Week 4
OptParse

24 October 2018

CS 35L Lab 4

Jeremy Rotman

Announcements

- ➔ Assignment #3 is due Friday by 11:55pm
- ➔ For Assignment #10
  - ◆ Email me to tell me what story you are choosing
  - ◆ Here is the link to see what stories people have signed up for already
    - Choose a story at least one week before you present
  - ◆ Here is the link to sign up to present
    - Sign up to present by Friday Oct 26
- ➔ The links are only accessible to users with a UCLA email

Questions?

Outline

- ➔ More on optparse
- ➔ Example presentation

Optparse Actions

- There are a fixed set of actions already in optparse
  - You should not need to make new ones
- Most of these are related to storing an argument
  - The type of variable you are storing may make a difference though

Optparse Actions: Store

- The most basic and probably most useful action
- It is also the default action
  - Meaning you don’t technically need to specify it
- This includes 3 arguments in the add_option function
  - action=“store”
    - Declares the action as store
  - type=“string”
    - Declares the type of argument being stored as a string
    - String is default
  - dest=“var_name”
    - Indicates the variable you want to store the argument as

Optparse Actions: Store

So an example may look like:

```
parser.add_option(“-f”, “--file”,
   action=“store”, type=“string”, dest=“filename”)
```

Optparse Actions: Booleans

- If you want an option that is simply a flag to turn things on or off, there is a different set of actions similar to store
  - store_true
  - store_false
- These will still need to be stored in a variable, but the type is not important

Optparse Actions: Booleans

So for example:

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”)

parser.add_option(“-q”, action=“store_false”,
   dest=“verbose”)
```

More Optparse Actions

- store_const
  - Store a constant value
- append
  - Append this option’s argument to a list
- count
  - Increment a counter by one
- callback
  - Call a specified function

Default Values for Options

- ➔ Everytime you add an option, you can include a default value
- ➔ Say for example, we want the option for the script to be verbose, but want it to be quiet by default

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”, default=False)
```

Generating Help Messages

- By default, Optparse uses the -h or --help options to display help messages
- The help messages include multiple useful things

Program Usage

- When creating your OptionParser, you can specify a program's usage message
- What is a usage message?

Program Usage

- ➔ When creating your OptionParser, you can specify a program's usage message
- ➔ What is a usage message?
  - ◆ The line at the top of the help file describing how to run the program
  - ◆ Additionally, it might include a description of what the program does

```
Usage: randline.py [OPTION]... FILE

Output randomly selected lines from FILE.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -n NUMLINES, --numlines=NUMLINES
                        output NUMLINES lines (default 1)
```

Program Usage

For example this creates the usage message in randline.py

```
usage_msg = """%prog [OPTION]... FILE

Output randomly selected lines from FILE"""
parser = OptionParser(usage=usage_msg)
```

Option Help Messages

- In addition to the usage at the top, the help display also showed help messages for all of the options that you could use
- This message must be defined in the add_option function

Option Help Messages

Going back to our verbose example before:

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”, default=False,
   help=“Print out extra information about
   the program while running”)
```

Printing Version Number

- ➔ Similar to usage, when creating the OptionParser object, you can give it a version message
- ➔ This will naturally be tied to the --version option

```
version_msg = “%prog 2.0”
parser = OptionParser(version=version_msg)
```

Error messages

- ➔ OptionParser objects also have a built in error function
- ➔ This won't actively look for errors
  - ◆ Instead you must anticipate potential errors and call the function when you encounter these errors
    - E.g. being passed two boolean options that contradict each other
    - optparse does catch some things, like option argument type
- ➔ Does a few things
  - ◆ Print the usage message to stderr
  - ◆ Print the error message (the function parameter) to stderr
  - ◆ Exit with error status 2

Example Presentation

- My example will be based on this correspondence in Nature Methods
  - Mutation frequency is not increased in CRISPR–Cas9-edited mice
- This gets a little out of the computer science world
  - But I didn't want to take something that anyone else might want to do

CRISPR-Cas9

➔ What is it?

CRISPR-Cas9

- What is it?
  - A specific system used in gene-editing
    - Originally coming from an organic bacterial system
- How do we edit genes with it?
  - CRISPR-Cas9 itself is more like a pair of scissors
  - It can find a specific piece of DNA and cut it out
  - Gene editing is then done using natural repair machinery, or an included piece of replacement DNA

What’s the problem?

What's the problem?

- ➔ This could make our DNA vulnerable
- ➔ There are rising concerns that CRISPR-Cas9 may introduce extra mutations in our DNA
- ➔ In fact, one study claims that they found hundreds of non-targeted mutations in CRISPR-Cas9 treated mice[1]

1. Schaefer, K. A. et al. *Nat. Methods* **14**, 547-548 (2017)

What's the problem?

- ➔ This could make our DNA vulnerable
- ➔ There are rising concerns that CRISPR-Cas9 may introduce extra mutations in our DNA
- ➔ In fact, one study claims that they found hundreds of non-targeted mutations in CRISPR-Cas9 treated mice[1]
- ➔ So is CRISPR-Cas9 doomed to fail?

1. Schaefer, K. A. et al. *Nat. Methods* **14**, 547-548 (2017)

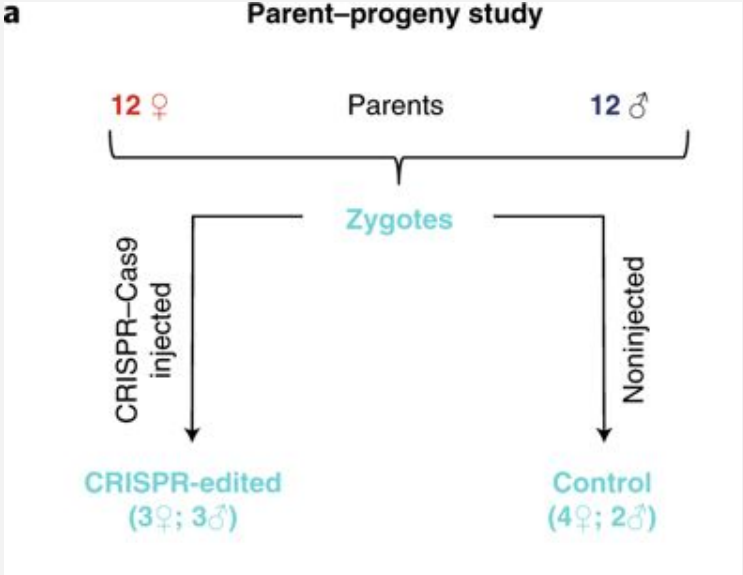
Problems with mutation counting

- ➔ According to Willi et al. there were flaws in Schaefer's study[2]
  - ◆ Pre-existing variants obtained from parents were ignored
    - A pre-existing variant is not de novo
  - ◆ The sample size was too small
    - One control mouse, and two CRSIPR-Cas9 mice
- ➔ Due to this, Willi et al. designed their own study that takes these issues into account

2. Willi M. et al. *Nat. Methods* **15**, 756-758 (2018)

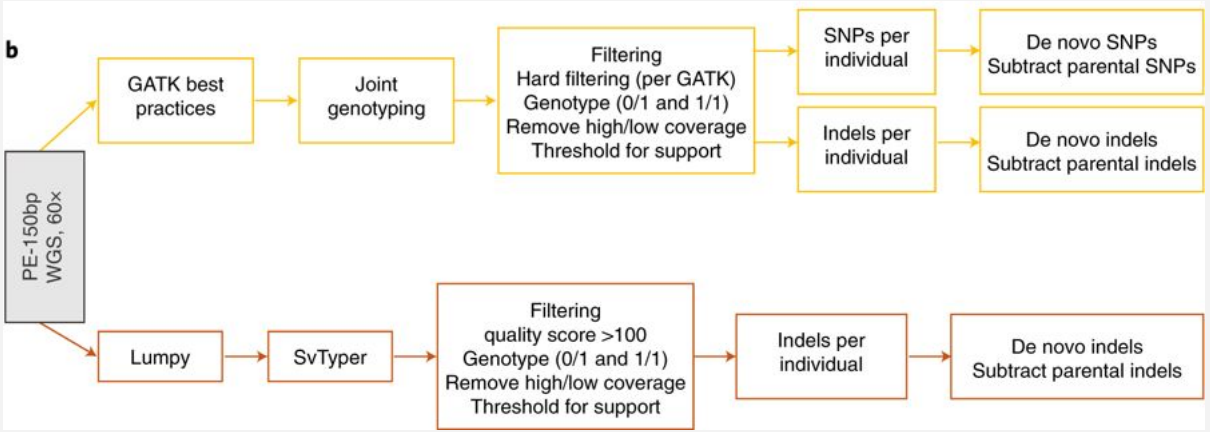
Parent-Progeny Study

In this study, they took 12 mice, along with their 24 parents, and studied both CRISPR-edited mice and Control mice



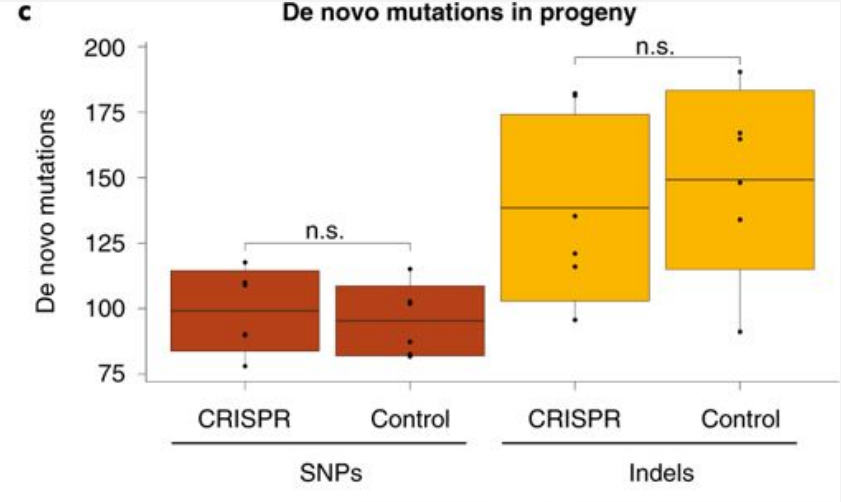
Parent-Progeny Study

Here the entire process is displayed. Particularly, they note how they subtract the parents' indels and SNPs



Parent-Progeny Study

Most importantly, there is no significant difference between the number mutations in the control group and the CRISPR-edited group



What's Next?

- ➔ Ultimately, Willi et al. successfully proved Schaefer et al. to be wrong
- ➔ But what comes next?

What's Next?

- ➔ Ultimately, Willi et al. successfully proved Schaefer et al. to be wrong
- ➔ But what comes next?
  - ◆ Even Willi et al. admit that extra mutations at target sites are still a concern
  - ◆ All we have proven is that CRISPR-Cas9 does not increase a mouse's overall mutation rate
  - ◆ But what if we increase risk at the target site?
  - ◆ And what if that target site is especially important?
  - ◆ Will CRISPR-Cas9 ever going to be deemed safe enough to actually make a difference to humans?

Questions?