SECAPR pipeline exercise

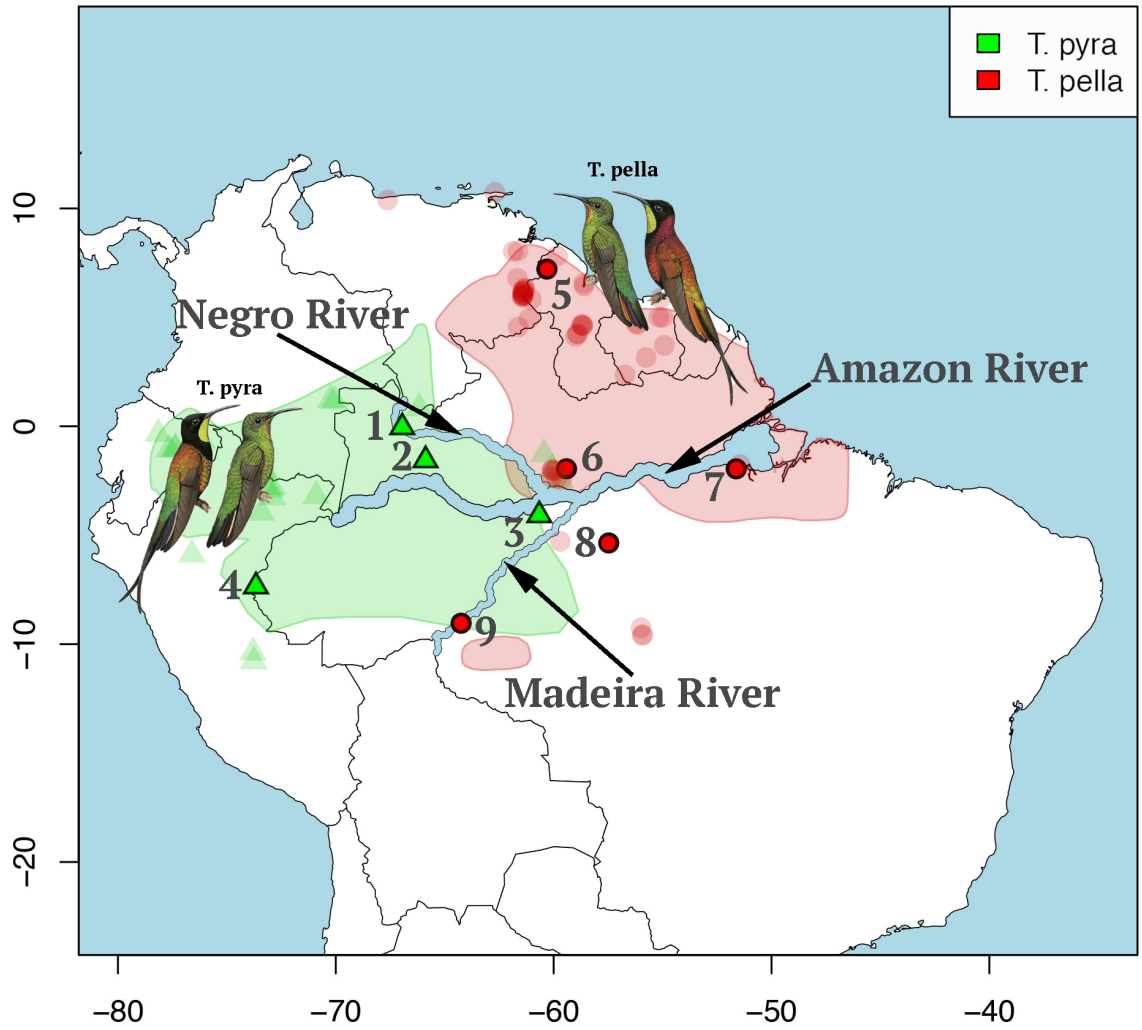
written by Tobias Andermann (tobias.andermann@bioenv.gu.se)

Background

Many of the common bioinformatic tools work on a per-sample basis. You can imagine that if you have hundreds of samples that you all want to process, it would become a quite painstaking process. For this purpose there exist computational pipelines which enable you to process many samples jointly, making the whole workflow more user-friendly. These pipelines also help to produce a consistent, documented, and therefore reproducible workflow, allowing you to process your target capture data even if you are not a trained bioinformatician. If you are completely new to Illumina data or target capture data in particular, make yourself cozy and read through this review article, which explains the whole process from project planning to producing, processing, and analyzing your data. Not now though, because we are here to get our hands dirty with some real data processing.

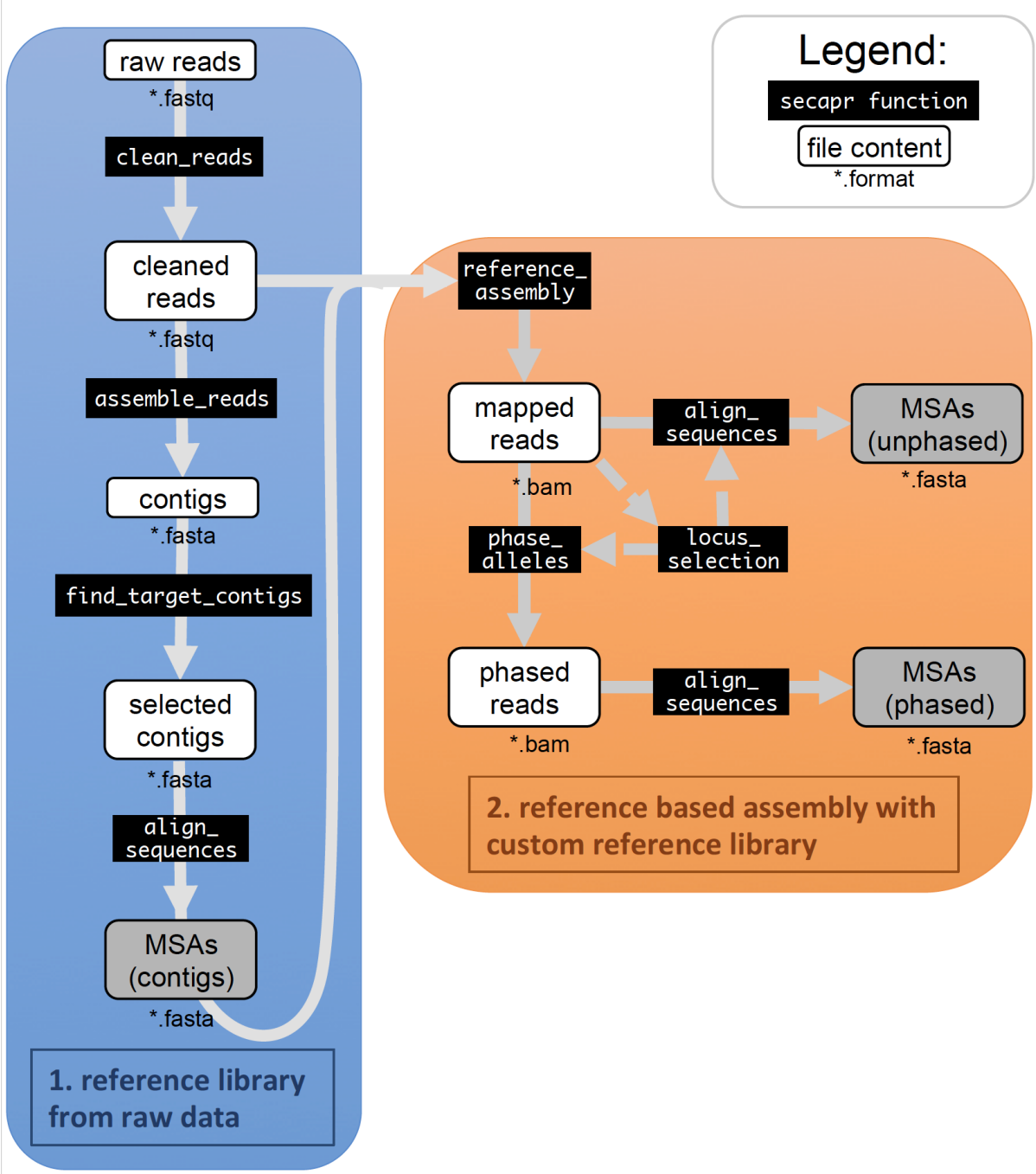
We are going to use the SECAPR pipeline (see SECAPR's original publication here) on a dataset of **Ultraconserved Elements (UCEs)** that were sampled for the hummingbird genus *Topaza* in South America. This dataset consists of sequence data for about 2500 individual loci, which were enriched prior to sequencing using target capture.

It should be mentioned that there also is the Phyluce pipeline, which is made and fine-tuned specifcially for UCE data. Usually you would use Phyluce for any UCE dataset. However in this case we will use SECAPR, since it is a more generalized pipeline that may come in handy for your non-UCE target capture dataset and therefore might be more useful for you to learn.



For this genus it is not clear if the existing morphological species assignments are justified and if there might be cryptic species within these morphospecies. We want to use this UCE dataset to generate multiple sequecnce alignments (MSAs), to eventually estimate a phylogeny (species tree) of these samples and define coalescent species.

Below is an outline of the main functions and workflow of the SECAPR pipeline. This tutorial will cover all these processing steps, producing different sets of MSAs with different properties.



Installation and setup

Set up conda installation manager

Conda is a software and environment manager, that makes installation of new software and of required dependencies very simple and straightforward. If you don't already have conda installed on your computer, you can rather quickly and easily install the light-weight version Miniconda.

Download and install the Python 3.8 version of Miniconda for your operating system.

Install secapr

The SECAPR pipeline is available as a conda package and can be installed together with all its dependencies using one single command. Since there are currently some installation issues on some systems with the SECAPR version hosted on the conda server, we instead are installing the secapr version directly from my GitHub repo. For this we have to do the extra step of first downloading the installation script with `wget` and then we will install the pipeline by executing the installation script with `sh`.

If you don't have `wget` installed on your computer you can install it with `conda install wget`.

```
wget https://raw.githubusercontent.com/AntonelliLab/seqcap_processor/master/
sh install_secapr_env.sh
```

If you can't get `wget` to run on your machine, you can instead access the file manually here and simply copy its content into your command line terminal.

SECAPR has a lot of dependencies which all need to be downloaded, so installation may take a few minutes. Once it's done with the installation, connect to the newly created environment `secapr_env`:

```
conda activate secapr_env
```

(Sometimes this command throws an error, in which case `source activate secapr_env` should do the trick.)

Get the data

To keep processing times and storage requirements at bay, I selected only the following four samples for this tutorial:

- T_pyra1
- T_pyra3
- T_pella5
- T_pella9

You can download the raw **fastq data** for this tutorial from Google Drive. That shared folder folder also includes an `adapter_info.txt` file with the **Illumina adapter and barcode information** for each sample, which you will need for adapter trimming. Also you can find the `Tetrapods-UCE-2.5Kv1.fasta` file there, which contains the **bait sequence for each targeted locus** (1 bait per locus).

Create a new folder on your computer with a name of your chosing as your working directory for this workshop. Unzip the downloaded folder, and move it into that working directory. All of the commands in this tutorial are assuming that you are using the command line, located at your working directory, which should contain the downloaded folder `pipeline_exercise`. Therefore make sure you `cd` to your working folder in your command line terminal. If you don't know how to do this, or don't know what the command line is, check out this basic and very helpful command line tutorial until you feel

like you understand the basic purpose and how to navigate through your file system (it is time well spent, since most bioinformatic programs are usually only available as command line programs).

Running SECAPR

Connect to secapr_env

Now we will start using the SECAPR pipeline to process our samples. Before you start make sure you are connected to the `secapr_env`:

```
conda activate secapr_env
```

To see the available SECAPR functions you can run the `secapr -h` command which will print the help page:

```
secapr -h
```

You will see a list of the available SECAPR functions. You can open the specific help page of each function by typing `secapr FUNCTION -h` (replace `FUNCTION` with the name of the function that you want to learn more about), e.g.:

```
secapr clean_reads -h
```

In case anything is unclear, you can check the more exhaustive explanations in the SECAPR documentation

---

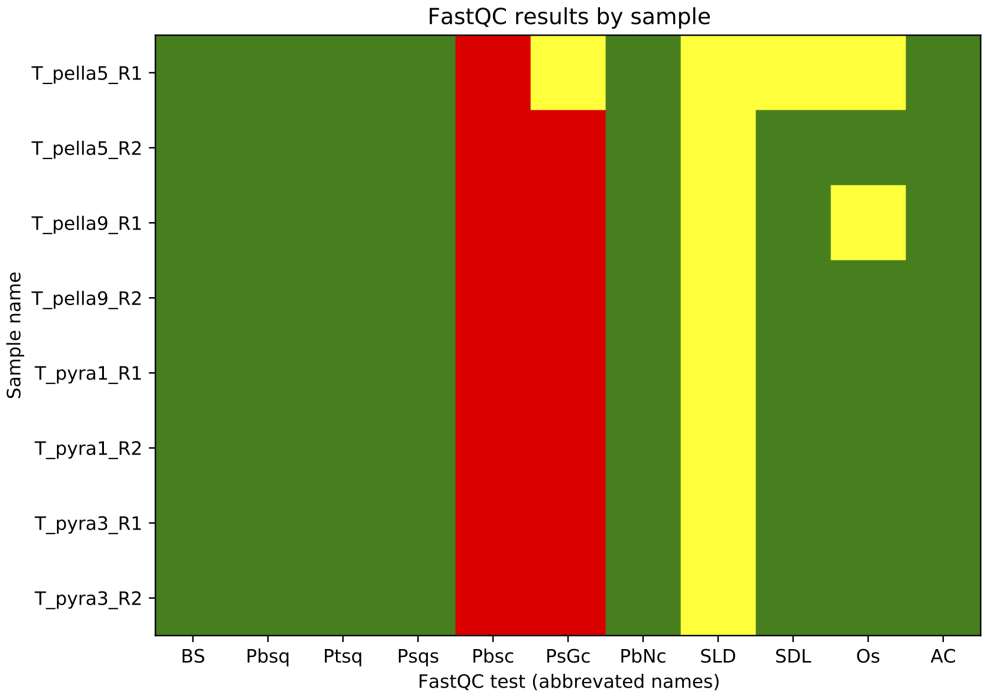
Trimming and filtering

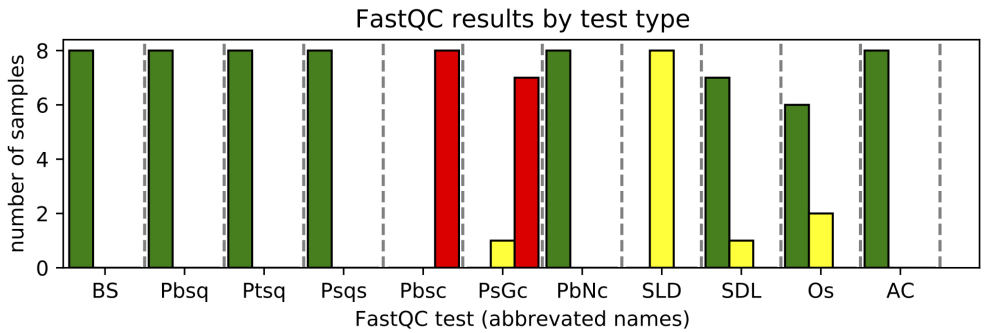
First step is running a quick quality check with `secapr quality_check`. This will run FastQC on all of your samples and produce some additional quality test results overview.

```
secapr quality_check --input pipeline_exercise/fastq_raw/ --output pipeline_
```

Inspect the created output folder at `pipeline_exercise/fastqc_results/raw`. Besides the FastQC quality reports, there are some summary plots created by SECAPR that help you get an overview of all your samples and whether or not they passed which tests.

My output looks like this:





The first plot shows which tests (x-axis) failed for which samples (y-axis). The other summary plot shows the same data in a slightly different way, making it easier to identify which stats need improvement over all (this overview becomes more useful when runnning lots of samples). Also take your time to explore some of the individual sample quality report by opening one of the html files in the output folder. Feel free to ask if you are wondering about some of the stats presented there.

It looks like the files will require some cleaning. In particular it is important and best practice to trim any remaining adapter contamination, even though in this case it does not show up as being problematic. Below it the `secapr clean_reads` command with default settings for the cleaning parameters. Run `secapr clean_reads –h` to see all available options and decide on a proper setting for your files, based on your quality test results. The adapter and barcode information is stored in the `adapter_info.txt` file. We need to tell the `secapr clean_reads` function if reads are single or double indexed (i.e. if they have a barcode in both adapter ends or only in one). This is information you can find from the library preparation protocol of your samples, and it depends on which adapter/barcode kit was used. In this case we worked with single-indexed adapters, i.e. only containing one barcode sequence each.

```
secapr clean_reads --input pipeline_exercise/fastq_raw/ --config pipeline_e
```

Run the `secapr quality_check` function again to test the quality of the cleaned files and in case you are not content with the quality test results, change the `secapr clean_reads` settings until the

results look good. When running the quality_check you can just provide the output folder from the clean_reads function as input, containing all cleaned samples.

Once you are happy with your trimmed reads, you can delete the `fastq_raw` folder containing the raw reads, to free up some storage space. Of course you would not do this with your real data, it is very important to keep an untouched and backed-up copy of your raw (as well as your cleaned) fastq files.

De novo contig assembly

Now where we have clean reads we can run our de novo contig assembly with the `secapr assemble_reads` function. In SECAPR you can choose between using the SPADES or the ABYSS assembler. You can choose what you prefer, but I will use ABYSS in the example. When using SPADES you can provide a list of kmer values to run, whereas ABYSS will only accept one kmer value. The assembly of these 4 samples takes about 2h on a single core.

```
secapr assemble_reads --input pipeline_exercise/cleaned_trimmed_reads --outp
```

If you don't want to wait until the de novo assembly is finished you can download my precompiled contigs here, which I created with a kmer value of 35.

Extracting target contigs

Now we will use the bait reference file (`Tetrapods-UCE-2.5Kv1.fasta`) to identify those contigs that represent our target loci. For this we can use the `secapr find_target_contigs` function, which also allows us to filter those contigs hitting multiple loci or/and those loci for which we have multiple contig matches. Since the former case is often not an issue, those contigs that match multiple neighbouring reference loci are not discarded by default, but are kept for each locus that they were matched to. Thos loci on the other hand that have multiple contig matches are discarded by default, as these may represent paralogous loci for which we have multiple assembled contig sequences, making it difficult to decide which one to pick for each sample. You can choose to include them by using the `--keep_paralogs` flag, in which case SECAPR will export the longest contig match for each of these loci.

As always, check out the available options by calling the help function: `secapr find_target_contigs -h`. You will see that you can also adjust your blast settings to be more conservative or more relaxed, using the `--min_identity` and `--seed_length` flags. This can help to avoid paralogs or to get a higher yield of target contigs if you seem to not get many blast hits.

```
secapr find_target_contigs --contigs pipeline_exercise/contigs --reference p
```

To evaluate how many target contigs you retrieved for each sample, SECAPR offers a handy plotting function that shows all loci that were recovered for each sample in one summary plot. To produce this plot you can use the `secapr plot_sequence_yield` function.

```
secapr plot_sequence_yield --extracted_contigs pipeline_exercise/target_cont
```

(Click here for a high resolution version of the plot.)

In this plot you see an overview of all the ~2,500 loci targeted in this dataset (x-axis), and whether or not a contig was assembled for these loci (blue = "contig found and extracted", white = "no matching contig"). You can see that this was a rather successful target capture experiment, as we have recovered sequences for the vast majority of loci for each sample, with only very few loci missing.

Produce Multiple Sequence Alignments from contigs

With SECAPR it is very easy to create Multiple Sequence Alignments (MSAs). You can use the `secapr align_sequences` for this purpose and provide it the joined fasta file resulting from the previous step, which contains the extracted target contigs from all samples (`extracted_target_contigs_all_samples.fasta`).

The alignment function will only produce an alignment for loci where sequences for at least 3 samples were recovered. So be prepared that some of your loci will not produce an alignment, in case the contig for a given locus could not be recovered in 2 or more of our 4 samples.

If you have multiple processing cores available on your machine, you can add the `--cores NUM` flag to the command below and replace `NUM` with the number of cores you want to parallelize the computation of alignments on.

```
secapr align_sequences --sequences pipeline_exercise/target_contigs/extracte
```

If you don't want to wait you can download the finished alignments here.

It is always good to have a look at some of the produced alignments and see if they look good overall. I recommend the Aliview alignment viewer, which is very lightweight and handy, and can be downloaded here. In case you are not happy with your alignments (e.g. they are too gappy or don't properly align), there is a lot you can adjust in the alignment settings, check all available options with `secapr align_sequences -h`.

You can see an overview over which loci SECAPR created alignments for by using the plotting script again, but this time adding the `--alignments` flag and the path to the folder with the contig MSAs.

```
secapr plot_sequence_yield --extracted_contigs pipeline_exercise/target_cont
```

There is also a helpful little function that adds dummy sequences to your alignments that are missing one or several of your samples. This helps preparing your data for some phylogenetic methods that require each alignment containing the same samples, such as e.g. BEAST. This function will add strings of `NNNNNN` as the sequence for these samples.

```
secapr add_missing_sequences --input pipeline_exercise/alignments/contig_al
```

Reference mapping

Now we will map the reads against a library of reference sequences using the `secapr reference_assembly` function. This reference library can **either be a fasta file of your choosing**, or it can be based on the target contigs we just assembled for each sample. In the latter case you can choose whether or not you want to do this on a **sample-specific** basis, i.e. map the reads of each sample against the contig sequences that were extracted for that sample, or alternatively you can map the reads of each sample against the **consensus sequences of the target MSAs** containing all samples recovered contigs at each locus. The latter allows to create e.g. a genus or family specific reference library, representing the consensus of all sequenced samples in the dataset. Be aware that when usinng the sample-specific approach, you won't get any reference assemblies for loci that lack an extracted contig for the given sample, leading to different loci missing for different samples, while the alignment-consensus approach creates reference assemblies for the same loci for all samples.

The choice of the reference library is controlled via the `--reference_type` flag (see the reference_assembly help function). Once finished with mapping, the `secapr reference_assembly` function will automatically create a IUPAC consensus sequence from the BAM file. The `--min_coverage` flag controls how much read coverage is required to make a basecall for that consensus sequence. Positions supported by fewer reads will be coded as `N`.

```
secapr reference_assembly --reads pipeline_exercise/cleaned_trimmed_reads --
```

This command will take around 20-30 minutes to finish. If you don't have the patience to wait, you can download the reference assembly results here.

There is a lot to explore in the resulting output files. The insteresting nuggets here are the `.bam` files located in each sample subfolder. You can view them using the Tablet BAM viewer, which you can download here. Once installed, open Tablet, go to the `mapped_reads` output folder, open one of the sample subfolders, mark the `.bam` and the `.bam.bai` files, and drag and drop them into the open Tablet window. On the left you will see an overview of the different mapped loci. You can click on each locus and see the reads that were mapped to it by sliding the box to the right across the locus. Can you find heterozygous sites in some of the loci? Any read errors? In case I forget, please remind me to look at one of these files together during the workshop to explore and discuss these things a bit.

You can use the `secapr plot_sequence_yield` function again to plot the average read coverage for each locus in comparison to the contig retrieval and alignment stats. The additional flag `--coverage_norm NUM` defines the maximum coverage for the color map. Adjust this value to see more detail (e.g. if most of your loci have a coverage of 5-10, set `NUM` to 10, if it's between 5-50, set it to 50).

```
secapr plot_sequence_yield --extracted_contigs pipeline_exercise/target_cont
```

Now let us build MSAs from the IUPAC consensus sequences of your mapped-read BAM files. These MSAa have the following advantages over the contig MSAs that we created earlier: a) they are not chimeric, i.e. they do not combine SNPs from multiple alleles in the same sequence, but instead code those sites as ambiguous (`N`); b) they offer more control over the support for each base call by setting read-depth thresholds; c) they are expected to be longer on avarage, because the reads are mapped to a sample or genus-specific reference library generated through de novo assembly (rather than using more distant reference sequences to extract target contigs, like e.g. those used for the bait set).

```
secapr align_sequences --sequences pipeline_exercise/mapped_reads/joined_unp
```

If you don't want to wait, you can download the finished alignments here.

Locus selection

The SECAPR pipeline has a function that helps you to select a subset of your loci, based on the read coverage of these loci (`secapr locus_selection`). This can be a helpful tool when you want to only continue processing loci exceeding a certain average read coverage. Or it can be a good tool for data reduction when planning to work with phylogenetic methods that are limited to smaller numbers of loci because of high computation times (such as *BEAST). Here we select the 100 best loci from our total dataset of 2500 loci (those with the best read coverage). Instead we could also use the `--read_cov` flag to set a minimum average read coverage for any locus to be selected.

```
secapr locus_selection --input pipeline_exercise/mapped_reads --output pipel
```

You can now build MSAs from the IUPAC consensus sequences of your selected loci.

```
secapr align_sequences --sequences pipeline_exercise/selected_loci/joined_fa
```

Allele phasing

Now we will phase our selected loci and produce MSAs containig allele sequences with `secapr phase_alleles`. This allows us to sort out heterozygous sites and map them on two separate sequences. The applied phasing algorithm works based on read-connectivity and is therefore most useful and accuracte for loci with good coverage with rather evenly spaced heterozygous sites, allowing the algorithm to decide which nucleotides are located on the same allele. Having paired end data in combination of decent length fragments greatly helps with this task, as it allows bridging over less variable or less covered parts of the locus. Decent length fragments in this context may be between 400-800 bp, i.e. on the longer length-range of suitable fragment lengths for Illumina seqeuncing, which can be controlled by the lab-protocol applied when fragmenting the DNA molecules.

```
secapr phase_alleles --input pipeline_exercise/selected_loci --output pipel
```

Produce allele sequence MSAs.

```
secapr align_sequences --sequences pipeline_exercise/allele_sequences_selec
```

Add missing species again to have complete alignments that all contain the same number of sequences.

```
secapr add_missing_sequences --input pipeline_exercise/alignments/selected_
```

Further steps

Now you have produced different set of MSAs which you can use for phylogenetic inference. With these tools at hand you will be able to process almost any target capture dataset. You could now produce individual gene trees from the alignments and then estimate the most likely species tree, using summary coalescent methods. Alternatively you could also co-estimate gene trees and species trees using *BEAST. Here you can find a tutorial for running allele MSAs like the one we produced here in *BEAST.