Seminar III: R/Bioconductor

Leonardo Collado Torres
lcollado@lcg.unam.mx
Bachelor in Genomic Sciences
www.lcg.unam.mx/~lcollado/

August - December, 2009

Working with HTS data: a *simulated* case study

Intro

R for scripts

BLAST

Velvet

Bowtie

Work

:O

Prepare yourself!

About

- The idea is to learn how to use R as a scripting language to call external programs such as BLAST, Velvet and Bowtie.
- We'll run these programs with as many default options as we can :)

For today you'll need

- formatdb
- blastall[1]
- Velvet
  `http://www.ebi.ac.uk/~zerbino/velvet/`
- Bowtie
  `http://bowtie-bio.sourceforge.net/index.shtml`
- Of course, some Bioconductor:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("chipseq"))
```

- And a LINUX or UNIX environment :)

[1] With Ubuntu use: `sudo apt-get install blast2` and voilÃą :)

External programs

- As you know, BLAST is very used and useful to find local alignments.
- Velvet is a great program to assemble short reads into contigs.
- Bowtie is great to align short reads to a reference genome.

Running R scripts

- Thanks to functions like system, you can use R as your scripting language.
- Of course, a lot of people prefer to use `shell` directly.
- Using R can be useful to make some plots on the fly and proc.time helps us track the time spent running our script.
- You can either use:
  `R CMD BATCH file.R` or
  `Rscript file.R > file.log` as a shortcut

Use paste

- To build system calls, the paste function with the sep or collapse arguments is quite useful:

```
> args <- c(1, 2)
> call <- paste("-arg1", args[1],
+     "-arg2", args[2], sep = " ")
> print(call)

[1] "-arg1 1 -arg2 2"

> call2 <- paste(c("-args", args),
+     collapse = " ")
> print(call2)

[1] "-args 1 2"
```

- Using a print coupled to a system.time can be useful for slow commands.

Command line

- You all know how it works, and have run it through the web interface:
  `http://www.ncbi.nlm.nih.gov/BLAST/`
- To run BLAST in command-line you mainly need two programs:
  1. formatdb: builds the database (targets)
  2. blastall: actually runs BLAST

formatdb

Main arguments:

- -i: the input file
- -p: the type of database. Use T for proteins or F for nucleotides.
- -n: the output name, meaning the name of the database.

Optional ones I use:

- -t: the title
- -l: the log file name
- -V: to check the names of the targets use V

For more info check:

- `formatdb - -help` on the terminal
- `http://www.molbiol.ox.ac.uk/analysis_tools/BLAST/formatdb.shtml`

blastall

Main arguments:

- -p: the type of BLAST to be run. BLASTP, BLASTN, ...
- -d: the database name[2]
- -i: the input file name.
- -o: the output file name.

Optional ones I use:

- -e: the maximum e value allowed for the output file.
- -m: the format of the output file. I like format 8 :) Click here for examples.

For more info check:

- `blastall - -help` on the terminal
- `http://www.molbiol.ox.ac.uk/analysis_tools/BLAST/BLAST_blastall.shtml`

[2] For custom dbs, use the path to the db.

Quick intro

- Published in 2008, Velvet is the most popular *de novo* genome assembler for short reads such as those generated by Illumina.
- Its based on *de Brujin graphs* and its most important parameter is the k-mer length; similar to the word size.
- For more info check the paper:
  `http://www.ncbi.nlm.nih.gov/pubmed/18349386`

velveth

In order to use Velvet we first need to run velveth and specify the:

- output dir: first value (without any flag)
- k-mer length: an integer up to 31.[3]
- input file format: main options are fasta and fastq.
- type of data: mainly either short or long.
- input file name

For more info type `velveth` or check the Velvet manual.

[3] The lower the value, the slower it runs.

velvetg

After running `velveth` we can run velvetg one or more times on the same directory.
Velvetg actually runs Velvet and creates the contigs.
To run it type `velvetg` specifying:

- the output dir: again, the first unflagged value.
- some filtering or output options such as `min_contig_lgth`

For more info type `velvetg` or check the Velvet manual.

Quick intro

- Bowtie is a second generation[4] short read aligner that is VERY fast.
- It's based on the Burrows-Wheeler Transform (BWT) as other fast aligners. Therefore, it builds an index[5] of the reference genome, which speeds up the process.
- It's very well maintained and for more info check the homepage and related paper :)

[4] If you consider MAQ to be the first generation.

[5] Similar to the BLAST database.

bowtie-build

- It's very simple to use :)
- Just specify the input file[6] and the output name for the index.
- After building the index, move the output files[7] into `PathToBowtie/indexes/`

For more info type: `bowtie-build -h`

[6] In FASTA format.

[7] Yup, a few are created.

bowtie

After building your index a quick way to check it is to type:
`bowtie -c IndexName GCGTGAGCTATGAGAAAGCGCCACGCTTCC`
Then to run Bowtie I normally use the following arguments:

- -f: the input file name
- - -all: to force Bowtie to find all the alignments.[8]
- - -al: the output name for the FASTA file with the reads that were aligned.
- - -un: the reads that did not align.
- Other useful arguments are -m and - -max.

For more info type `bowtie -h` or check the manual.

[8] Obviously increases the time quite a bit on real cases.

Data and problem to solve

- I generated 18 sets of 70 thousand 50bp reads. One set per student ;)[9]
- Imagine that these sequences come from a genome related to our species of interest. We want to find variation signatures such as: deletions, invertions and duplications.
- Always be open to *fishy* stuff!

[9] To find out which one is yours, use the order from *Usuarios* at *Cursos*. For example, Fonseca is number 4 and Zepeda Martinez is 17.

Part I

- We don't know the name of our species of interest!!!
- Find it out by building contigs and aligning them versus all known genomes (nucleotides).
- Explore[10] the reads that were not used to build the contigs.
- Conclude, remark, etc.

[10] Check the files, check the alphabet by cycle frequency, ...

Part II

- How many protein coding genes did we cover at 90% or greater identity and 90% or greater query coverage?
- You will need to download the FASTA file with the sequence from those genes. Easy to do with the GenBank identifier :)
- Conclude, remark, etc.

Part III

- Align the reads versus our the reference genome of our species of interest.
- Explore and compare the reads that align more than once and those that aling only once.
- Identify the number of deletions, duplications and inversions. Plots like coverageplot, densityplot and stripplot will be most useful. To use them re-check the chipseq workflow :) Make some example plots and for the latter two try to make plots spanning all the genome[11].
- Conclude, remark, etc.

[11] Only where you have reads mapped to it.

Optional parts

- Using the chipseq worflow, explore only those reads that map to more than one spot.
- Plot the reads using GenomeGraphs and add boxes for every known gene.
- Try to pinpoint the exact deleted, duplicated and/or inverted bases. Specially the breakpoints.

Time to work!

- Once you are done, let me know and I'll upload all files related to your case :)
- Compare your conclusions with files such as `segments.txt` and explore the `fig` folder.
- The `ref.fa` file is the actual *reference* genome from where I got the 70k reads. Feel free to map your reads to it; some will cannot be uniquely aligned!
- Once everyone is done, I'll upload the fastagen.R script that created all the data.

SessionInfo

```
> sessionInfo()

R version 2.10.0 (2009-10-26)
i686-pc-linux-gnu

locale:
 [1] LC_CTYPE=en_US.UTF-8
 [2] LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=C
 [6] LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8
 [8] LC_NAME=C
 [9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8
[12] LC_IDENTIFICATION=C

attached base packages:
```

SessionInfo

```
[1] stats     graphics  grDevices
[4] utils     datasets  methods
[7] base
```