could not be assembled efficiently. It can happen that a mate pair has two singletons, and in some contexts these pairs are called mini-scaffolds.
Surrogate	A unitig whose arrival rate statistics was beyond the expected range. Such unitigs are treated as collapsed repeats. Their consensus may get placed in one or more scaffolds. Some of their reads may get placed, by mates, late in the pipeline. When a repetitive unitig cannot be placed even once, it becomes a degenerate.
Unitig	A uniquely assembleable subset of overlapping fragments. A unitig is an assembly of fragments for which there are no competing choices in terms of internal overlaps. This means that a unitig is either a correctly assembled portion of a contig or it is an overcompressed assembly of several high-fidelity copies of a repeat.

For your assembly, using just the HiFi reads, you produced an assembly like on the left. Just the primary contigs:

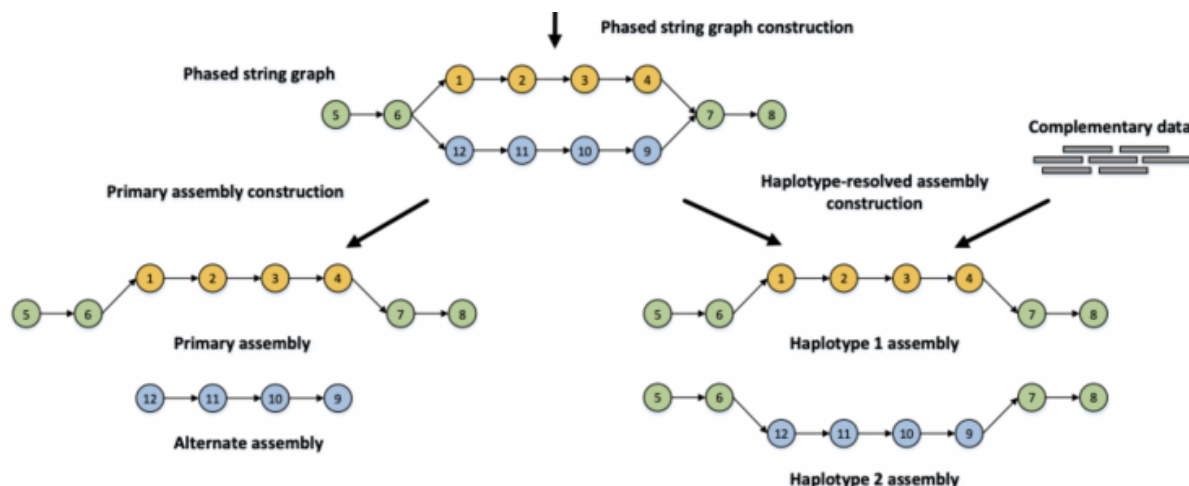


Fig. 25: Image Source: Cheng, H., Concepcion, G.T., Feng, X. et al. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nat Methods 18, 170–175 (2021). <https://doi.org/10.1038/s41592-020-01056-5>

Step 2: Get the basic stats.

You should have run `assemblathon_stats.pl` on your “`toomers.subset.gfa.bp.p_ctg.fasta`” assembly. It reports a handful of statistics, both on the contigs and the scaffolds. We’ll talk about the difference between these two things in class. In short, scaffolds have gaps (NNNNNNNN) of unknown length that connect contigs together. Contigs are contiguous, meaning no gaps.

Molecular Ecologist describes N50 in a simple way: Imagine that you line up all the contigs in your assembly in the order of their sequence lengths (Fig. 1a). You have the longest contig first, then the second longest, and so on with the shortest ones in the end. Then you start adding up the lengths of all contigs from the beginning, so you take the longest contig + the second longest + the third longest and so on — all the way until you’ve reached the number that is making up 50% of your total assembly length. That length of the contig that you stopped counting at, this will be your N50 number.

Step 2: Figure out the lengths of contigs

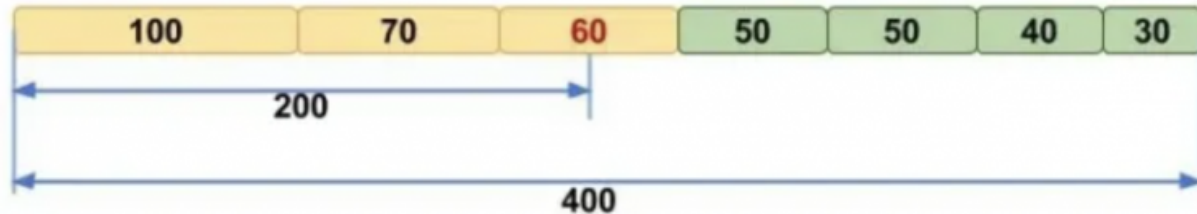
Here’s another one of those one-liners that I keep around in my back pocket for things like this. Change “`assembly.fasta`” to whatever your assembly is called.

```
cat assembly.fasta \
| awk '{c+=length($0);} END { print c; }' \
| awk '{print $1,$3}' \
| sort -nk 2
```

You can copy/paste this into Excel or Google Sheets if that helps. How many haploid chromosome does *Q. virginiana* have? How many large contigs do we have? Wow !



1a. Contigs, sorted according to their lengths.



1b. Calculation of N_{50} using sorted contigs.

Fig. 1. Example of calculating N_{50} for a set of seven contigs.
Here N_{50} equals 60 kbp.

Fig. 26: Image Source: Molecular Ecologist

Step 3: Check out the assembly produced with ALL of the data, using the Hi-C integrated build

When you add in Hi-C data to the assembly process, hifiasm is allowed to use an additional data type to phase the two haplotypes. I've run the exact same command as you all, adding both flow cells worth of data, plus the Hi-C data, and started a hifiasm run. Just like the assembly on the right side:

The output that matters the most to us, the two phased haplotype fasta files, can be found in scratch:

```
hifiasm.hic.gfa.hic.hap1.p_ctg.fasta
```

```
hifiasm.hic.gfa.hic.hap2.p_ctg.fasta
```

First, we want to see how similar these two assemblies are in terms of length.

Next, how different are they in terms of structural variations? *Assemblytics* is a nifty online GUI that can build dotplots that compare two reference genome assemblies. Download and install MUMMER (<https://sourceforge.net/projects/mummer/files/mummer/3.23/>).

If you use Conda, make sure you download MUMMER3 and NOT MUMMER4, or else everything will break.

I keep this dotplot reference handy for how to interpret dotplots that compare a Reference versus a Query.

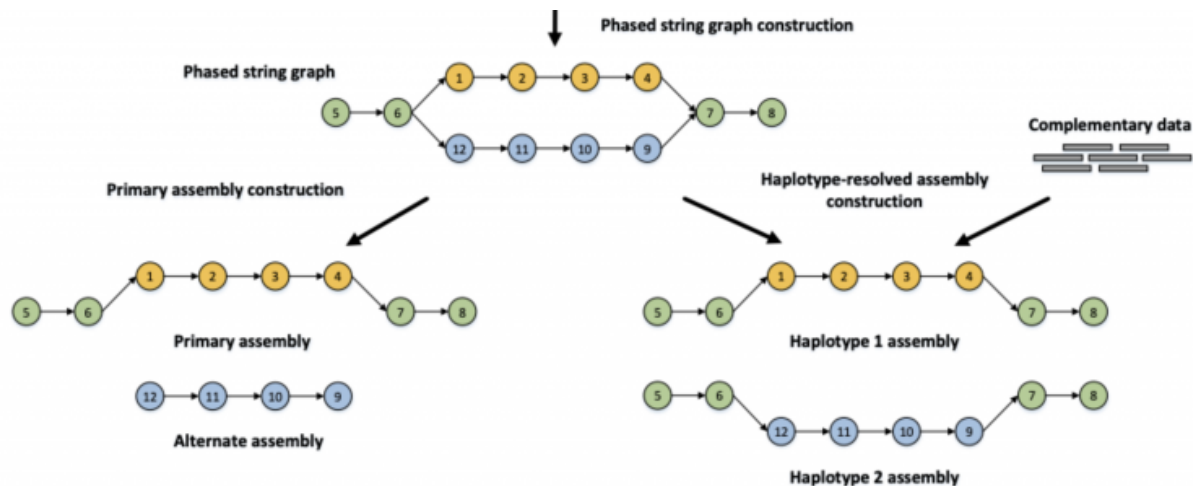


Fig. 27: Image Source: Cheng, H., Concepcion, G.T., Feng, X. et al. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nat Methods 18, 170–175 (2021). <https://doi.org/10.1038/s41592-020-01056-5>

Assessing haplotypes

Step 1: Run Assemblytics

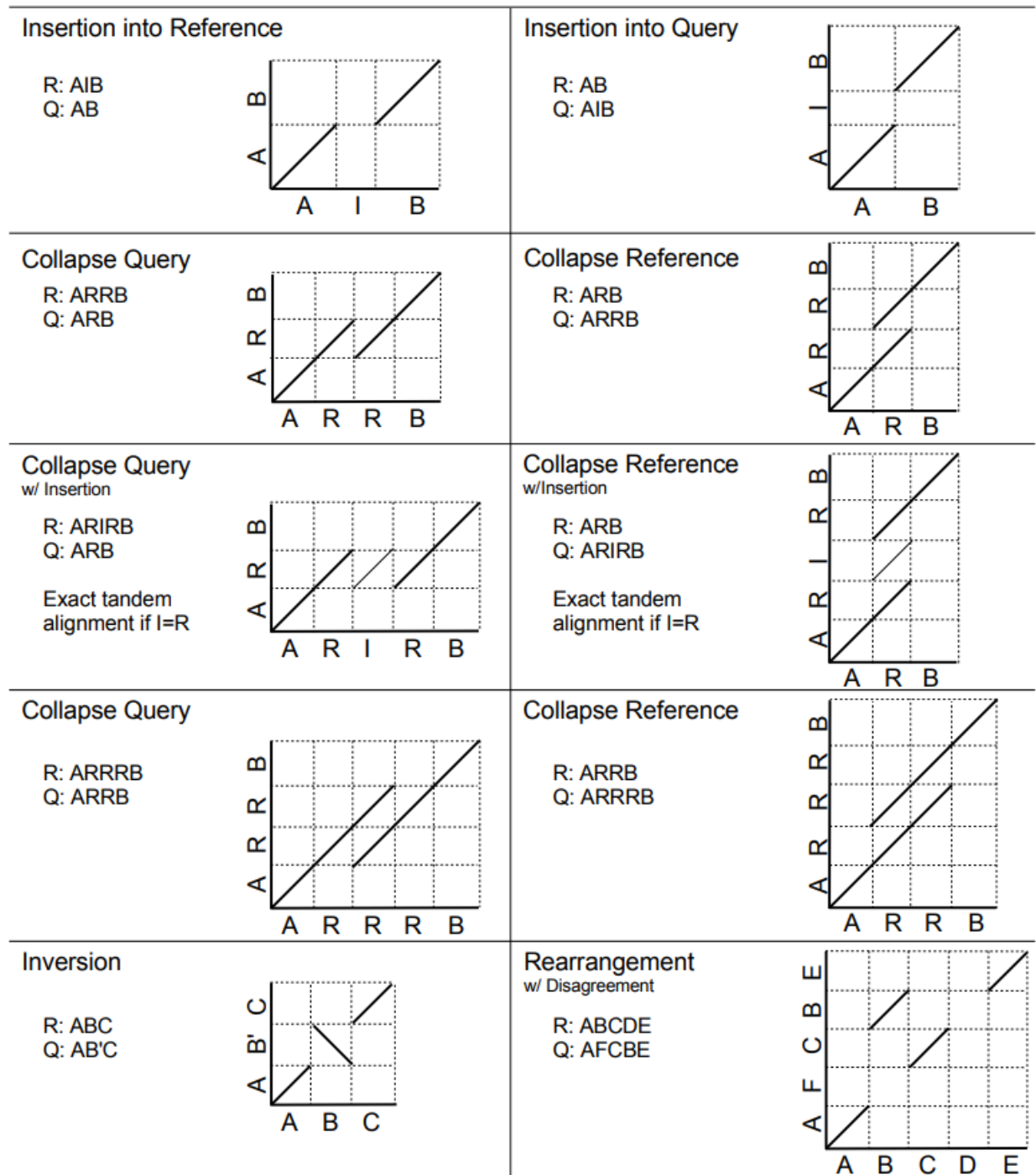
How similar are our two haplotypes? Which haplotype do we want to move forward with for scaffolding with Hi-C? Assemblytics is a nifty and quick way to quickly build dot plots that compare to sequences (or sets of sequences, e.g. in fasta files).

A **dot plot** is a graphical method that allows the comparison of two biological sequences and identify regions of close similarity between them. It is probably the oldest way of comparing two sequences [Maizel and Lenk, 1981].

Dot plot are two dimensional graphs, showing a comparison of two sequences. The principle used to generate the dot plot is: The top X and the left y axes of a rectangular array are used to represent the two sequences to be compared.

Calculation: Matrix - Columns = residues of sequence 1 - Rows = residues of sequence 2.

A dot is plotted at every co-ordinate where there is similarity between the bases.



Michael Schatz mschatz [a t] umiacs.umd.edu

Fig. 28: Image Source: GalaxyProject Training Material

Construct a simple dot plot for

GCTGAA

GCGAA

	G	C	T	G	A	A
G	*			*		
C		*				
T			*			
A					*	
A						*

One sequence goes horizontally, the other vertically
 Mark boxes w/ matched horizontal and vertical symbols
 Look for diagonal(s)

Alignment:

GCTGAA

GCT-AA

What about an example with longer sequences? Plus repeats!

Construct a simple dot plot for
 GCTAGTCAGATCTGACGCTA
 GATGGTCACATCTGCCGC

	G	C	T	A	G	T	C	A	G	A	T	C	T	G	A	C	G	C	T	A
G	*				*				*					*			*			
A				*				*		*					*				*	
T			*			*				*		*						*		*
G	*				*				*					*			*			
G	*				*				*					*			*			
T			*			*				*		*						*		*
C		*				*				*		*				*		*		*
A				*		*		*		*				*		*		*		*
C		*				*			*		*			*		*		*		*
A				*		*		*		*		*		*		*		*		*
T			*		*	*		*		*		*	*		*		*	*	*	*
C		*			*	*		*		*		*	*		*		*	*	*	*
T			*		*	*		*		*		*	*		*		*	*	*	*
G	*				*				*					*			*			
C		*			*				*					*			*		*	*
C		*			*				*					*			*		*	*
G	*				*				*					*			*		*	*
C		*			*				*					*			*		*	*

A long stretch of nearly identical residues is revealed starting at the fifth nucleotide of each sequence (GTCA-ATCTG-CGC).

Simple dot plots get too noisy when comparing every single nucleotide in a string. The solution is to compare windows of strings.

Install MUMMER and run assemblytics, just as the online instructions tell you to.

Instructions

Upload a delta file to analyze alignments of an assembly to another assembly or a reference genome

1. Download and install [MUMmer](#)
2. Align your assembly to a reference genome using nucmer (from MUMmer package)

```
$ nucmer -maxmatch -l 100 -c 500 REFERENCE.fa ASSEMBLY.fa -prefix OUT
```

Consult the [MUMmer manual](#) if you encounter problems

3. Optional: Gzip the delta file to speed up upload (usually 2-4X faster)

```
$ gzip OUT.delta
```

Then use the OUT.delta.gz file for upload.

4. Upload the .delta or delta.gz file ([view example](#)) to Assemblytics

Important: Use only contigs rather than scaffolds from the assembly. This will prevent false positives when the number of Ns in the scaffolded sequence does not match perfectly to the distance in the reference.

The unique sequence length required represents an anchor for determining if a sequence is unique enough to safely call variants from, which is an alternative to the mapping quality filter for read alignment.

Depending on how you installed it, you might run into some problems.

Setting the window size of matches

We use the `-l 100` and `-c 500` options for Assemblytics, per the online manual. Check out the nucmer manual for what these options mean:

MANDATORY:

Reference	Set the input reference multi-FASTA filename
Query	Set the input query multi-FASTA filename

OPTIONS:

--mum	Use anchor matches that are unique in both the reference and query
--mumcand	Same as --mumreference
--mumreference	Use anchor matches that are unique in in the reference but not necessarily unique in the query (default behavior)
--maxmatch	Use all anchor matches regardless of their uniqueness
-b breaklen	Set the distance an alignment extension will attempt to extend poor scoring regions before giving up (default 200)
--[no]banded	Enforce absolute banding of dynamic programming matrix based on diagdiff parameter EXPERIMENTAL (default no)
-c mincluster	Sets the minimum length of a cluster of matches (default 65)
--[no]delta	Toggle the creation of the delta file (default --delta)
--depend	Print the dependency information and exit
-D diagdiff	Set the maximum diagonal difference between two adjacent anchors in a cluster (default 5)
-d diagfactor	Set the maximum diagonal difference between two adjacent anchors in a cluster as a differential fraction of the gap length (default 0.12)
--[no]extend	Toggle the cluster extension step (default --extend)
-f	
--forward	Use only the forward strand of the Query sequences
-g maxgap	Set the maximum gap between two adjacent matches in a cluster (default 90)
-h	
--help	Display help information and exit
-l minmatch	Set the minimum length of a single match (default 20)
-o	
--coords	Automatically generate the original NUCmer1.1 coords output file using the 'show-coords' program
--[no]optimize	Toggle alignment score optimization, i.e. if an alignment extension reaches the end of a sequence, it will backtrack to optimize the alignment score instead of terminating the alignment at the end of the sequence (default --optimize)
-p prefix	Set the prefix of the output files (default "out")
-r	
--reverse	Use only the reverse complement of the Query sequences
--[no]simplify	Simplify alignments by removing shadowed clusters. Turn this option off if aligning a sequence to itself to look for repeats (default --simplify)
-V	
--version	Display the version information and exit

-l 100 means that a minimum match between two sequence strings must be at least 100 nucleotides. -c 500 means that we must have several overlapping matches that equal at least 500 nucleotides. Only alignments matching these two parameters will be output. This filters out quite a bit of noise, especially in our case, since the two haplotypes should be *fairly* similar (~1.5% heterozygous).

1.5.3 Lesson 3: Scaffolding Algorithms

3.3 Instructions

3.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [Genome Assembly and Haplotyping with Hi-C](#)

3.3 Lesson

Learning Objectives

Vocabulary

Learning Material

3.3 Lab Exercises

Overview

In this lab, we will learn the basics of Hi-C, and how to QC your data, and scaffold a genome.

We will do two major things in this lab:

- Discuss the basics of Hi-C
- QC your Hi-C data to assess the quality of the long-range links
- Align Hi-C data to your haplotype-resolved assembly
- Explore the contact matrix to resolve assembly issues
- Scaffold your haplotype assembly into chromosome pseudomolecules

Be for real, don't be a stranger

—Spice Girls

Task A

Step 1: What is Hi-C sequencing?

How do we take assembled contigs and order/orient them into chromosomes? Hi-C is a powerful technique that captures interactions of chromatin.

DNA is balled up in a cell, packed and positioned in a somewhat predictable way based on interactions with nucleosomes that wrap DNA. There is order, to a degree.

DNA is wrapped around **nucleosomes**. Sometimes, DNA can form predictable **loops**, where distant pieces of DNA interact. Enhancers that increase gene expression from a distance are sometimes found here. **TADs** (topologically-associated domains) are regions of DNA that show a higher-than-expected level of chromatin interaction. That is, they form associations more often than expected compared to the genome average. Different regions of chromosomes, or

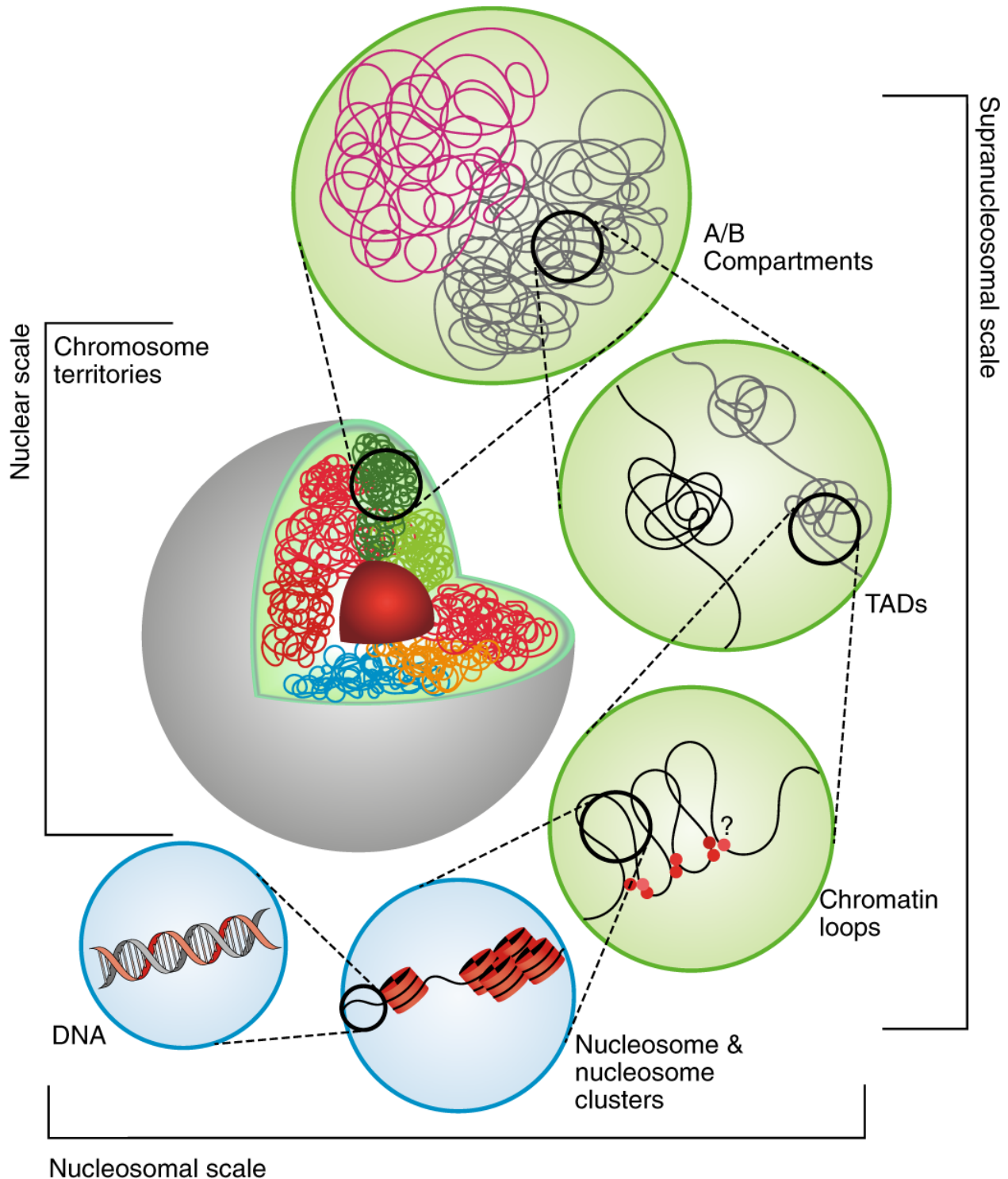


Fig. 29: Image Source: Doğan, E.S., Liu, C. Three-dimensional chromatin packing and positioning of plant genomes. *Nature Plants* 4, 521–529 (2018). <https://doi.org/10.1038/s41477-018-0199-5>

different chromosomes altogether, can form different **compartments** in a nucleus of transcriptionally active (A) and repressed (B) regions. That is, there is a hidden layer of organization within a nucleus that Hi-C can uncover.

We can cross-link DNA that is in close proximity, creating covalent bonds that “attach” two DNA strands together. Usually, these two cross-linked regions are very closely linked on the same chromosome, with little physical distance between them on the linear length of a chromosome. Sometimes, these two cross-linked strands can be from physically distant parts of the same chromosome. Typically, individual homologous chromosomes tend to occupy distinct territories inside a nucleus, meaning that Hi-C can be used to phase DNA haplotypes in a genome assembly. In other words, since we assembled two fairly distinct haplotypes in our assembly, we should be able to extend some of those haplotype blocks using Hi-C.

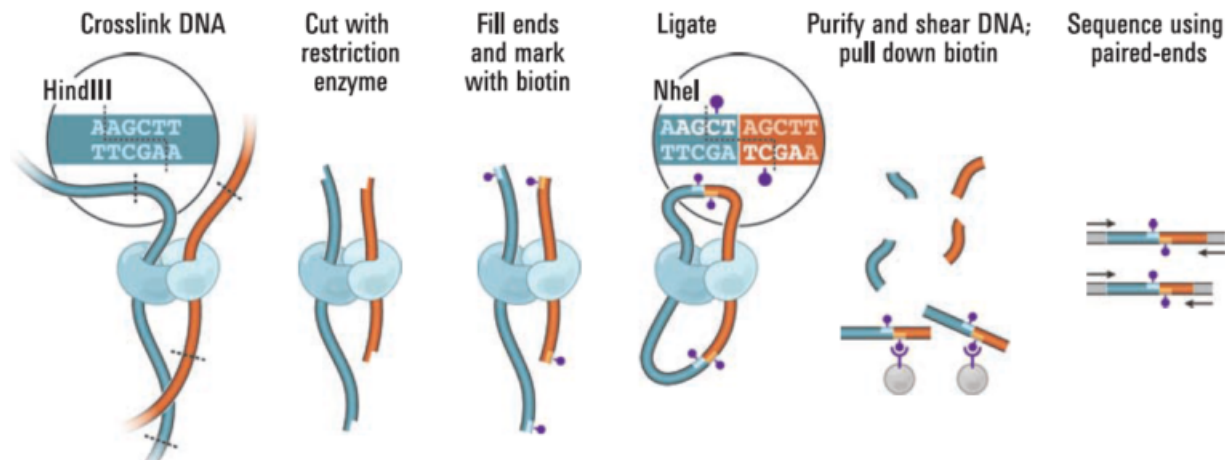


Fig. 30: Image Source: [Nucleus Biotech Website](#)

We can use these short-range and long-range interactions within homologous chromosomes to scaffold our genome assembly into full-length chromosomes, and to phase out the haplotypes.

(with hifiasm, we did both of these things at the same time using the hi-c integrated assembly option)

We can count up the number of interactions between DNA regions from the Hi-C sequencing, and build a giant matrix of all those counts. We expect to find a large number of interactions between DNA that is close together on the same chromosome, which leads to an overwhelmingly high signal along the diagonal of a dot plot. But, we also want these longer-range interactions that allow us to link together contigs that are distant.

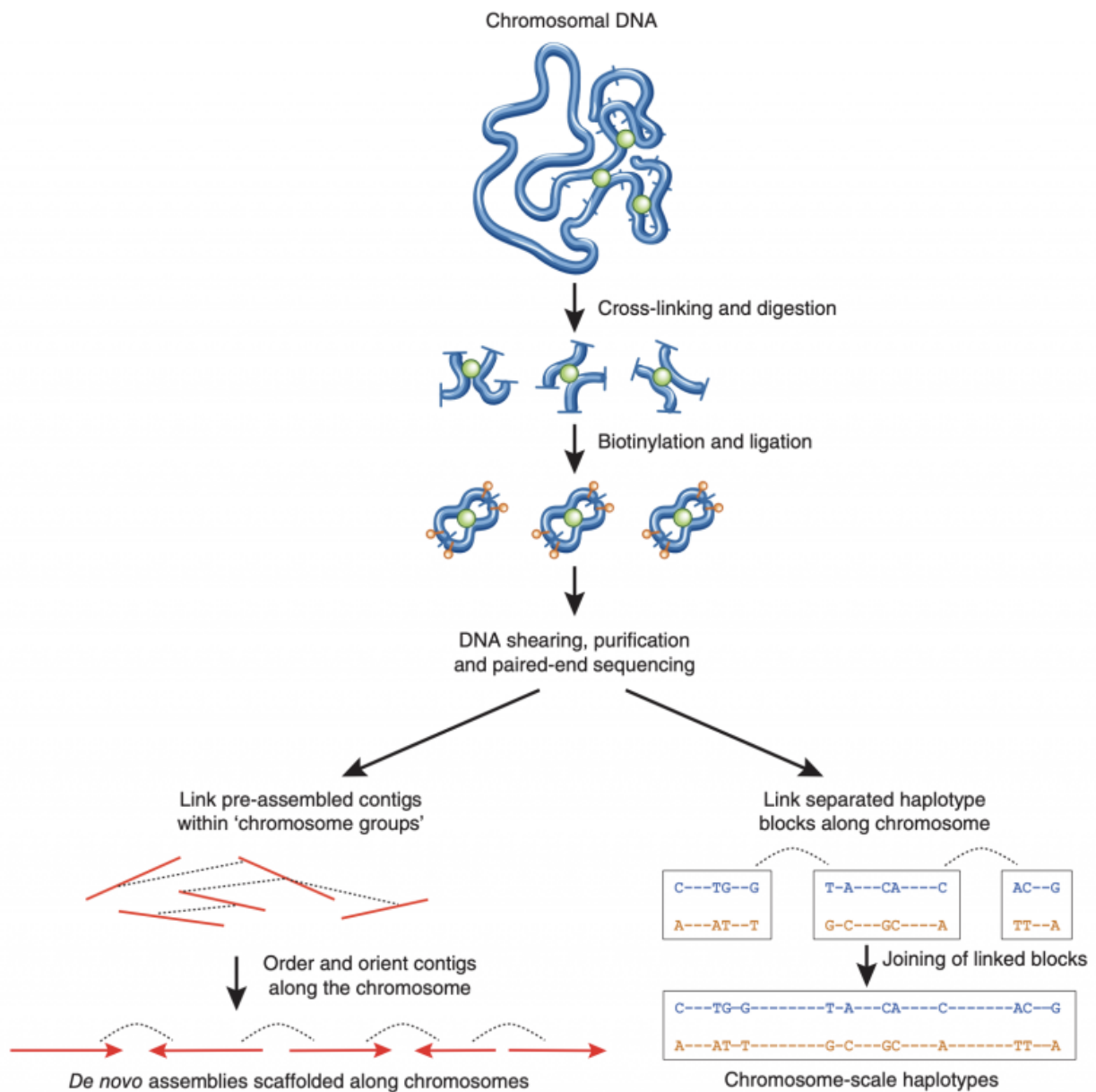
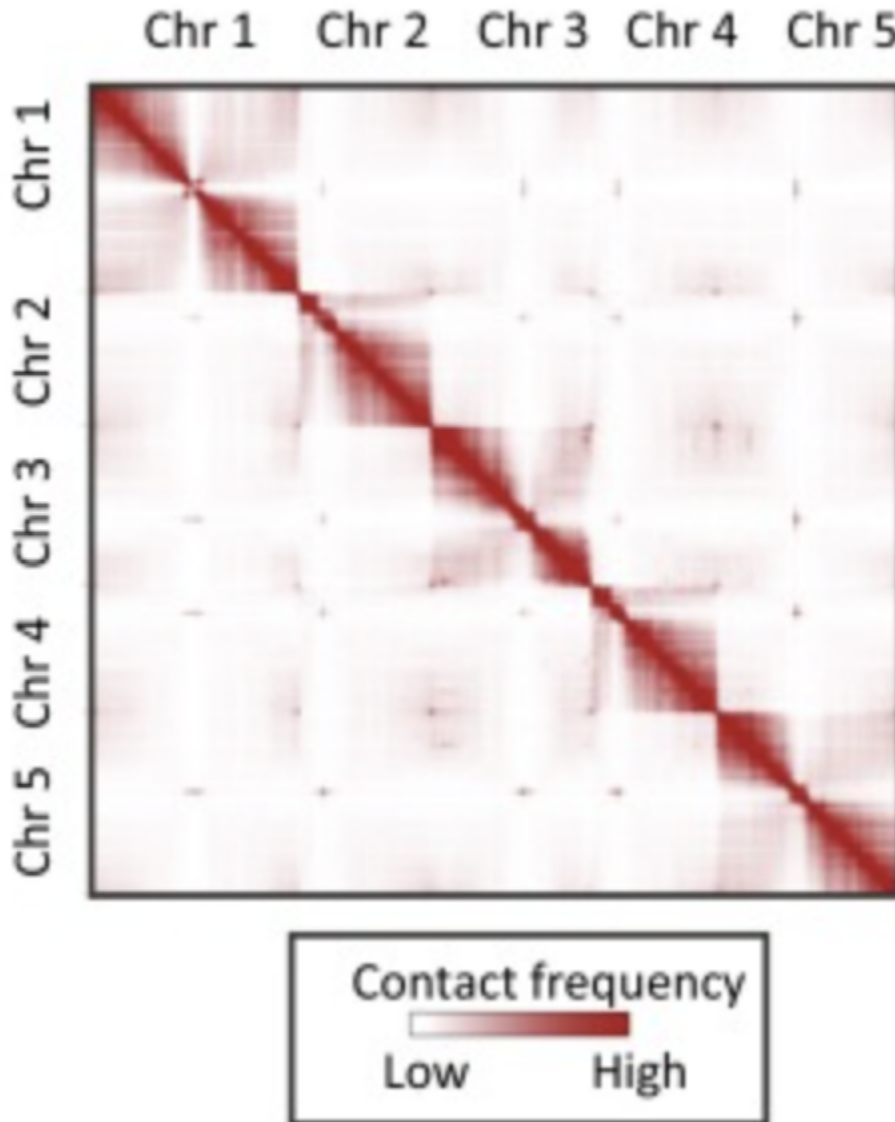


Fig. 31: Image Source: Korbelt, J., Lee, C. Genome assembly and haplotyping with Hi-C. Nat Biotechnol 31, 1099–1101 (2013). <https://doi.org/10.1038/nbt.2764>



Step 2: QC our Hi-C data

[Phase Genomics](#) is a popular company that does fee-for-service (contract) work on genome assembly and metagenomic analyses. They provide kits where you can generate your own Hi-C data from your individual and do all the informatics yourself, or you can send them frozen leaf material and they will do 100% of the work (library prep, sequencing, and informatics/scaffolding).

First, we want to check to see how good our Hi-C data is. That is, are there short-range AND long-range interactions occurring, that will help us scaffold our genome? Phase Genomics provides a helpful QC pipeline and scripts that can quickly tell us the quality of our data, given only a few million reads, and a reference genome of something very closely related or a draft genome.

From Phase Genomics:

“The best way to know if a Hi-C library worked is to look at how much long-range signal is in it. There are also several metrics which correlate with a suspicious library, such as a high number of PCR duplicates or a large number of reads

which align to the same position in the genome (this happens when there are very short fragments in the library due for example to two restriction sites being very close together). Our QC script measures these quantities and makes a recommendation about the library based on the result.”

<https://phasegenomics.github.io/2019/09/19/hic-alignment-and-qc.html>

You should have bwa and samtools installed already, but if not, use Conda to install them. I have left a 1 million read subset of the Hi-C data in */scratch*.

Let’s run it on hap1 (*/scratch/hifiasm.hic.gfa.hic.hap1.p_ctg.fasta*). The R1 and R2 files start with *toomers.omni-c.2M.subset.**

Hint: You can also speed up bwa by multi-threading it...

Then, follow the instructions for installing Phase Genomics’ hic_qc program with Conda: https://github.com/phasegenomics/hic_qc

1.5.4 Lesson 4: Interacting with Hi-C Maps

3.4 Instructions

3.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

3.4 Lesson

Learning Objectives

Vocabulary

Learning Material

3.4 Lab Exercises

Overview

In this lab, we will learn the basics of interacting with Juicebox and .hic files to manually correct errors with automated Hi-C scaffolding.

We will do two major things in this lab:

- Learn the basics of Juicebox
- Interact with Human ENCODE maps
- Load the Toomer’s haplotype1 and haplotype2 Hi-C maps

Be for real, don’t be a stranger

—Spice Girls

Task A

Step 1: Working with Hi-C maps

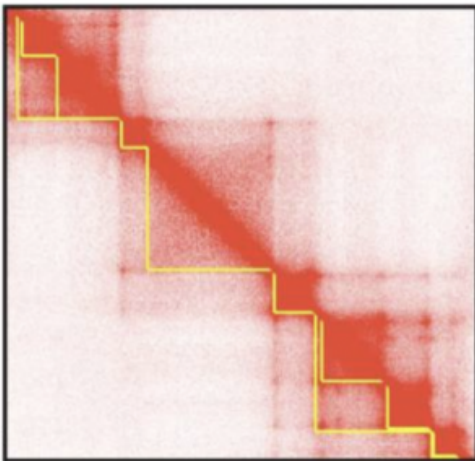
How do we interact with these Hi-C maps, that often contain hundreds of millions, or billions, of chromatin-chromatin interactions?

[Juicebox](#) is an excellent set of tools to interact with these maps. Juicebox allows you to visually investigate the contact matrix of all chromatin-chromatin interactions. This contact matrix can be produced using several programs that take raw Hi-C reads and map them to a reference genome, such as Juicer and HiC-Pro. There are additional packages that can attempt to scaffold a reference genome, as we want to do here. These include SALSA, Juicer/3D-DNA, INSTAGRAAL, and more.

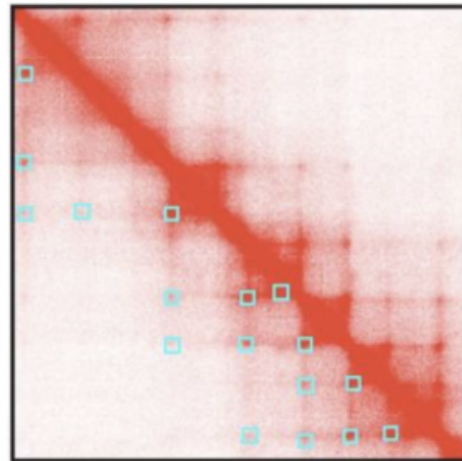
There is also a [desktop version of Juicebox](#), which is the one I use the most.

There are additional suites of software that interact with these contact matrices to call **features** in the data, such as long-range chromatin loops, and short-distance contact domains. Arrowhead can call these contact domains (TADs) that interact with each other more than expected by chance, and HICCUPS can call these long-range loops (highlighted in blue squares).

Arrowhead
contact domains



HICCUPS
loops



The best way to learn is to dive in. Today we'll focus on using Juicebox, an interactive way to work with Hi-C maps and explore them. First, watch this high-level video from the Juicebox developers:

[Juicebox web](#) is a cloud-based web app that allows us to look at and share Hi-C maps with each other. Open up Juicebox web and poke around a little bit before moving on.

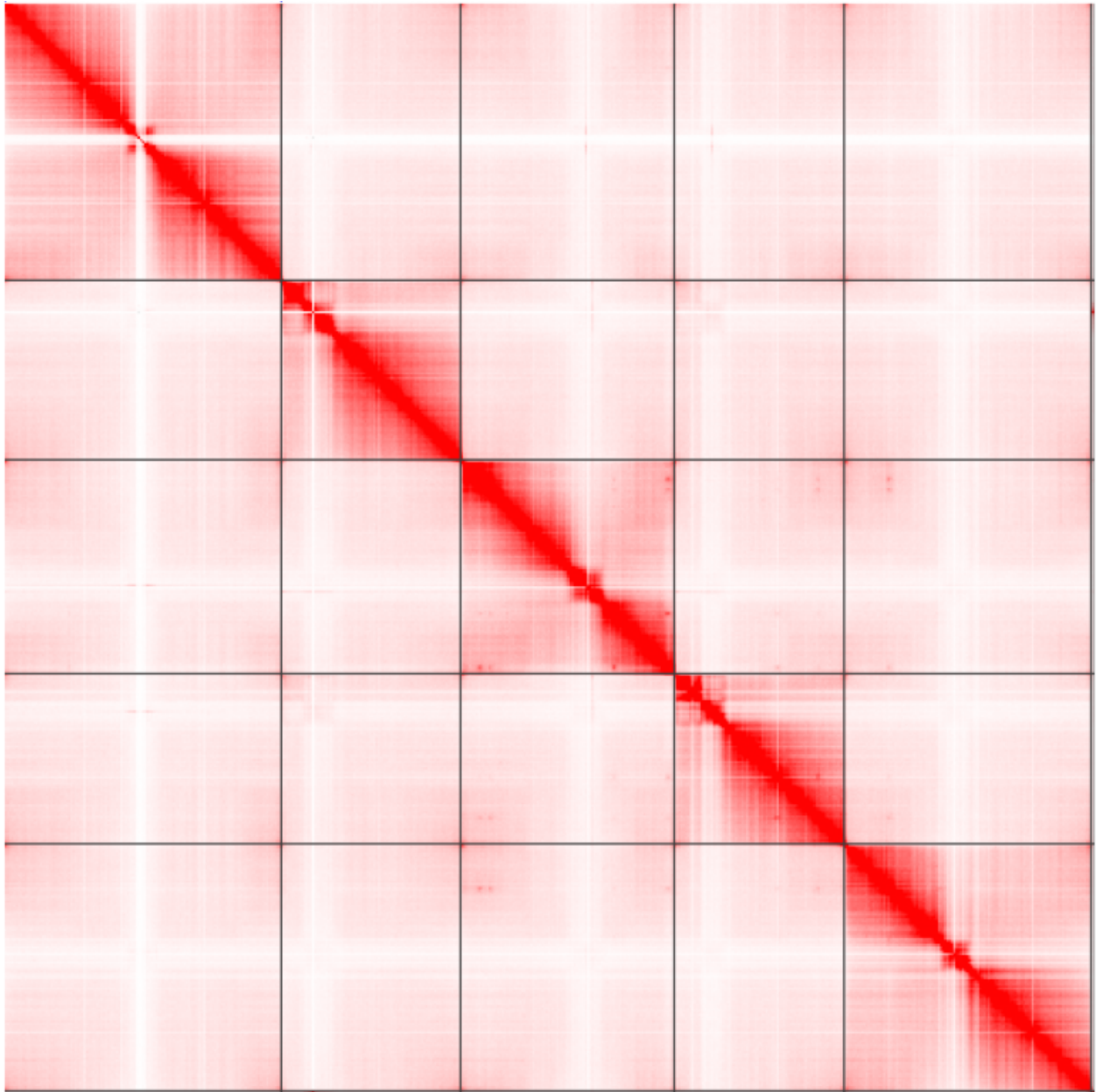
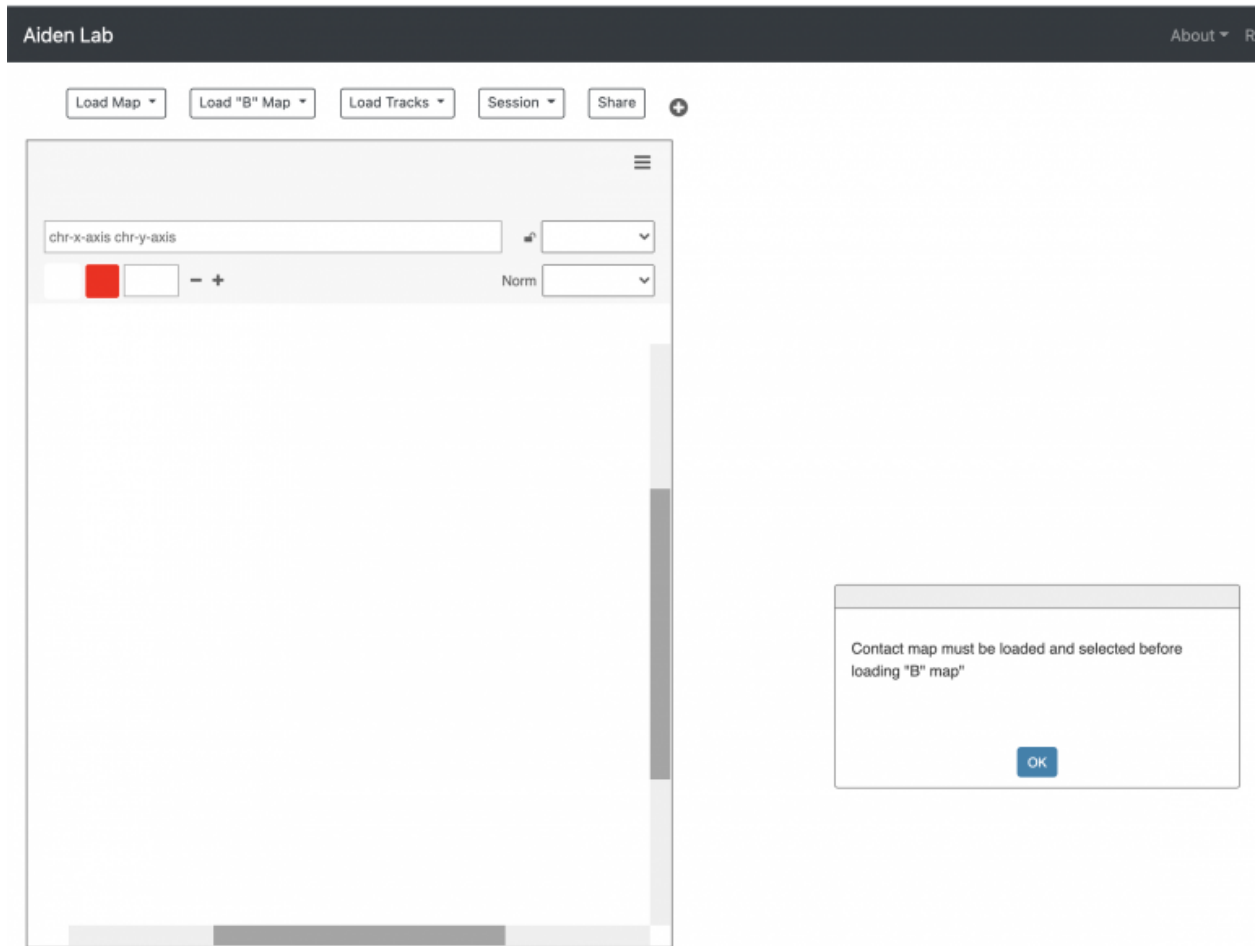


Fig. 32: An example contact matrix from *Arabidopsis thaliana*. Notice how there are 5 obvious chromosomes?

Step 2: Load a Hi-C contact matrix

On the Juicebox web app, you should see something like this:



Juicebox web allows you to look at two Hi-C contact matrices at the same time, which will be nifty for us soon. But for now, go to “Load Map -> Juicebox Archive”. Juicebox has a lot of hi-c maps from diverse organisms already loaded. Nifty! Is your organism of choice in there?

For instance, Search for “Arabidopsis”, and you should see two experiments:

Contact Map ✕									
Show 10 entries		Search: arabido ✕							
name	author	journal	year	organism	reference genome	cell type	experiment type	protocol	
A. thaliana Wang2015,Liu2016	Rowley and Nichols et al.	Molecular Cell	2017	Arabidopsis thaliana	-	-	Hi-C	-	
Arabidopsis	Xie et al.	Molecular Plant	2015	-	-	-	-	-	

Click the Rowley and Nichols et al. paper, and select “OK” at the bottom right of the window to load the map. Even without annotations loaded, can you see 5 chromosomes?

Task B

Next, let's follow the video + hands-on tutorial for exploring Human ENCODE Hi-C data. By the end of these 4 tutorial videos, you should know how to 1) explore raw HiC maps, 2) visualize two maps at the same time, 3) Load annotations, 4) share your map.

Loading maps

Loading Annotations

Comparing two maps

Sharing maps

Mastering Content

Now that you know how to use Juicebox.js to explore two maps at the same time, this is the ideal scenario for us to explore our two haplotype .hic contact matrices at once. I used SALSA to map our raw Hi-C data to each of the two haplotypes for scaffolding into chromosomes.

The .hic format data will be available in /scratch ASAP (it is still running!)

Anyone like Tetris? Here's an example of how someone uses Hi-C maps to fix a genome, and order/orient contigs into chromosomes —

We'll be doing this with toomers!

Finding mis-assemblies

Now comes the hard part: How do we find and correct mis-assemblies in the Hi-C data to produce our **final haplotype assemblies**.

It turns out that SALSA2 performed poorly on our data. I ran a more intensive,
but usually more accurate, scaffolding program called Juicer/3D-DNA.

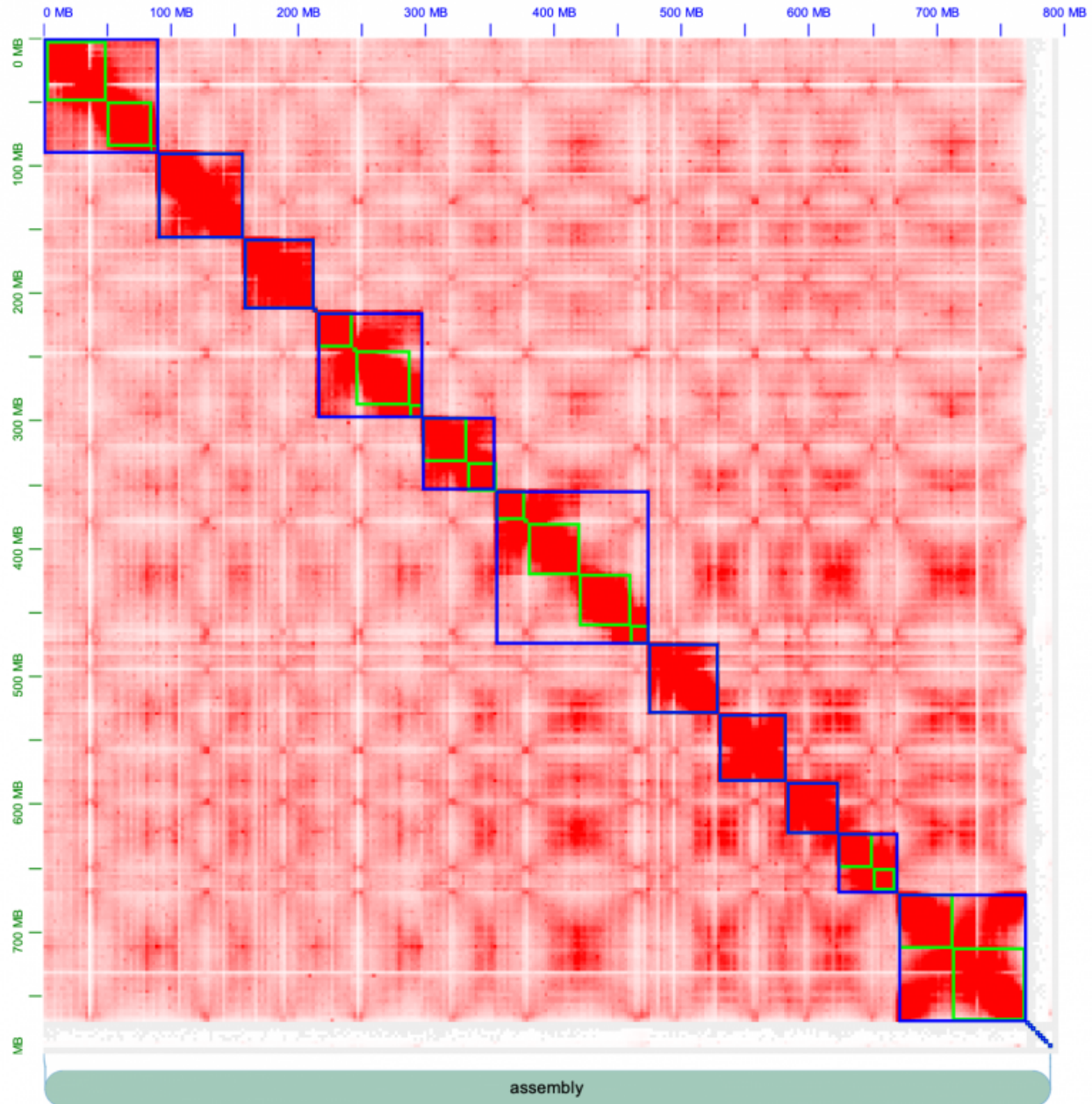
I've left the raw results in */scratch/hic-scaff/*

There are two files per haplotype:

- .hic map: `hifiasm.hic.gfa.hic.hap1.p_ctg.rawchrom.hic`
- .assembly file describing raw chromosomes: `hifiasm.hic.gfa.hic.hap1.p_ctg.rawchrom.assembly`

Download these two files per haplotype to your laptop, and load these maps into Juicebox — the Desktop version.

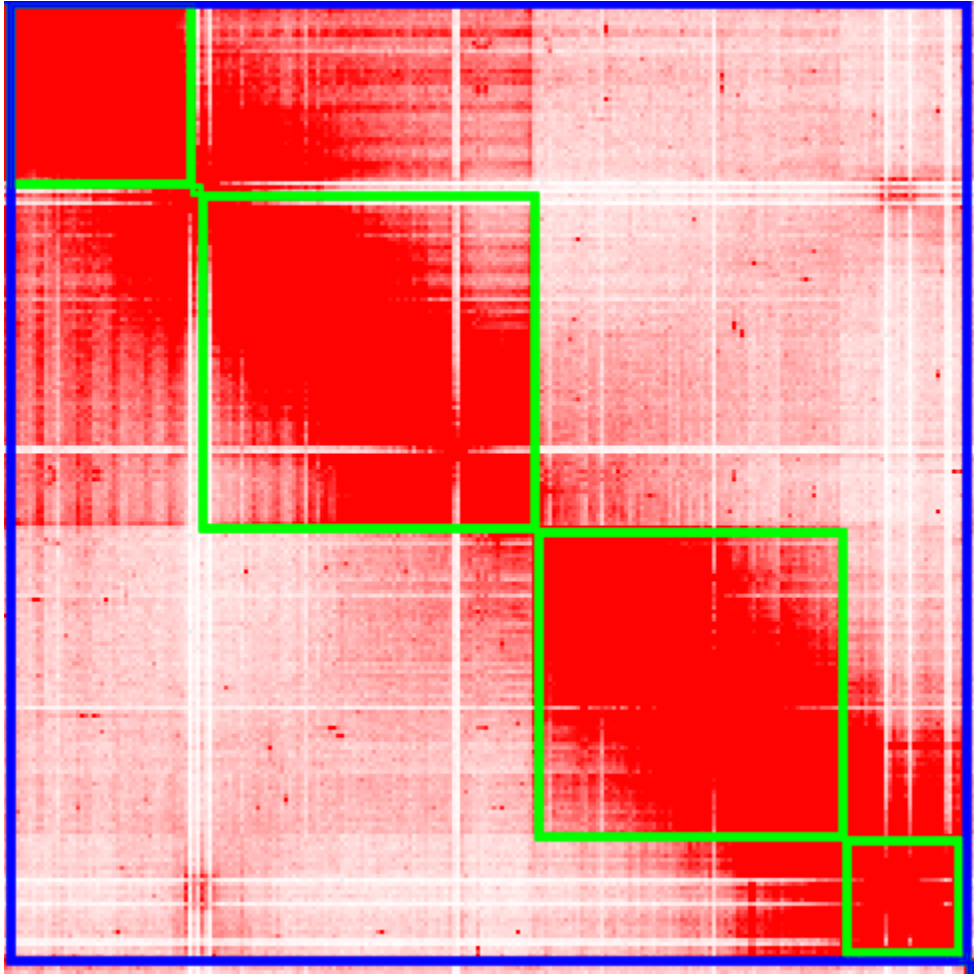
Load the .hic map for haplotype1 using File->Open. Then load the .assembly file using Assembly->Import Map Assembly. You should see a contact map that looks like this:



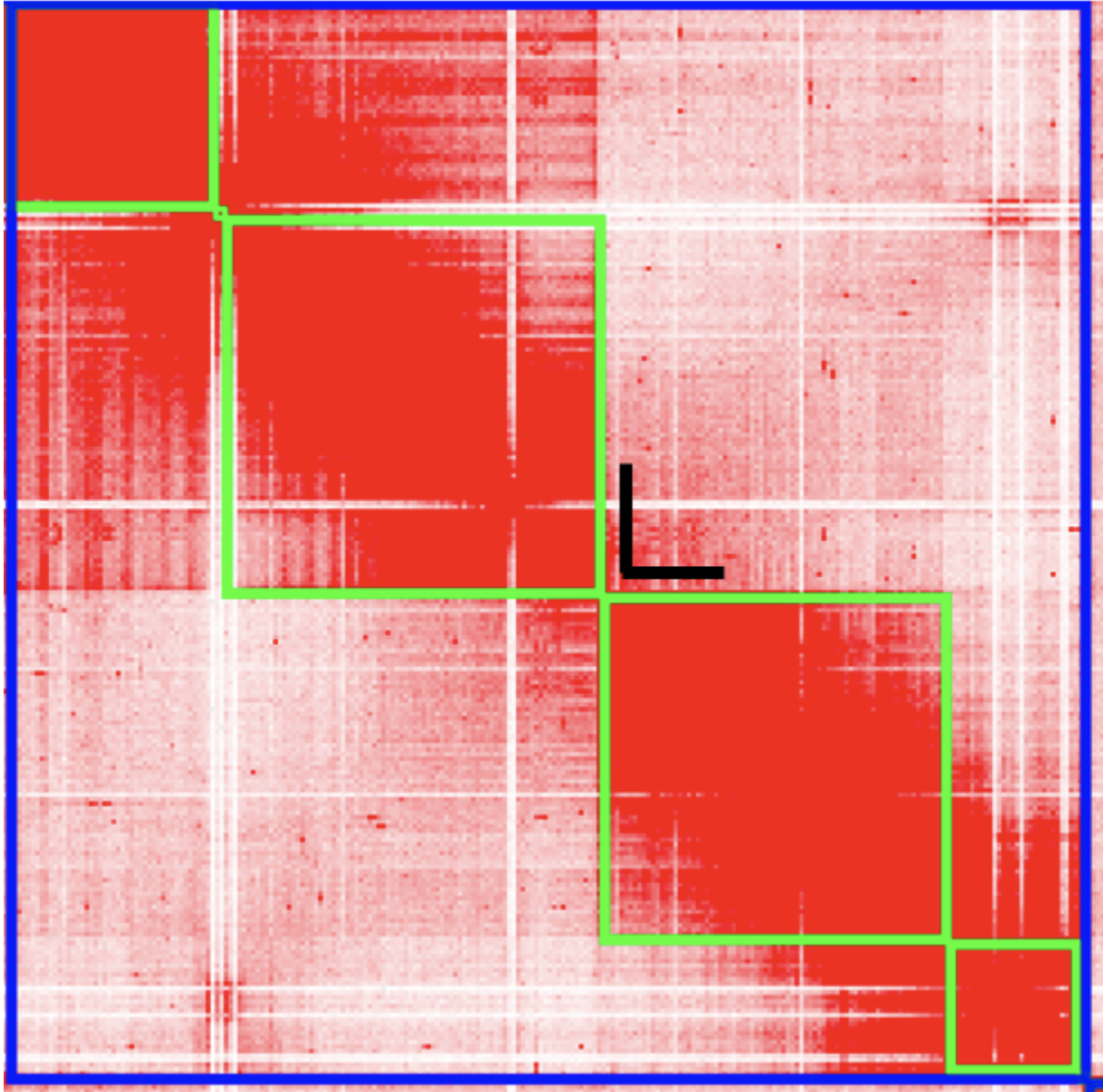
Chromosomes are outlined in blue, contigs outline in green. There are strong inter-chromosome interactions, or in other words, the chromosomes form strong square blocks. Fiddle around with the knobs on Juicebox to get acquainted; change the **normalization** (e.g. I often use “Balanced” normalization), and pull the slider on “Color Range” all the way to the maximum. The contact map looks pretty excellent, for the most part: there are strong, within-chromosome interactions, that appear as a strong diagonal line.

For a quick primer on how to manually edit genomes, watch this Aiden lab video from Olga, who wrote Juicebox. Afterwards, I’ll walk you through one of these manual edits to break a chromosome.

Juicer thinks there are 11 chromosomes, and that’s not right. Do you see where we should make the break, and split a chromosome into two? Zoom into the 6th chromosome by double-clicking it.

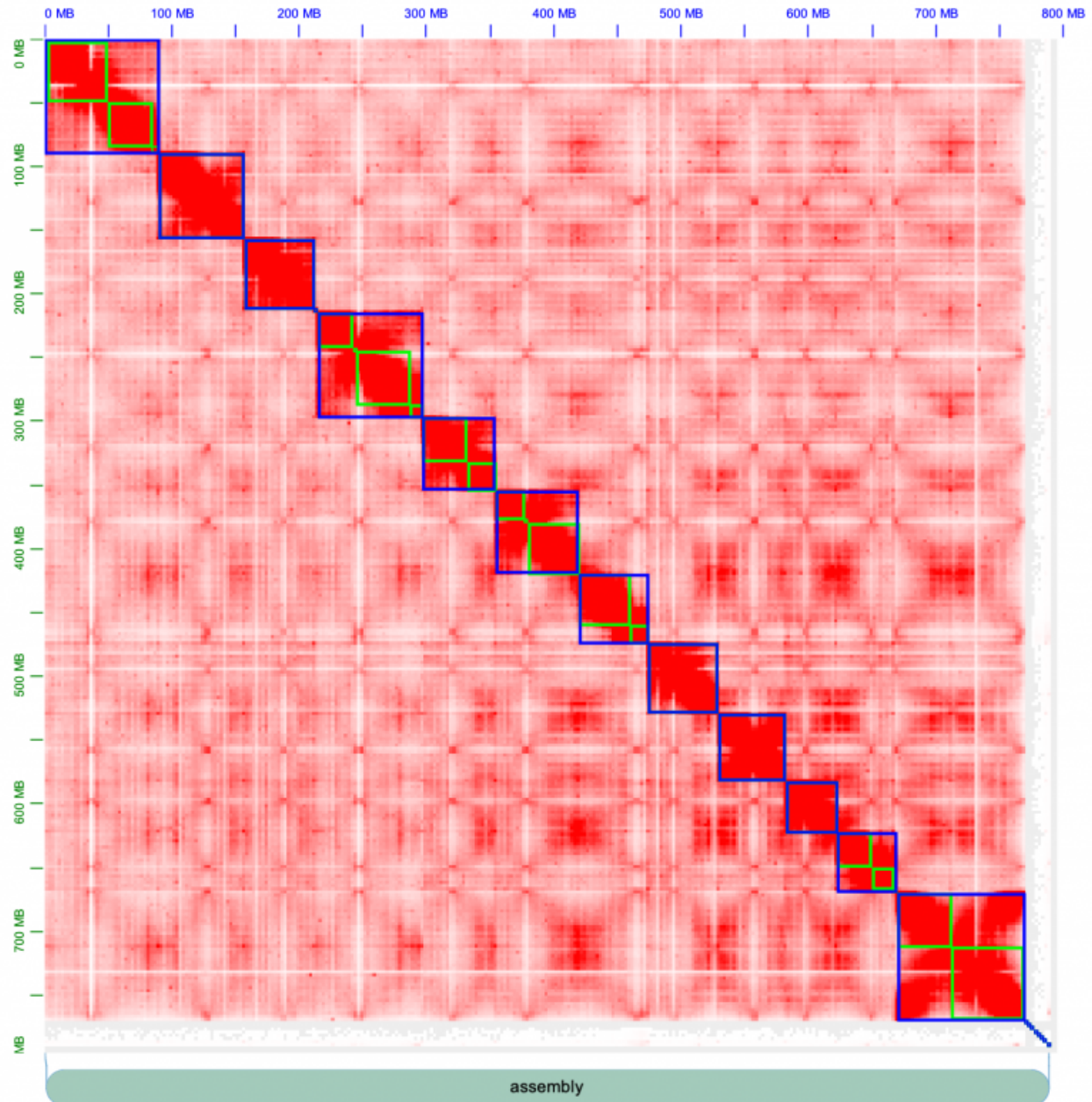


To create a break in the chromosomes, drag your mouse close to the gap and you'll see a right angle appear:

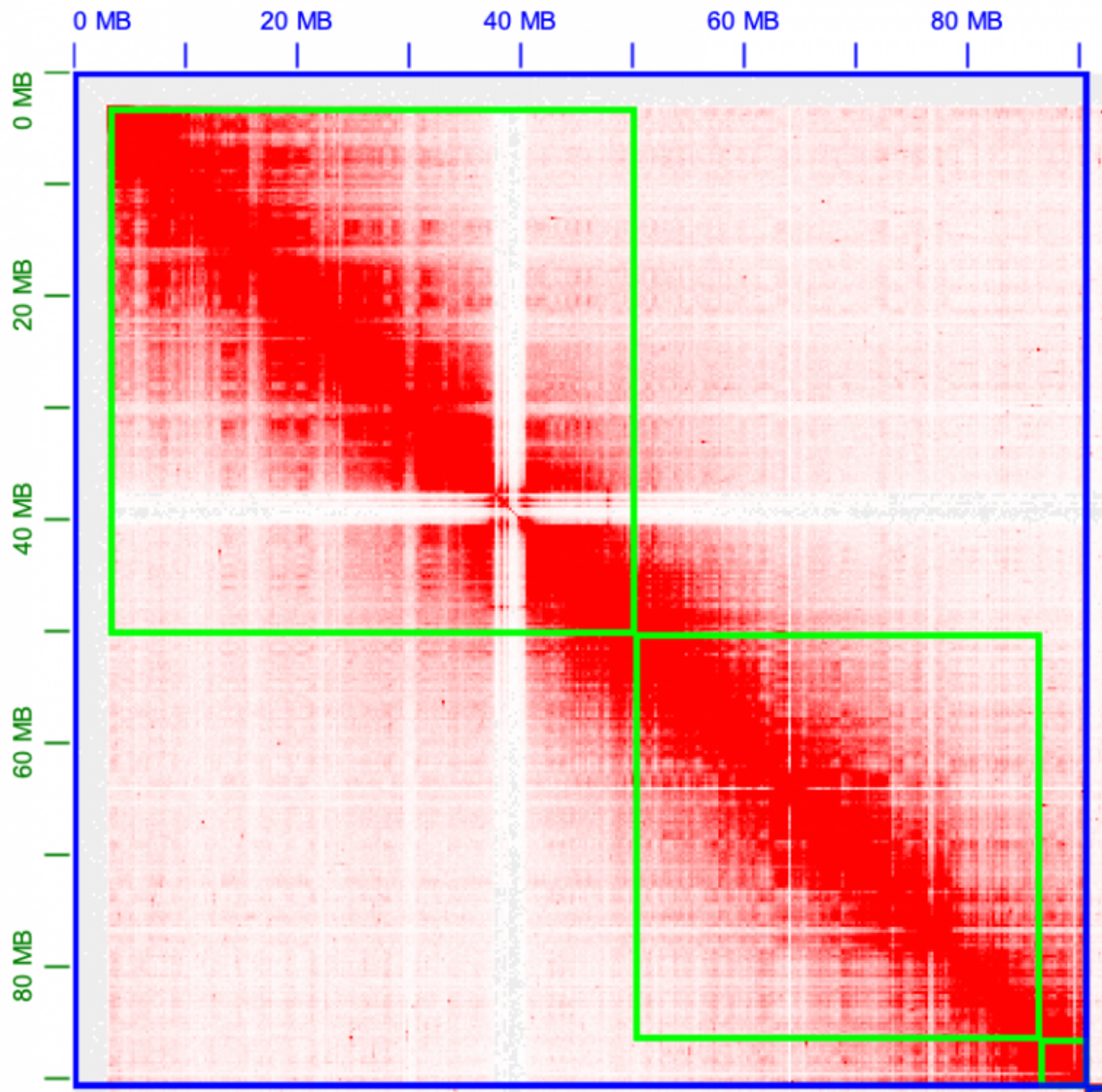


Click your mouse at that right-angle gap, and you'll see the chromosome split into 2.

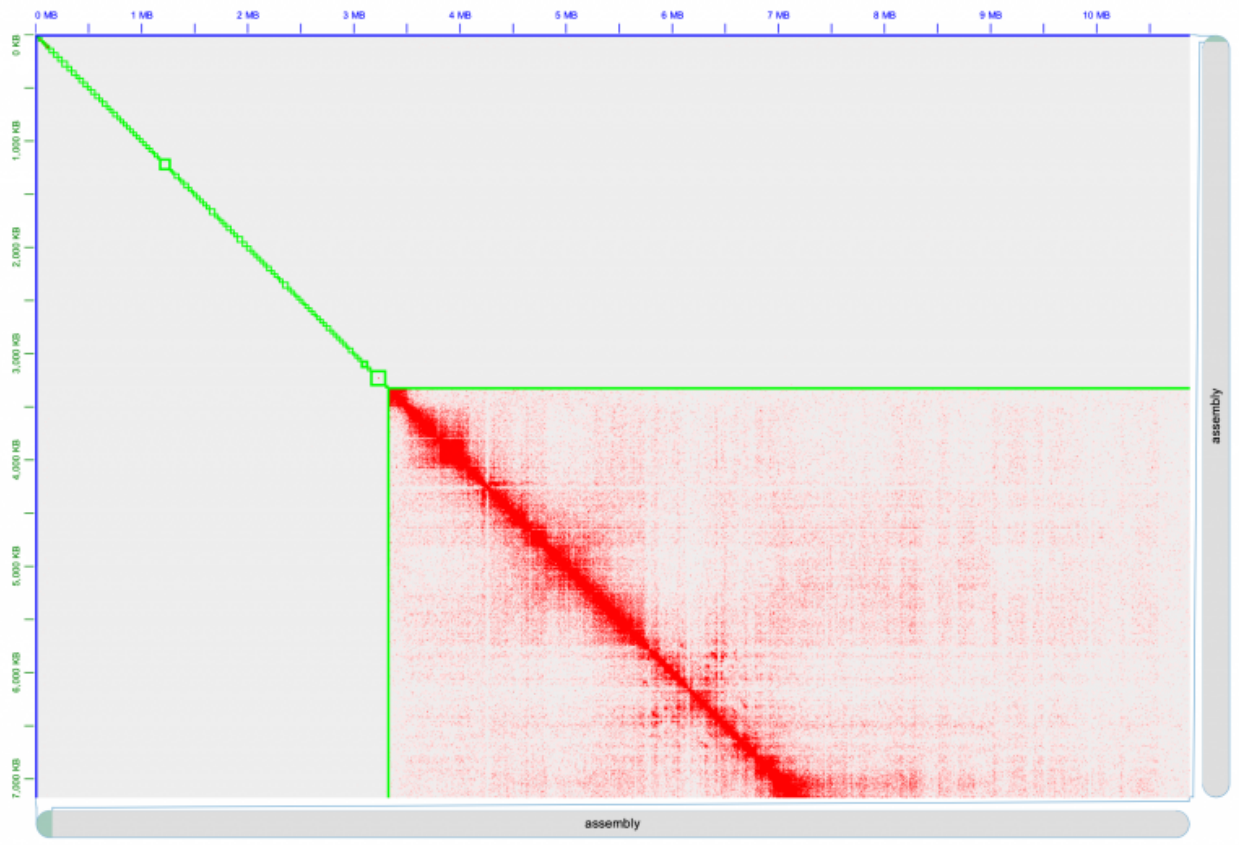
Now it's looking better! We have 12 chromosomes.



Now we can zoom in fine-scale and look at the contigs more deeply. There's always more than meets the eye. Zoom in one chromosome 1. There is a lot of trash in this contig, at the very beginning of the assembly, in the very top left corner. Zoom in more!



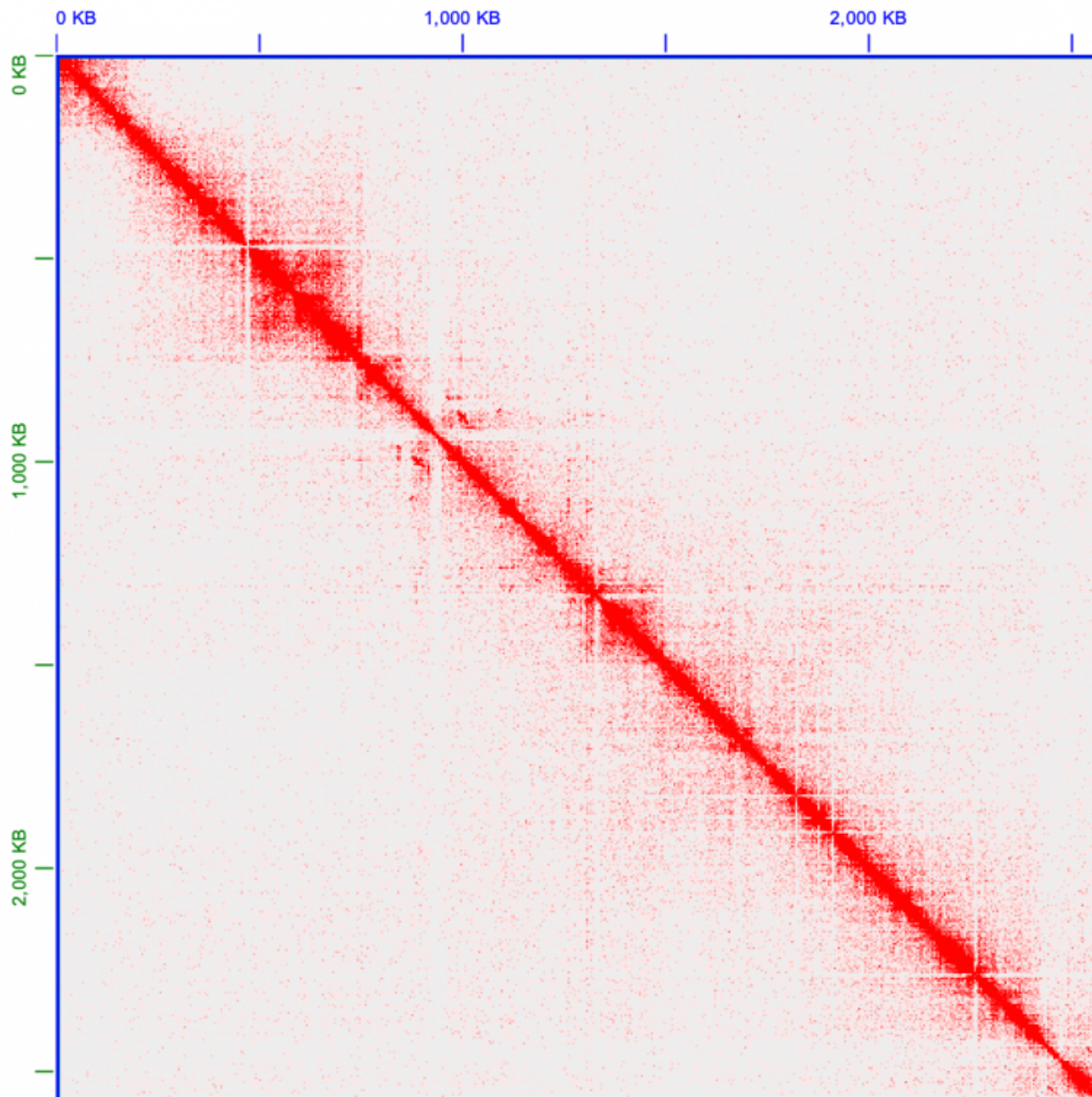
All of these little tiny contigs need to be moved to the trash, or “debris” as it’s called in Juicebox.



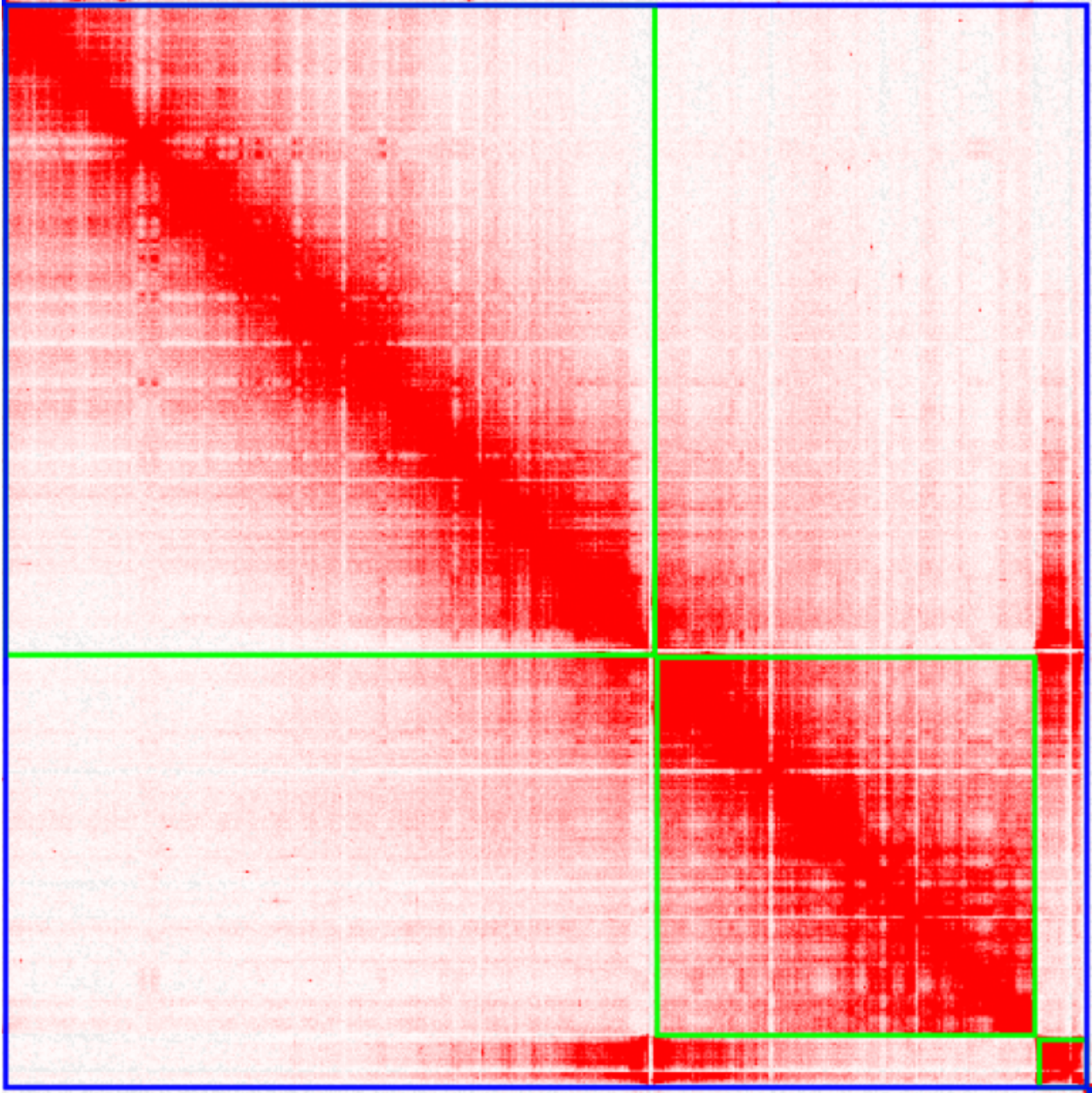
Hold shift and then drag your mouse to include ALL of these tiny little contigs. They'll turn black, and be surrounded by a faint yellow box. Right click one of the boxes, and select "Move to debris". Voila.



And just like that, you've made your first chromosome edit! It should look like this now.



Make your way through every chromosome, and just like Olga does in her instructional video, find mis-assemblies where the chromosomes look incorrectly placed. Here's an example on chromosome 11, that looks very much like Olga's example in her Youtube video: Can you fix chromosome 11? That piece at the end looks like it's in the wrong place...



Make your way through every chromosome. Create manual edits where necessary. To save your edits, use Assembly->Export Assembly.

Before class on Wednesday, create a folder in our shared google drive and leave your edits for both haplotypes in the drive.

1.5.5 Lesson 5: Assessing Completeness

3.5 Instructions

3.5 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs](#)
- [Mercury: reference-free quality, completeness, and phasing assessment for genome assemblies](#)

3.5 Lesson

Learning Objectives

Vocabulary

Learning Material

3.5 Lab Exercises

In this lab, we will learn the basics of assessing the completeness of a draft assembly.

We will do two major things in this lab:

- Run assemblytics to compare haplotype1 and haplotype2 of our phased assembly, to identify any major issues
- Run BUSCO to identify core, conserved genes in the assembly, and objectively assess its quality

“Be for real, don’t be a stranger” - Spice Girls

Task A

There are several ways that we can objectively measure the quality of our genome assembly. One of these methods is by searching for super-conserved, core eukaryotic genes. There are genes that exist in nearly all plant genomes that have very conserved functions, sometimes called “housekeeping genes”, that tend to 1) evolve slowly, 2) maintain their gene structure over time, 3) be present in low or single copy.

[BUSCO](#) is a software tool that has built databases of these conserved, core eukaryotic genes for different lineages on the tree of life. For plants, a variety of databases exist, including e.g. all Viridiplantae, or focusing more finely in one some lineages, e.g. Solanaceae family (potato, tomato).

Just like the manual tells us, install BUSCO with Conda

```
conda install -c conda-forge -c bioconda busco=5.2.2
```

However, the install will fail, and tell you that BUSCO needs python3. Oops. It even tells us this in the manual!

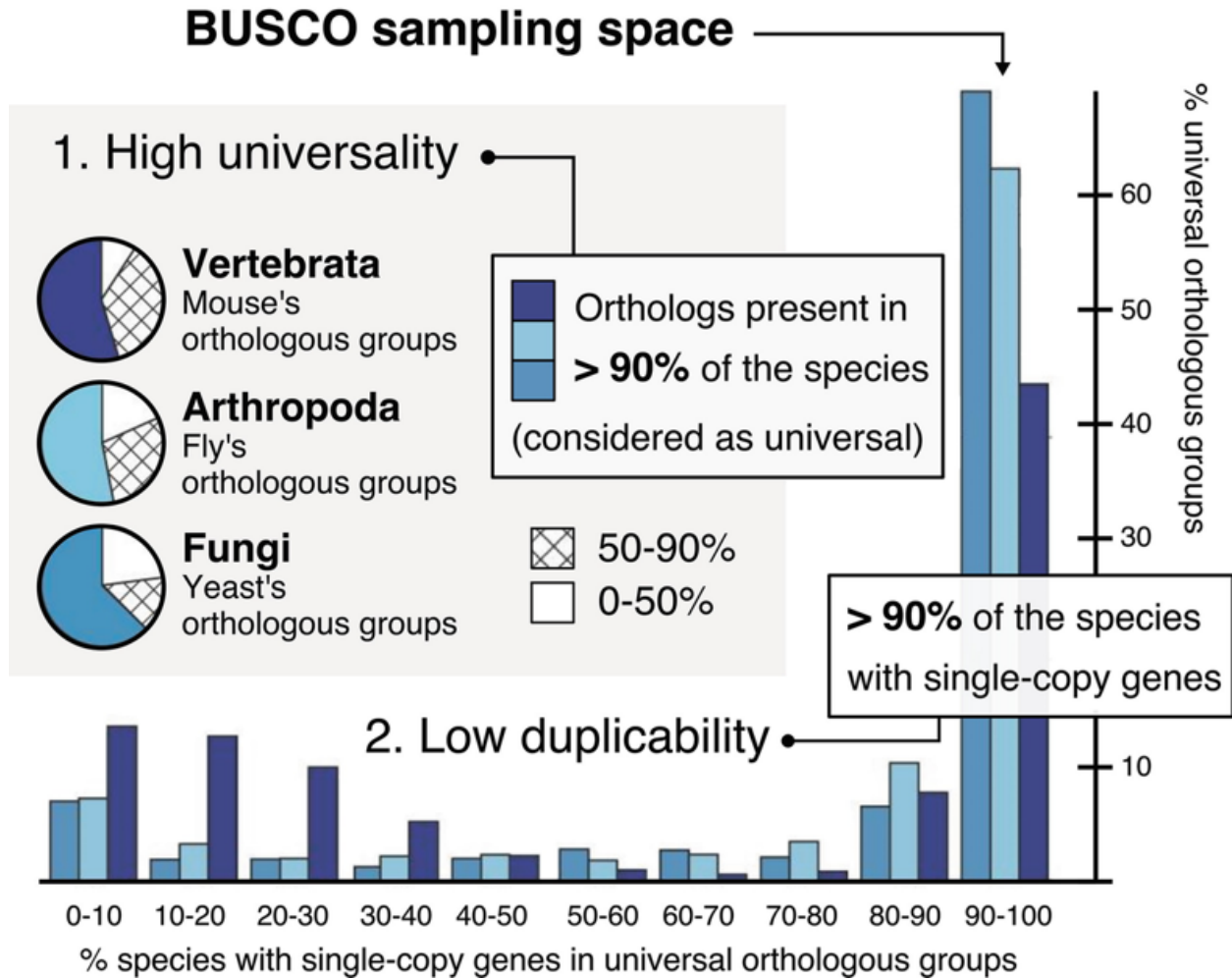


Fig. 33: Image Source: BUSCO Website

Third-party components

A full installation of BUSCO requires Python 3.3+ (2.7 is not supported from v4 onwards), BioPython, pandas, tBLASTn 2.2+, Augustus 3.2, Prodigal, Metaeuk, HMMER3.1+, SEPP, and R + ggplot2 for the plotting companion script. Some of these tools are necessary only for analysing certain type of organisms and input data, or for specific run modes.

Luckily you've already made a conda environment that runs python3. Activate it:

```
conda activate py3.7
```

Then reinstall.

Next, follow the BUSCO user guide. First, explore the possible databases.

```
busco --list-datasets
```

Most of these databases are not useful to you. There are just a handful of plant databases:

```
- viridiplantae_odb10
  - chlorophyta_odb10
  - embryophyta_odb10
    - liliopsida_odb10
      - poales_odb10
    - eudicots_odb10
      - brassicales_odb10
      - fabales_odb10
      - solanales_odb10
```

We'll just use "eudicots_odb10".

Then check out the BUSCO options, using the -h flag

```
-i FASTA FILE, --in FASTA FILE
    Input sequence file in FASTA format. Can be an assembled genome or
    ↳transcriptome (DNA), or protein sequences from an annotated gene set.
-o OUTPUT, --out OUTPUT
    Give your analysis run a recognisable short name. Output folders
    ↳and files will be labelled with this name. WARNING: do not provide a path
-m MODE, --mode MODE
    Specify which BUSCO analysis mode to run.
    There are three valid modes:
    - geno or genome, for genome assemblies (DNA)
    - tran or transcriptome, for transcriptome assemblies (DNA)
    - prot or proteins, for annotated gene sets (protein)
-l LINEAGE, --lineage_dataset LINEAGE
    Specify the name of the BUSCO lineage to be used.
--auto-lineage
    Run auto-lineage to find optimum lineage path
```

(continues on next page)

(continued from previous page)

```

--auto-lineage-prok  Run auto-lineage just on non-eukaryote trees to find optimum
↳ lineage path
--auto-lineage-euk  Run auto-placement just on eukaryote tree to find optimum lineage
↳ path
-c N, --cpu N       Specify the number (N=integer) of threads/cores to use.
-f, --force         Force rewriting of existing files. Must be used when output files
↳ with the provided name already exist.
-r, --restart       Continue a run that had already partially completed.
-q, --quiet         Disable the info logs, displays only errors
--out_path OUTPUT_PATH
                    Optional location for results folder, excluding results folder
↳ name. Default is current working directory.
--download_path DOWNLOAD_PATH
                    Specify local filepath for storing BUSCO dataset downloads
--datasets_version DATASETS_VERSION
                    Specify the version of BUSCO datasets, e.g. odb10
--download_base_url DOWNLOAD_BASE_URL
                    Set the url to the remote BUSCO dataset location
--update-data       Download and replace with last versions all lineages datasets and
↳ files necessary to their automated selection
--offline           To indicate that BUSCO cannot attempt to download files
--metaeuk_parameters METAEUK_PARAMETERS
                    Pass additional arguments to Metaeuk for the first run. All
↳ arguments should be contained within a single pair of quotation marks, separated by
↳ commas. E.g. "--param1=1,--param2=2"
--metaeuk_rerun_parameters METAEUK_RERUN_PARAMETERS
                    Pass additional arguments to Metaeuk for the second run. All
↳ arguments should be contained within a single pair of quotation marks, separated by
↳ commas. E.g. "--param1=1,--param2=2"
-e N, --evaluate N  E-value cutoff for BLAST searches. Allowed formats, 0.001 or 1e-03
↳ (Default: 1e-03)
--limit REGION_LIMIT How many candidate regions (contig or transcript) to consider per
↳ BUSCO (default: 3)
--augustus          Use augustus gene predictor for eukaryote runs
--augustus_parameters AUGUSTUS_PARAMETERS
                    Pass additional arguments to Augustus. All arguments should be
↳ contained within a single pair of quotation marks, separated by commas. E.g. "--
↳ param1=1,--param2=2"
--augustus_species AUGUSTUS_SPECIES
                    Specify a species for Augustus training.
--long              Optimization Augustus self-training mode (Default: Off); adds
↳ considerably to the run time, but can improve results for some non-model organisms
--config CONFIG_FILE Provide a config file
-v, --version       Show this version and exit
-h, --help          Show this help message and exit
--list-datasets     Print the list of available BUSCO datasets

```

The minimum requirements to run BUSCO are

Running BUSCO

Mandatory arguments

```
busco -i [SEQUENCE_FILE] -l [LINEAGE] -o [OUTPUT_NAME] -m [MODE] [OTHER OPTIONS]
```

Mandatory arguments unless provided in the config file

`-i` or `--in` defines the input file to analyse which is either a nucleotide fasta file or a protein fasta file, depending on the BUSCO mode. As of v5.1.0 the input argument can now also be a directory containing fasta files to run in batch mode.

`-o` or `--out` defines the folder that will contain all results, logs, and intermediate data

`-m` or `--mode` sets the assessment MODE: genome, proteins, transcriptome

`-l` or `--lineage_dataset`

Launch a BUSCO run using...

- `-i genome.fasta`
- `-m genome`
- `-o haplotype1_busco` (or `haplotype2_busco`)
- `-l eudicots_odb10`
- `-c 4`

Task B

Run Assemblytics to compare both haplotypes

Now that we have two phased assemblies, one for each haplotype, we should do some sanity checks. Remember: once you move on from this phase of a genome project, it is hard to “go back” and fix issues with the genome assembly.

You’ve run [assemblytics](#) before, so you should be familiar. This will take ~6 hours. Focus in particular on the synteny dotplot that is produced. We’ll want to make sure our two haplotypes are largely contiguous with each other.

Note: just because a genome is highly heterozygous doesn’t mean that the two haplotypes will be structurally similar. That is, there can certainly be large-scale structural rearrangements and variants that occur.

Mastering Content

After your BUSCO run has finished, use the “generate_plot.by” script from the user guide to construct a barplot of both haplotypes that shows the number of Complete, Fragmented, Missing genes, like this:

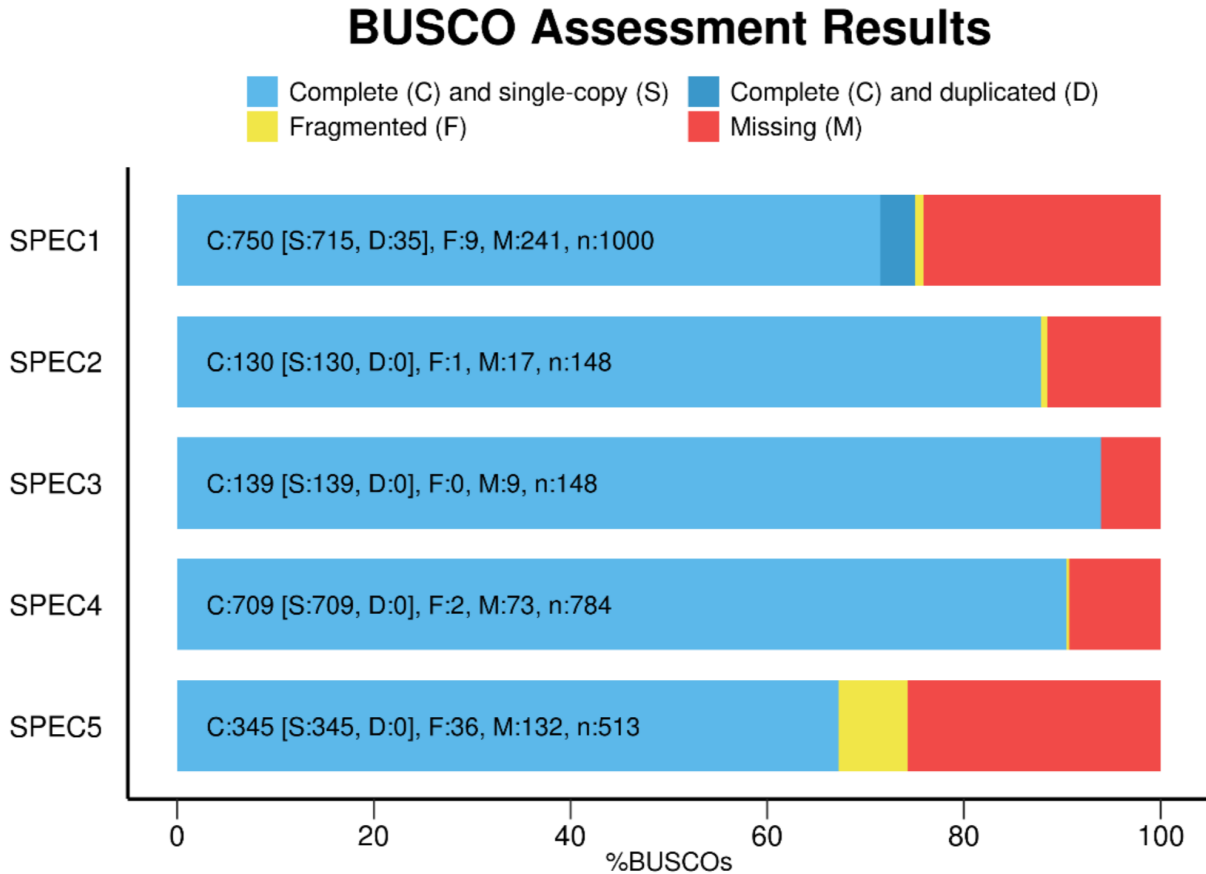


Fig. 34: Image Source [Busco Website](#)

1.6 Module 4: Genome Annotation

1.6.1 Lesson 1: Genome Arithmetic

4.1 Instructions

4.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [BedTools Manual](#)
- [BED Format](#)
- [Interactive Introduction to BedTools](#)

4.1 Lesson

Learning Objectives

Vocabulary

Learning Material

4.1 Lab Exercises

Overview

In this lab, we will learn the basics of performing genome arithmetic, and operating on intervals.

We will do two major things in this lab:

- Work through an interactive bedtools tutorial from the Quinlan lab (who developed it)
- Walk through some examples of different file formats that bedtools can operate on
- Run Liftoff2 to lift-over gene annotations from the *Q. rubra* genome onto ours

“Be for real, don’t be a stranger” - Spice Girls

Task A

At the end of the day, a fasta file with our genome assembly is just a string of ATCG’s. We have to layer annotations on top of that fasta file. These can be gene annotations (in .gff or .gtf format), repetitive element annotations (in .gff format), RNA-seq alignments (in .bam format), RNA-seq expression counts (in .bed format), and more.

These annotation formats all have one thing in common: they are based on intervals of genomic coordinates that have starts and stops. For instance, GeneA is on chromosome 4 at basepair location 21244 to 23299.

How can we start to interact with those annotations? Typically, you will need to overlap intervals of interest with other features of the genome, again represented as intervals. For example, you may want to overlap transcription factor binding sites with CpG islands or promoters to quantify what percentage of binding sites overlap with your regions of interest. Overlapping mapped reads from high-throughput sequencing experiments with genomic features such as exons, promoters, and enhancers can also be classified as operations on genomic intervals. You can think of a million other ways that involve overlapping two sets of different features on the genome. For example,

- “What gene overlaps with this QTL peak?”
- “I need the 1 kb in the 5 upstream region of these 20 genes”
- “Which genes are within 500 nt of an LTR retrotransposon?”

All of these questions can be answered with bedtools, perhaps the most useful set of tools to answer questions regarding genome arithmetic.

How bedtools intersect works with one or more files.

Work through this [interactive Bedtools tutorial](#), based on the Quinlan lab (who wrote bedtools).

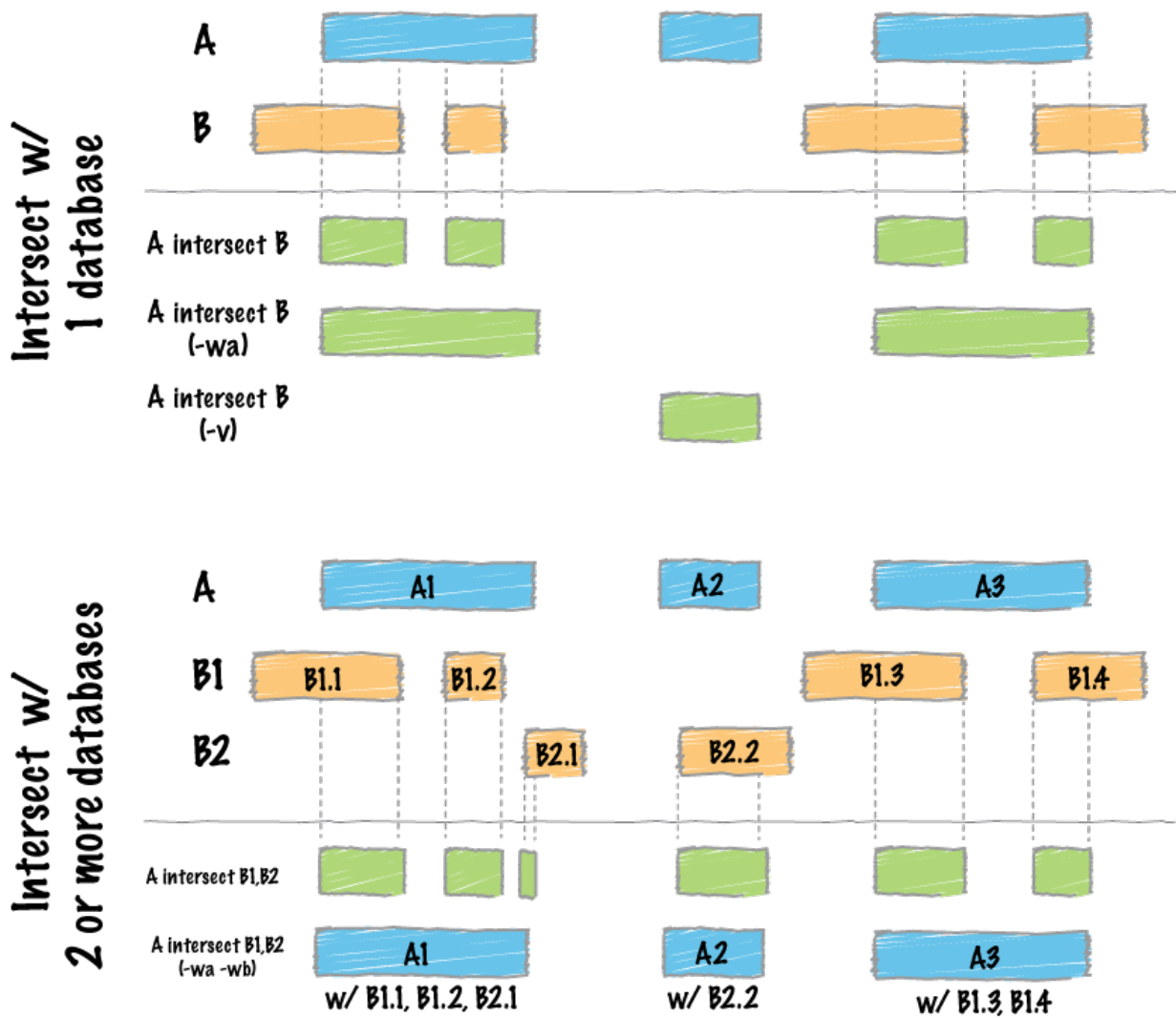


Fig. 35: Image source: BedTools documentation

Task B

Explore data formats

Bedtools can operate on a number of different formats of files., e.g. vcf, bam, bed, fasta, at the same time. Having some knowledge of these formats is necessary.

Format 1: BED (Browser Extensible Data) format provides a flexible way to define the data lines that are displayed in an annotation track. BED lines have three required fields and nine additional optional fields. The number of fields per line must be consistent throughout any single set of data in an annotation track. The order of the optional fields is binding: lower-numbered fields must always be populated if higher-numbered fields are used.

The first three required BED fields are:

- **chrom** – The name of the chromosome (e.g. chr3, chrY, chr2_random) or scaffold (e.g. scaffold10671).
- **chromStart** – The starting position of the feature in the chromosome or scaffold. The first base in a chromosome is numbered 0.
- **chromEnd** – The ending position of the feature in the chromosome or scaffold. The chromEnd base is not included in the display of the feature, however, the number in [position format](#) will be represented. For example, the first 100 bases of chromosome 1 are defined as chrom=1, chromStart=0, chromEnd=100, and span the bases numbered 0-99 in our software (not 0-100), but will represent the position notation chr1:1-100. Read more [here](#). chromStart and chromEnd can be identical, creating a feature of length 0, commonly used for insertions. For example, use chromStart=0, chromEnd=0 to represent an insertion before the first nucleotide of a chromosome.

After these 3 mandatory columns, you can append all kinds of additional information to these locations. You can add gene names, strand (+ or -), gene counts (expression), and so on.

Format 2: GFF (General Feature Format) consists of one line per feature, each containing 9 columns of data, plus optional track definition lines.

Fields **must** be tab-separated. Also, all but the final field in each feature line must contain a value; “empty” columns should be denoted with a ‘.’

- **seqname** – name of the chromosome or scaffold; chromosome names can be given with or without the ‘chr’ prefix. Important note: the seqname must be one used within Ensembl, i.e. a standard chromosome name or an Ensembl identifier such as a scaffold ID, without any additional content such as species or assembly. See the example GFF output below.
- **source** – name of the program that generated this feature, or the data source (database or project name)
- **feature** – feature type name, e.g. Gene, Variation, Similarity
- **start** – Start position* of the feature, with sequence numbering starting at 1.
- **end** – End position* of the feature, with sequence numbering starting at 1.
- **score** – A floating point value.
- **strand** – defined as + (forward) or – (reverse).
- **frame** – One of ‘0’, ‘1’ or ‘2’. ‘0’ indicates that the first base of the feature is the first base of a codon, ‘1’ that the second base is the first base of a codon, and so on.
- **attribute** – A semicolon-separated list of tag-value pairs, providing additional information about each feature.

What to look out for: 0-based vs 1-based coordinates

Do we start at 0, or start at 1?

The example above shows (an imaginary) first seven nucleotides of sequence on chromosome 1:

- 1-based coordinate system - Numbers nucleotides directly

```

0 ##gff-version 3.2.1
1 ##sequence-region ctg123 1 1497228
2 ctg123 . gene 1000 9000 . + . ID=gene00001;Name=EDEN
3 ctg123 . TF_binding_site 1000 1012 . + . ID=tfbs00001;Parent=gene00001
4 ctg123 . mRNA 1050 9000 . + . ID=mRNA00001;Parent=gene00001;Name=EDEN.1
5 ctg123 . mRNA 1050 9000 . + . ID=mRNA00002;Parent=gene00001;Name=EDEN.2
6 ctg123 . mRNA 1300 9000 . + . ID=mRNA00003;Parent=gene00001;Name=EDEN.3
7 ctg123 . exon 1300 1500 . + . ID=exon00001;Parent=mRNA00003
8 ctg123 . exon 1050 1500 . + . ID=exon00002;Parent=mRNA00001,mRNA00002
9 ctg123 . exon 3000 3902 . + . ID=exon00003;Parent=mRNA00001,mRNA00003
10 ctg123 . exon 5000 5500 . + . ID=exon00004;Parent=mRNA00001,mRNA00002,mRNA00003
11 ctg123 . exon 7000 9000 . + . ID=exon00005;Parent=mRNA00001,mRNA00002,mRNA00003
12 ctg123 . CDS 1201 1500 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
13 ctg123 . CDS 3000 3902 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
14 ctg123 . CDS 5000 5500 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
15 ctg123 . CDS 7000 7600 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
16 ctg123 . CDS 1201 1500 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
17 ctg123 . CDS 5000 5500 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
18 ctg123 . CDS 7000 7600 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
19 ctg123 . CDS 3301 3902 . + 0 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
20 ctg123 . CDS 5000 5500 . + 1 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
21 ctg123 . CDS 7000 7600 . + 1 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
22 ctg123 . CDS 3391 3902 . + 0 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
23 ctg123 . CDS 5000 5500 . + 1 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
24 ctg123 . CDS 7000 7600 . + 1 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4

```

Fig. 36: Image source: [Next-Generation Sequencing Analysis Resources](#)

- 0-based coordinate system - Numbers between nucleotides

chr1		T		A		C		G		T		C		A	
1-based															
		1		2		3		4		5		6		7	
0-based	0		1		2		3		4		5		6		7

Fig. 37: Image source:

0-based: BED, BAM, BCF

1-based: GTF, GFF, SAM, VCF, BLAST, GenBank/EMBL

Bedtools recognizes these differences and inter-converts for you.

1.6.2 Lesson 2: Classifying Repeats

4.2 Instructions

4.2 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

- [GyDB](#)
- [Global analysis of repetitive DNA from unassembled sequence reads using RepeatExplorer2](#)
- [Repeat Masker](#)
- [The genomic ecosystem of transposable elements in maize](#)

4.2 Lesson

Learning Objectives

Vocabulary

Learning Material

4.2 Lab Exercises

Overview

This is a longer lab that takes us away from the command line.

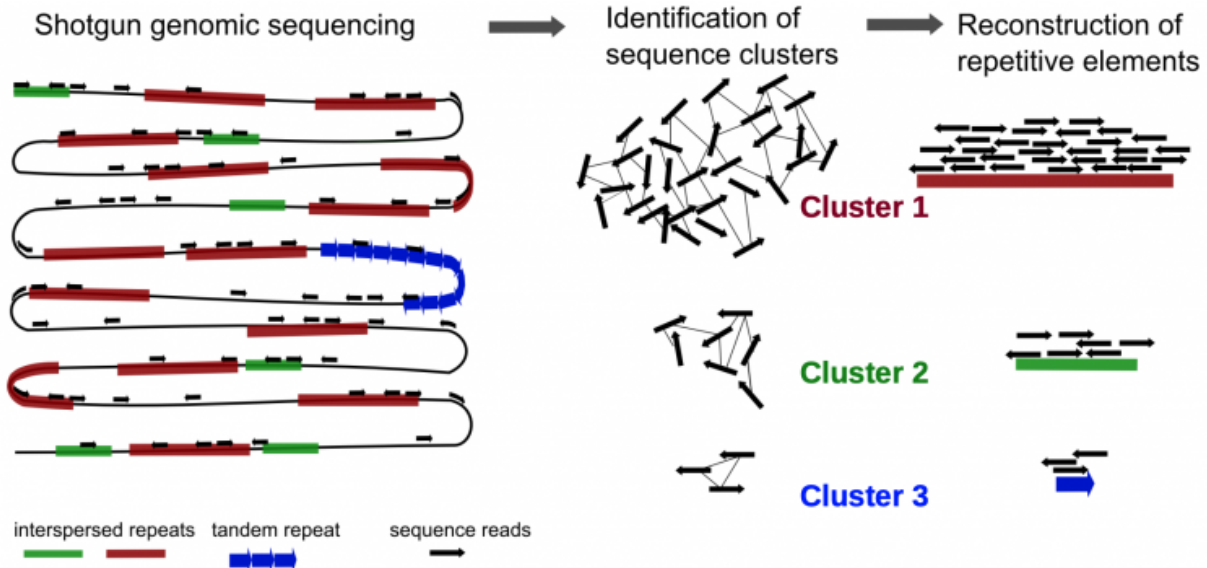
We will do two major things in this lab:

- Learn how to use the Repeatexplorer2 portal on Galaxy
- Walk through the RepeatExplorer2 Nature Protocols paper and subsample shotgun data from Toomers Oak
- Upload data to Galaxy
- Run RepeatExplorer2 to characterize TE content in Toomers Oak

Task A

We can use Illumina paired-end shotgun genome data to predict how repetitive a genome is. Plant genomes are filled with repetitive DNA — but do we need to sequence an entire plant genome to figure out what percent of a genome is filled with repeats?

No. Imagine you have a 3 billion piece puzzle. Even if you only take 10,000 random pieces of that puzzle, you can roughly see what the puzzle *should* be. That premise underlies how RepeatExplorer2 works. Even with a small subsample, 1% coverage of the genome, you can nicely estimate the repetitive content of a plant genome. In short, it works by taking a small sample of reads from your genome of interest, clustering them together, and trying to reconstruct consensus sequences of repetitive elements. By doing this, you can estimate what total % of a genome is repetitive, and break down individual superfamilies of transposable elements.



Here is the main RepeatExplorer2 site to start from: <https://galaxy-elixir.cerit-sc.cz/>

First, Register (don't re-use any known username or password of yours!) : <https://perun.cesnet.cz/elixir2/registrar/?vo=elixir-cz&group=repeatExplorer>

Then you'll have access to the Galaxy server: <https://repeatexplorer-elixir.cerit-sc.cz/galaxy/>

Second, follow along with the Nature Protocols paper describing how to run RepeatExplorer2. You'll be using the Toomers Oak PE150 PCR-free shotgun data, though. I've also left this pdf in the class resources for this module.

<https://www.nature.com/articles/s41596-020-0400-y.epdf>

1.6.3 Lesson 3: Annotating Repeats

4.3 Instructions

4.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

Error: LTR Assembly Index link is missing

- [EDTA Manuscript](#)
- [LTR Assembly Index](#)

4.3 Lesson

Learning Objectives

Vocabulary

Learning Material

4.3 Lab Exercises

Overview

This lab will walk us through the process of annotating a genome for repeats.

We will do two major things in this lab:

- Learn how to run EDTA, an all-in-one transposon annotation pipeline
- Build an insertion time distribution for LTR elements of major superfamilies

“Be for real, don’t be a stranger” - Spice Girls

Task A

Now that we have a fully phased genome assembly, we can begin the process of annotation. There are two major phases to annotation: repetitive elements, and genes. First, we need to annotate repetitive elements so that we can mask them before annotating genes, effectively “hiding” the repeats from gene annotation and prediction programs. Repeats are the most frustrating aspect of annotating a genome. Not only are they ubiquitous, and typically different between every species, **if you do a poor job of repeat annotation, your gene annotation will also be poor**. Remember: repetitive elements contain genes!

EDTA is an all-in-one repeat annotation pipeline. It runs a handful of programs that identify different kinds of repetitive elements, such as LTR_FINDER, LTRharvest, Generic Repeat Finder, and HelitronScanner. Then it filters these annotations to build a custom repeat library, and creates a .gff file of repeat annotations for the genome assembly. Here’s an overview of the pipeline:

The EDTA workflow First, read the github wiki page, then create a new Conda environment for EDTA:

```
conda create -n EDTA
conda activate EDTA
```

Then install via Conda.

```
conda install -c bioconda -c conda-forge edta
```

.... This is taking **forever**, isn’t it? My install just hangs. I think it finishes in a few hours, but we don’t have that kind of time to waste, do we? Sometimes Conda installs are just painfully slow.

Mamba was developed to beat this problem. Mamba is a reimplementaion of the conda package manager in C++. It allows for:

- parallel downloading of repository data and package files using multi-threading
- libsolv for much faster dependency solving, a state of the art library used in the RPM package manager of Red Hat, Fedora and OpenSUSE
- core parts of mamba are implemented in C++ for maximum efficiency

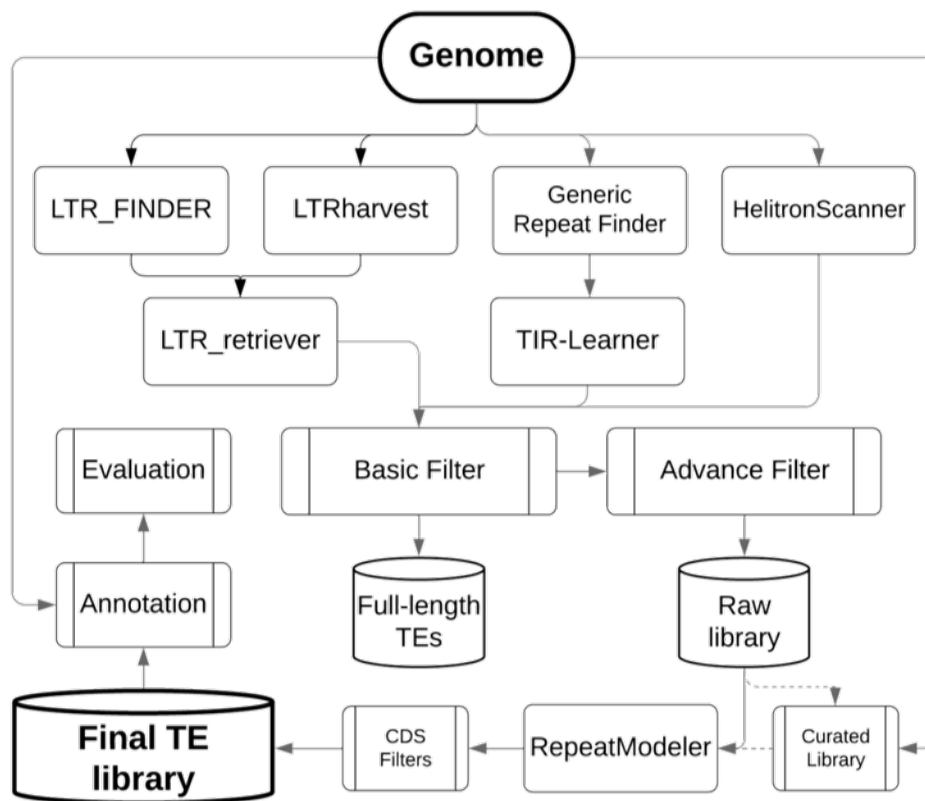


Fig. 38: Image Source: [EDTA GitHub Page](#)

Use `mamba` to accelerate the installation:

```
conda install -c conda-forge mamba  
mamba install -c conda-forge -c bioconda edta python=3.6 tensorflow=1.14 'h5py<3'
```

Test your installation by typing:

```
EDTA.pl -h
```

Task B

First, read some of the caveats of EDTA. It looks like all we need, at minimum, is just a genome file .fasta file. We have two, one for each haplotype, so we will divide-and-conquer as a class. We'll decide which half of the class takes haplotype1 and which half takes haplotype2 when we meet.

It does mention that “make sure sequence names are short and simple”. Okay, I’ve taken care of this, and renamed our .fasta headers to be “Qv1_chr1”, “Qv1_chr2”, etc.

Now, read the EDTA Usage section, and see if you can launch a full run on your own. Remember that you can use at most 4 threads.

Mastering Content

Next, we want to produce some figures that describe the repeat landscape in the genome. In particular, we want to show the percent sequence divergence of similar repeats, in order to describe any “bursts” of repeat expansion.

We’re not the first people using EDTA who want to do the same. Check out this Github Issues describing the same thing: <https://github.com/oushujun/EDTA/issues/92>

If you’ve used R before, this should look somewhat familiar. Open the R terminal in your PRAXIS welcome page, install the required packages (google if you don’t remember how).

1.6.4 Lesson 4: Annotating Genes with RNA-Seq

4.4 Instructions

4.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

– [A beginner’s guide to eukaryotic genome annotation](#)

4.4 Lesson

Learning Objectives

Vocabulary

Learning Material

4.4 Lab Exercises

Overview

This lab will walk us through the process of annotating a genome for genes using the BRAKER2 pipeline.

We will do three major things in this lab:

- Install BRAKER
- Collect all of the available Quercus peptides that exist, and align them to our haplotype references
- Run the BRAKER2 pipeline using orthologous gene evidence to train AUGUSTUS and build an annotation set

“Be for real, don’t be a stranger”` - Spice Girls

Task A

First, we need to produce a softmasked genome. Softmasked means we convert any repetitive region of the genome to a lowercase letter. BRAKER2, our gene annotation pipeline, prefers softmasked genomes so it can “ignore” those regions for gene annotation.

Bedtools has a tool that can do this for us. Using the the .gff3 you produced as part of EDTA, e.g. (toomers.hap1.fasta.mod.EDTA.TEanno.gff3), run bedtools maskfasta with the -soft flag to create a softmasked genome fasta file.

Task B

Next we will run the BRAKER2 pipeline to annotate genes in the reference haplotypes you have assembled. In the absence of RNA-seq data, we will need to rely on closely related peptides for gene prediction. Luckily, there are three other Quercus genomes that exist: lobata, robur, and rubra. We can leverage the gene annotations from these three genomes, as well as the seemingly high degree of conservation across the Quercus genus, to train gene predictors and build an annotation set.

First, read the [BRAKER2 github page](#). We will be using “Pipeline C” that leverages orthologs of any evolutionary distance to train AUGUSTUS.

You can install BRAKER2 with conda, with the caveat that it is sometimes a few versions behind. That will be fine for us, though.

Create a new conda environment called “braker”, then install BRAKER2 using Conda.

First, collect the **protein fasta annotations** for Quercus rubra, robur, lobata. Make sure they’re proteins/peptides! Use Phytozome and/or any other means you need to. Concatenate them into a single .fasta file, like this:

```
cat species1.fasta species2.fasta species3.fasta > quercus_proteins.fasta
```

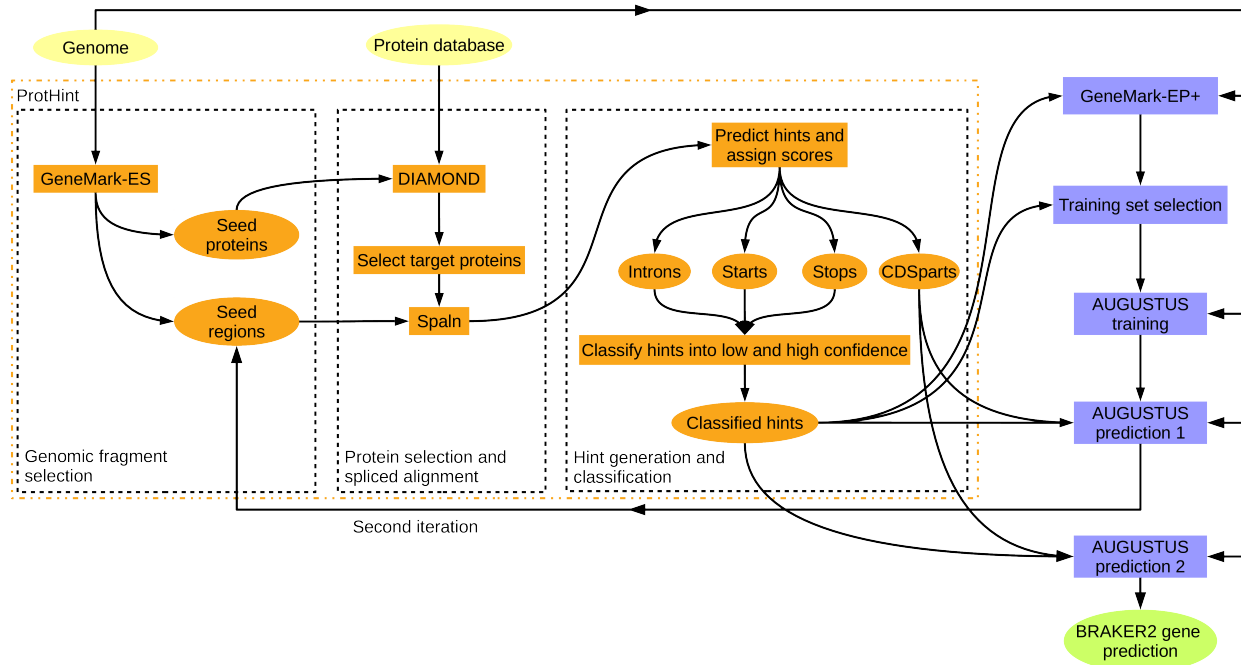


Fig. 39: Image Source: [BRAKER2 GitHub Page](#)

Then, run BRAKER2 using the “proteins from any evolutionary distance” pipeline. Make sure you use the softmask flag!

1.6.5 Lesson 5: Quantifying Gene Expression

4.5 Instructions

4.5 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

4.5 Lesson

Learning Objectives

Vocabulary

Learning Material

4.5 Lab Exercises

1.7 Module 5: Comparative Genomics

1.7.1 Lesson 1: Identifying Synteny

5.1 Instructions

5.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

5.1 Lesson

Learning Objectives

Vocabulary

Learning Material

5.1 Lab Exercises

1.7.2 Lesson 2: Building Gene Families

5.2 Instructions

5.2 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

5.2 Lesson

Learning Objectives

Vocabulary

Learning Material

5.2 Lab Exercises

1.7.3 Lesson 3: Estimating a Gene Tree

5.3 Instructions

5.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

5.3 Lesson

Learning Objectives

Vocabulary

Learning Material

5.3 Lab Exercises

1.7.4 Lesson 4: Aligning Resequencing Data

5.4 Instructions

5.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

5.4 Lesson

Learning Objectives

Vocabulary

Learning Material

5.4 Lab Exercises

1.7.5 Lesson 5: Variant Calling

5.5 Instructions

5.5 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

5.5 Lesson

Learning Objectives

Vocabulary

Learning Material

5.5 Lab Exercises

1.8 Module 6: Publishing Scientific Results

1.8.1 Lesson 1: Intro to R Programming

6.1 Instructions

6.1 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

6.1 Lesson

Learning Objectives

Vocabulary

Learning Material

6.1 Lab Exercises

1.8.2 Lesson 2: Plotting Genomic Data

6.2 Instructions

6.2 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

6.2 Lesson

Learning Objectives

Vocabulary

Learning Material

6.2 Lab Exercises

1.8.3 Lesson 3: Using Gene Browsers

6.3 Instructions

6.3 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

6.3 Lesson

Learning Objectives

Vocabulary

Learning Material

6.3 Lab Exercises

1.8.4 Lesson 4: Making Data Public

6.4 Instructions

6.4 Resources

This page contains a list of resources that are referenced by this lesson or which may provide additional information for the topic of this lesson.

6.4 Lesson

Learning Objectives

Vocabulary

Learning Material

6.4 Lab Exercises