Dennis Larsson 09.07.2020 Universität Wien

Demultiplexing using illumina2bam & process_radtags

Installation

This is a protocol to describe how the demultiplexing of multiplexed RAD-libraries is done. First download the bam-formated file that contains your sequenced reads. You should get a link to the file in an email from the sequencing facility.
This file contains all reads in all individuals in one mixed pile, so you want to separate the reads into files representing each individual. To do this you use a program called illumina2bam. This program can be tedious to install so I included installation instructions in this protocol.

1. Firstly the software requires java 8 (later version might work too) and the compiler ant. To install using 'apt', run the following commands in the terminal:
   sudo apt-get install openjdk-8-jdk openjdk-8-jre ant
2. Ant also requires the 'bcel' library in order to compile illumina2bam. Bcel (version 6.4.1 is the latest that I have tried and works, possible that later will work too) can be downloaded from here:
   https://commons.apache.org/proper/commons-bcel/download_bcel.cgi
3. Extract the compressed file and copy the file 'bcel-6.4.1.jar' in the 'bcel-6.4.1' folder to /usr/share/ant/lib. It is necessary to do this in the terminal since copying files into that folder require superuser rights (sudo) which is not normally granted in the graphical mode (change /path/to/ to whatever the path to the file is and change the file name to the version-specific name in your downloaded file):
   sudo cp path/to/bcel-6.4.1.jar /usr/share/ant/lib
   Then it might be necessary to give the file executive rights, this can be done using the command:
   sudo chmod +x /usr/share/ant/lib/bcel-6.4.1.jar
4. Now you are ready to install illumina2bam. Download the software form their github repository (https://github.com/wtsi-npg/illumina2bam) and extract the compressed file into the folder where you would like to install it. then navigate using the terminal into the installation folder and run the command *ant jar*

The final step of demultiplexing requires stacks 2. Download the latest version from their website (http://catchenlab.life.illinois.edu/stacks/). Extract into the folder where you want the installation files. Then navigate using the terminal into the installation folder and type the following commands:

1. *./configure*
2. *make* (the argument '-j x' can be used to use several threads to make the compilation go faster, replace x with desired number of threads)
3. *sudo make install*

Stacks 2 should now be installed and can now be called like any other program in the terminal.

Dennis Larsson 09.07.2020 Universität Wien

Demultiplexing

The way we multiplex our library is through the usage of 3 sets of barcodes, 1 on the P2 adapter and 2 on the P1 adapter. the P1 adapter normally only has 3 barcodes, but in order to increase the level of multiplexing we ligate another four barcodes to the end of the P1 adapter. This means that we get 3 x 4 = 12 P1 adapters. When we demultiplex we need to separate the reads by one barcode at a time, which means we have to do three demultiplexing steps, for this purpose we refer to the index barcode (the barcode inside the adapter) as P11, and the inline barcode (at the end of the adapter and effective in the sequenced fragment) is referred to as P12.

When looking in the .bam file using *samtools* (samtools view NameOf.bam) We will find the P11 barcode in the field B2 and The P2 adapter in BC (it is opposite to how one would think intuitively from the numbers). The easiest way to confirm that this is the case is by using some linux commands and search for them, but first let's decipher the adapter tables in the pdf called adapter_for_RAD.pdf.

On the first page we have two tables, the first is the normal one and the second (starts with phos) is the reverse complement of the adapter. If we look closer on the first table, since the P1 is the forward adapter, we can see there are two areas that have a bolded text style. The first bolded string of letters is the index barcode and the second is the ligated inline barcode. The second page is similar to the first but for the P2 adapter and here we will look at the first table again (the second table is something else, I am not sure what, ask Ovidiu Paun if you are interested) and we will find the barcode in bolded letters.

Now if we want to check where our barcodes are in the bam file, we can use a set of simple linux commands to find out (make sure to have *samtools* installed, by using: *sudo apt-get install samtools* , in the terminal). First let's look for the P11 barcodes (index barcode on P1 adapter), which as we can see in the pdf are: TAATCTTA, CAGGACGT and GTACTGAC. Let look for the first using the below command:

```
samtools view CC58GANXX_5_20180212B_20180215.bam | head -n 100 | grep TAATCTTA
```

Where CC58GANXX_5_20180212B_20180215.bam is the name of my bam file, yours will be different. *Samtools* will stream the bam file to the terminal and we then pipe it (using "|") to the linux command *head* which will only print the first 100 lines that *samtools* streams out. These 100 lines are then piped to *grep* which will look for the string that follows (our barcode) in each of the strings. You will see a number of lines appearing on your screen, each line is one read from the bam file and in read we can find the barcode. Hopefully it should be in B2.

You might have noticed an extra letter after the barcode, this is just part of the adapter that appears in there since our barcode is eight characters but the bam file will always fill up to 9 characters in the B2 field, this is nothing to worry about. You might also have noticed that sometimes the barcode is found in the sequence, this is most likely a coincident, but can

also means that one fragment was so short that the part of the adapter was read in the sequencing part (say the fragment was only 70 bp long, then you would expect a large part of the adapter to be sequenced too, since it follow immediately after the fragment). It can also be due to adapter dimers that were sequenced (basically two adapters bonded with each other instead of to either end of the fragments, this can happen if too much adapter was added to too little DNA). These will be filtered out during the *process_radtags* step, since we set the program to require the sequence to start with the restriction site (after the P12 inline barcode).

You can now repeat this for one of the P2 adapters, and we will find a similar pattern, but now it will be found in the BC field. Finally when we use the second barcode on the P1 adapter (P12) we will find that it is at the very beginning of the sequence (mostly, again for the same reasons as above).

--- Advanced ---
If you want to check which types if barcodes are in your BC and B2 fields you can use *cut* to get just the barcode (*cut -f 12* is for BC which is the P2 barcode and 14 is for the P11 adapter, if you want the P12 adapter you need a command/script that will take the sequenced fragment using 10, then cut out the first six bases) and then use *sort* and *uniq* to get the different variants found:

samtools view CC58GANXX_5_20180212B_20180215.bam | cut -f 12 | head -n 1000 | sort | uniq -c

This will give you a summary of the first 1000 reads, but remove the *head* part to see all, but I will take a long time! You can also do another *sort* at the end to get the highest at the top then do *head* again to get only the 10 most common:

samtools view CC58GANXX_5_20180212B_20180215.bam | cut -f 14 | head -n 1000 | sort | uniq -c | sort -n -r | head

To access the P12 barcode we need the first 6 bases in the actual sequenced fragment. For this you can use this command:

samtools view CC58GANXX_5_20180212B_20180215.bam | cut -f 10 | head -n 1000 | cut -c1-6 | sort | uniq -c | sort -n -r | head

This will print the 10 most common P12 barcodes found (the first four should hopefully be your barcodes)

You can also use this to access the restriction site (should be TGCA, but will frequently be otherwise (10%)), which follows the first six bases in the barcode, using this command:

samtools view CC58GANXX_5_20180212B_20180215.bam | cut -f 10 | head -n 1000 | cut -c7-10 | sort | uniq -c | sort -n -r | head

Again it will show the 10 most common variants.

Now that we have determined that our barcodes are there, we can start the demultiplexing.

you will start off by making sure that you have all the correct barcode files. you require P2 barcodes and P11 barcodes for the first two steps. They will look like this:
P2 barcodes:
barcode_sequence barcode_name library_name
CGATGT 2s 2s
GATCAG 3s 3s
TGACCA 4s 4s
GCCAAT 6s 6s
ACTTGA 9s 9s
TAGCTT 10s 10s
CTTGTA 12s 12s

P1 barcodes:
barcode_sequence barcode_name library_name
TAATCTTA 506 506
CAGGACGT 507 507
GTACTGAC 508 508

Put this into two separate files called barcodes_P2 and barcodes_P11 respectively.

To run the first demultiplexing round (demultiplexing using the P11 barcode) using illumina2bam you first need to design the command you want to run in the terminal. This is a template for the command:
java -Xmx4g -Xms4g -jar /path/to/illumina2bam/dist/BamIndexDecoder.jar OUTPUT_DIR=/path/to/RAD_folder OUTPUT_PREFIX=RAD OUTPUT_FORMAT=bam BARCODE_TAG_NAME=B2 BARCODE_QUALITY_TAG_NAME=Q2 INPUT=/path/to/RAD_folder/my_sequences.bam BARCODE_FILE=/path/to/barcodes_P11 MAX_RECORDS_IN_RAM=6000000 CONVERT_LOW_QUALITY_TO_NO_CALL=true MAX_LOW_QUALITY_TO_CONVERT=20 MAX_MISMATCHES=1 CREATE_MD5_FILE=true COMPRESSION_LEVEL=9 METRICS_FILE=/path/to/RAD_folder/RAD_metrics.tab

java -Xmx4g -Xms4g -jar tells the ubuntu that we want to use java and that we want to use no more (*-Xmx4g*) and no less (*-Xms4g*) than 4g of RAM memory (tailor this as you see fit) and that the file we want to run is a .jar file. */path/to/illumina2bam/dist/BamIndexDecoder.jar* is naturally the path to the illumina2bam application BamIndexDecoder.jar, which will do the demultiplexing. *OUTPUT_DIR* should be followed by the path to your chosen output folder. *OUTPUT_PREFIX* is what your output files' name should start with. *OUTPUT_FORMAT* determines the format. *BARCODE_TAG_NAME* tells illumina2bam where it will find the barcode in the bam file. *BARCODE_QUALITY_TAG_NAME* tells illumina2bam where to find

the barcode quality in the bam file. *INPUT* should be the path to the input bam file. *BARCODE_FILE* gives the path to the barcode file we want to use. For the first run we want to use barcodes_P11. *MAX_RECORDS_IN_RAM* tell how much data can be processed in the RAM memory at any one time. *CONVERT_LOW_QUALITY_TO_NO_CALL* tells, if set to true, illumina2bam to convert sites in the barcode with low quality score to Ns. Ns will always count as mismatch. *MAX_LOW_QUALITY_TO_CONVERT* gives the phred score threshold to convert low quality sites to Ns. *MAX_MISMATCHES* tells how many mismatches are allowed in the barcode. *CREATE_MD5_FILE* creates a check code file to check for corruption, normally not used. *COMPRESSION_LEVEL* sets the compression level for the output file. *METRICS_FILE* sets the path where a metrics file will be created with various statistics.

After having modified your command, run it in the terminal and while it runs we will create three folders and three more commands to demultiplex using the P2 barcode. First create three folders called 506, 507 and 508 (use command *mkdir 506 507 508* in the output folder of illumina2bam while in the terminal). Then copy the first illumina2bam command and paste three times, one for each P11 barcode (506 etc) and modify the arguments so each command demultiplexes one of the three output files of the P11 demultiplexing. *OUTPUT_DIR* should be set to the respective folder for the given input bam file, which will be specified in the *INPUT* argument. For example, if the command will continue to demultiplex the output file that contain the 506 barcode, the *OUTPUT_DIR* should be to the 506 folder (example: */path/to/RAD_folder/506*) and the *INPUT* should be to the demultiplexed 506 output bam file (example: */path/to/RAD_folder/RAD#506.bam)*. Then *BARCODE_TAG_NAME* needs to be set to where the P2 barcode is, which is BC instead of B2. Same for *BARCODE_QUALITY_TAG_NAME*, which is set to QT instead of Q2. *INPUT* should be modified as mentioned earlier. *BARCODE_FILE* should now have the path to the P2 barcodes file (example: */path/to/barcodes_P2*). Finally *METRICS_FILE* should create a metrics file in the folder for the respective bam file (example: */path/to/RAD/506/RAD_506_metrics.tab)*. See template below for command to demultiplex 506 output bam file:

java -Xmx4g -Xms4g -jar /path/to/illumina2bam/dist/BamIndexDecoder.jar
OUTPUT_DIR=/path/to/RAD_folder/506 OUTPUT_PREFIX=RAD_506
OUTPUT_FORMAT=bam BARCODE_TAG_NAME=B2
BARCODE_QUALITY_TAG_NAME=Q2 INPUT=/path/to/RAD_folder/RAD#506.bam
BARCODE_FILE=/path/to/barcodes_P11 MAX_RECORDS_IN_RAM=6000000
CONVERT_LOW_QUALITY_TO_NO_CALL=true
MAX_LOW_QUALITY_TO_CONVERT=20 MAX_MISMATCHES=1
CREATE_MD5_FILE=true COMPRESSION_LEVEL=9
METRICS_FILE=/path/to/RAD_folder/RAD_metrics.tab

For the final step we will use *process_radtags* from Stacks 2. However the output from the second demultiplexing consist of 3 x 6 (or however many P2 barcodes you used) files. This means that in order to demultiplex them all we need 18 unique barcode files, since we need each individual name in the barcode files. To do this more easily we will run a python script and have it prepare the files and folders for us and also run the 18 iterations of the *process_radtags* script. But first we need to prepare a table of information for the python

script. The table needs to contain: the name of the individual, what barcode it has on the P12 adapter and what P11 and P2 barcodes it has on the P1 adapter. Usually we prepare a table containing this information during the library preparation, so this information should be easily copy and pasted from it. The final table should look something like this (the below list is obviously shortened):

SampleRAD1 P1_2 P1_1 P2
PorGMS195_1 ACTGAT 506 2
PorGMS182_2 TGACCA 506 4
PspGMS237_1 GAGTGG 507 3
PspGMS228_1 CTCATC 508 10

First a header with what info each column contains, then a row for each individual. Each column should be separated by a tab, not white space!
Where the P1_2 barcodes are:
ACTGAT
TGACCA
GAGTGG
CTCATC

See adapter_for_RAD.pdf to familiarize yourself with these second barcodes on the P1 adapter.

If you are using a different restriction enzyme than pstI, then you need to change that in the python script. Ask your local bioinformatician if you don't know how to do that.
The run the script with this command:
python3 /path/to/process_radtag_prep.py -o /path/to/RAD_folder/ -t /path/to/adapter_table

This is how the *process_radtags* command looks for P11 adapter 506 and P2 adapter 2s that will be automatically run by the python script *process_radtag_prep.py*:
process_radtags -f /path/to/RAD_folder/506/RAD_506#2s.bam -o /path/to/RAD_folder/506/2s -b /path/to/RAD_folder/506/2s/barcodes -e pstI -E phred33 -r -c -q -i bam -y gzfastq --barcode_dist_1 1 -s 20 -w 0.06

-f /path/to/RAD_folder/506/RAD_506#2s.bam is the argument which directs *process_radtags* to use the proper input file. *-o /path/to/RAD_folder/506/2s* is the output folder where the individual fastq files will be output. *-b /path/to/RAD_folder/506/2s/barcodes* is the path to the barcode which will be automatically generated by the *process_radtag_prep.py* script. *-e pstI* is the restriction enzyme used. *-r -c -q* tells *process_radtags* to clean and filter the read and not just demultiplex. *-i bam -y gzfastq* gives information on the format of the input (-i) and the output file (-o). *--barcode_dist_1 1* this means that we allow one mismatch. *-s 20 -w 0.06* makes *process_radtags* scan each read with a window size that is 6% of the read length and if at any point the bases within the window have an average phred score below 20, the entire read is dropped.

Once all of the *process_radtags* runs have finished you now have each individual demultiplexed into a gzipped fastq file.

To make it easier to move the files into one folder we use a few shell commands, while the terminal is in the output folder for the specific library (*/path/to/RAD_folder/*):
mkdir allSeq
To create a new folder, then:
mv -i ./506//*.fq.gz ./allSeq*
mv -i ./507//*.fq.gz ./allSeq*
mv -i ./508//*.fq.gz ./allSeq*
To move each file into the allSeq folder
The final thing to do is to collect the read counts for each individual, we do this using a bash script called collect_log.sh
It will find each log file and collect only the read count information for each indviduals and put it into a file called logAll. Make sure the script is in */path/to/RAD_folder/* and run in the terminal:
sh collect_log.sh

Now you have all the demultiplexed individuals in the folder allSeq and read count data in the text file logAll.

Now you can proceed with doing an assembly.