Schmutzi Workshop Documentation

Release 1.0

Alexander Peltzer

May 10, 2016

Contents

This is intended to provide a little hands on experience with schmutzi by G. Renaud et al and is solely designed to provide some more detailed step-by-step information on how to determine contamination using the tool. If you find bugs in this, I'm happy to fix them; if you find bugs in the tool itself, please use the projects GitHub repository to open issues for them here . This is _not_ my tool, but I happen to be one of the more frequent users of the method, thus this was basically writing up things I found out myself or with help from the developer(s).

Contents:

CHAPTER 1

Detection of contamination on mitochondrial data

schmutzi can be used to detect human contamination in mitochondrial data. In case you have enough reads mapping on to the mitochondrion reference genome, you can utilize the methods provided by *schmutzi* to automatically detect present contamination from other human sources automatically. This procedure is split into two parts, one performing a damage-based contamination estimation and a second one utilizing a database of mitochondrial allele frequencies. During this workshop, we will work on two human endogenous samples of undisclosed origin, that you will have to analyze yourselves: *Sample_A* and *Sample_B*. Of course, you can also transfer your own test cases to our system and then apply the methods taught in this course on these instead.

Warning: The provided test BAM files are only for testing purposes and should not be distributed further.

Note: You can run these methods and single steps manually but you could also run this in a more concise way instead by creating a simple bash script for your convenience. In case you'd like to do this, follow the simple template provided below:

```
#!/bin/bash
command1
command2
...
```

1.1 Sample preparation

1.1.1 Subsampling

In order to make these use cases computationally feasible, please do not use samples with more than 50x mitochondrial coverage or subsample the samples if you have more coverage than this.

Note: In case you use our provided sample datasets, you don't need to do any subsampling, as the samples we provide as use cases are small in size anyways.

For those of you who would like to use their own datasets, please apply *samtools view* and produce a subsampled version of your input data if the input file is too large for our course.

```
samtools view -b -s 0.2 input.bam -o output.bam
```

This should produce a rough 20% of your input file, which is taking randomly reads from the sample instead of taking an order set of reads from the input.

1.1.2 MD-Tagging

In order for *schmutzi* being able to access the data properly, we need to add **MD** tags to the BAM files. **MD** tags can be used by programs to perform e.g. SNP detection without a reference genome, as the tag contains information on which positions of the corresponding read there are matches, mismatches or indels. To add MD tags to your data, use the *samtools calmd* command:

```
samtools calmd -b Sample_A.bam ../ref/human_MT.fa > Sample_A.MD.bam
```

Warning: In order for this to work, you need to ensure to use the **same reference** that you used for mapping/creating your BAM file(s)!

1.2 Damage based *contDeam*

This can solely be used to determine contamination based on endogenous deamination. This means, if you use for example UDG treated data, that *contDeam* will tell you that your sample is severely contaminated (as it shows *no deamination* or at least *less contamination*). We are only using the default way of *contDeam*, a more complete documentation can be found on GitHub for your convenience.

First, run *contDeam* on each sample individually. As this produces quite an amount of output files during the iterative process, we should create a folder structure to store our output in a logical way.

```
mkdir -p Results/Sample_A/
mkdir -p Results/Sample_B/
```

This creates two folders in our current folder, making it possible to store all the output created by our methods to be applied in a logical way. Now, we can move on to use *contDeam*:

```
contDeam --library double --out Results/Sample_A/Sample_A --uselength --ref Ref/human_MT.fa RAW_BAMs/
contDeam --library double --out Results/Sample_B/Sample_B --uselength --ref Ref/human_MT.fa RAW_BAMs/
```

Note: You should make sure to use the proper commandline here: Specifying *single* for a double stranded library would not produce any meaningful results and thus render your estimation wrong potentially. Make sure to check prior using the command **which** kind of data you have here! Typically you do have double stranded data, but in case you are not certain that you have, you may want to check this with sequencing before.

This should produce something like this on your command line:

```
Reading BAM to set priors for each read ...
..  done
running cmd /projects1/tools/schmutzi/posteriorDeam.R Results/Sample_A/Sample_A.cont.deam Results/Sa
amination patterns"
null device
          1
Program finished succesfully
Files created:The plot of the posterior probability is Results/Sample_A/Sample_A.cont.pdf
The contamination estimate is here Results/Sample_A/Sample_A.cont.est
The configuration file is here Results/Sample_A/Sample_A.config
```

You may have a look now at the output of this initial contamination estimation run. How do your samples seem to look like for *Sample_A* and *Sample_B* ? To check this, you can have a look at the output initially generated using e.g. *cat*:

```
cat Results/Sample_A/Sample_A.cont.est
0 0 0.95
cat Results/Sample_B/Sample_B.cont.est
0 0 0.005
```

This means, that based on the deamination patterns both samples look relatively clean with a initial lower estimate of 0 % contamination, an average of 0% and an upper estimate of 95% for the first and 0.05% for the second sample. Relatively means in this case, that *Sample_B* looks clean completely, whereas *Sample_A* shows an initial high contamination of 95%.

However, you can't trust these results individually if:

1. You have less than 500 Million reads (which is very rarely the case)
2. You don't have enough deamination, less than 5% won't work for example (Attention: UDG treatment!)
3. Very little / No deamination of the contaminant fragments
4. (Independence between 5' and 3' deamination rates is required for the Bayesian inference model)

This method could be used for running contamination estimates on both nuclear and mitochondrial data in general, however I would recommend applying DICE for samples with nuclear data in general or perform other tests (X-chromosomal contamation test, looking forward to Stephan Schiffels introduction on this). I will generate a HowTo for DICE in the upcoming weeks, following the *schmutzi* manual here, too.

1.3 Mitochondrial based *schmutzi*

Now that we have successfully estimated contamination using *deamination patterns*, we will proceed by using allele frequencies on mitochondrial data, too. *schmutzi* comes with a database of 197 allele frequencies accompanied by an Eurasian subset of allele frequencies, that can be used for our analysis.

Note: If you would like to test e.g. for contamination on other organisms, e.g. some other mammals and you do possess enough datasets to generate such a database, you can also generate these frequencies yourself. For more details, follow Gabriel Renaud's HowTo here .

Now let's run the *schmutzi* application itself. Prior to doing this, we need to index our MD tagged BAM file first:

```
samtools index RAW_BAMs/Sample*.MD.bam
schmutzi --ref Ref/human_MT.fa --t 8 Results/Sample_A/Sample_A /projects1/tools/schmutzi/alleleFreqM
schmutzi --ref Ref/human_MT.fa --t 8 Results/Sample_B/Sample_B /projects1/tools/schmutzi/alleleFreqM
```

Warning: Make sure to use the correct **freqs** folder, or the tool will crash.

The whole process might run for a couple of minutes, mainly depending on the number of CPU cores `--t 8` you assigned your estimation process.

Warning: Do not use more CPU cores than available, or the whole system might get unstable. *schmutzi* can be pretty heavy in terms of memory / CPU usage, taking up a lot of your systems computational capacities.

In the end, this should produce some output:

```
Reached the maximum number of iterations (3) with stable contamination rate at iteration # 5, exiting
Iterations done
Results:
      Contamination estimates for all samples        : Results/Sample_A/Sample_A_final_mtcont.out
      Contamination estimates for most likely sample : Results/Sample_A/Sample_A_final.cont
      Contamination estimates with conf. intervals   : Results/Sample_A/Sample_A_final.cont.est
      Posterior probability for most likely sample   : Results/Sample_A/Sample_A_final.cont.pdf
      Endogenous consensus call                      : Results/Sample_A/Sample_A_final_endo.fa
      Endogenous consensus log                       : Results/Sample_A/Sample_A_final_endo.log
      Contaminant consensus call                     : Results/Sample_A/Sample_A_final_cont.fa
      Contaminant consensus log                      : Results/Sample_A/Sample_A_final_cont.log
      Config file                                     : Results/Sample_A/Sample_A.diag.config
Total runtime 248.527676105499 s
```

Running a small `cat` again to check the results of the contamination analysis:

```
cat Results/Sample_A/Sample_A_final.cont.est
0.99 0.98
```

This means, we have between 98%-99% contamination in *Sample_A*, making it useless for any downstream analysis.

Doing the same with our other sample now:

```
cat Results/Sample_B/Sample_B_final.cont.est
0.01  0 0.02
```

Which looks good - this sample seems safe to be used for downstream analysis, as it shows between 0 (low) - 1% (avg) - 2% (high) contamination estimate.

1.4 Output interpretation

1.4.1 `EST` Files

schmutzi generates a couple of output files that can be used to determine whether your samples are clean or not. The table above in *output_files* describes what kind of output to expect on a successful run of *schmutzi*. The most important file is the one with ending `est` as it provides the contamination estimate for your data.

The content of the `est` file should look like this:

```
X  Y Z
```

Where X is your average estimate, Y your lower estimate and Z your upper estimate. In some cases you will only see two numbers appearing, meaning that this is your upper and lower bounds respectively. It depends on your kind of analysis you'd like to perform whether you want to include edge cases with e.g. upper contamination of 3% estimate or not.

In the case you performed a full evaluation using both the *contDeam* and the *schmutzi* tools, you will see several `est` files, containing estimates in each iteration. *schmutzi* iteratively refines the consensus called by the *mtCont* subprogram, meaning that it will provide intermediate results in these files, numbered ascendingly from 1, 2 to *final*.

```
Sample_B_1_cont.3p.prof
Sample_B_1_cont.5p.prof
Sample_B_1_cont.est
Sample_B_1_cont.fa
Sample_B_1_cont.freq
Sample_B_1_cont.log
Sample_B_1_endo.3p.prof
Sample_B_1_endo.5p.prof
```

```
Sample_B_1_endo.fa
Sample_B_1_endo.log
Sample_B_1_mtcont.out
Sample_B_2_cont.3p.prof
Sample_B_2_cont.5p.prof
Sample_B_2_cont.est
Sample_B_2_cont.fa
Sample_B_2_cont.freq
Sample_B_2_cont.log
Sample_B_2_endo.3p.prof
Sample_B_2_endo.5p.prof
Sample_B_2_endo.fa
Sample_B_2_endo.log
Sample_B_2_mtcont.out
```

The first *est* file is based on the *contDeam* results (on the damage patterns), whereas the others are based on the iterative process used when estimating contamination using the mt database.

1.4.2 `FA` Files

These contain for both the endogenous part as well as the contaminant part the respective consensus sequences produced. Note that this has not been filtered at all and should therefore only be used for determining contamination and not for any downsteam analysis.

1.4.3 `Log` Files

These files are the raw output schmutzi produces using a bayesian method to infer the endogenous part of your sample. If you want to use downstream analysis on your data, e.g. calling haplotypes on your mitochondrion, you should apply some filtering on your dataset, which we will do in the next part of our analysis journey.

1.5 Consensus Calling for Downstream analysis

1.5.1 Filtered Consensus Calling

In order to get filtered calls, e.g. no SNPs for regions covered with only a single read, one should apply some filtering criteria:

```
/projects1/tools/schmutzi/log2fasta -q 20 Results/Sample_A/Sample_A_final_endo.log > Results/Sample_A
/projects1/tools/schmutzi/log2fasta -q 30 Results/Sample_A/Sample_A_final_endo.log > Results/Sample_A
/projects1/tools/schmutzi/log2fasta -q 20 Results/Sample_B/Sample_B_final_endo.log > Results/Sample_B
/projects1/tools/schmutzi/log2fasta -q 30 Results/Sample_B/Sample_B_final_endo.log > Results/Sample_B
```

It is advisable to choose these parameters increasingly, e.g. with a range of `-q 20`, `-q 30`, `-q 40`, `-q 50` and check whether you still have enough diagnostic positions in the end.

A good way to determine whether we have a lot of undefined positions relative to our used reference genome is by iteratively running several times the above command, to find an acceptable threshold between filtering and reserving enough information for the analysis.

```
tr -d -c 'N' < Results/Sample_A/Sample_A_q20.fasta | awk '{ print length; }'
16,569
tr -d -c 'N' < Results/Sample_A/Sample_A_q30.fasta | awk '{ print length; }'
16,569
```

As you see, for our *Sample_A*, the output doesn't change, meaning we already have pretty high numbers of Ns in our output, meaning they have been filtered out. As you might recall, this is totally fine, since *schmutzi* declared this sample to be heavily contaminated anyways. Therefore we repeat this for *Sample_B* now to see if this behaves better:

```
tr -d -c 'N' < Results/Sample_B/Sample_B_q20.fasta | awk '{ print length; }'
90
tr -d -c 'N' < Results/Sample_B/Sample_B_q30.fasta | awk '{ print length; }'
351
```

As you can see we only have 90 bases not defined with a pretty decent filtering parameter already. When going down to filtering even more conservative with `q=30`, you can see that we are loosing even more positions but still have a reasonable amount of diagnostic positions. I leave it up to you to figure out a good threshold when you loose more than you gain in the end.

1.5.2 Unfiltered Consensus Calling

For modern samples we can use the application `endoCaller` coming with schmutzi instead, as we don't want to run contamination checks on this. This can be done using:

```
/projects1/tools/schmutzi/endoCaller -seq youroutput.fasta -log outputlog.log reference.fasta input.k
```

This will produce a consensus call, which is **unfiltered**. To test what kind of difference this makes, you may for example try running this method on one of our ancient samples comparing the output to a filtered output FastA directly. You will observe that especially in lower coverage situations, the `endoCaller` incorporates SNPs based on e.g. a coverage of 1 or low quality regions, whereas the filtering approach as defined in *Filtered Consensus Calling* .