reference_assembly

March 14, 2018

1 Reference-based assembly

In the previous steps we have gone through cleaning and trimming of the reads, de-novo assembly of contigs and the alignment of those contigs that represent our target sequences. Now we will use those contigs in order to generate new reference libraries and assemble the reads with the hekp of those reference sequences (=reference-based assembly vs. de-novo assembly in the previous steps). Why doing an assembly again and on top of that, why are we assembling a new reference library, you may wonder. There are several reasons for the steps that follow:

1. **Sensitivity**: So far we have been using one reference library (the bait sequence set or equivalent) for finding matching sequences across all of our samples. While this may be fine if all your organisms are closely related (e.g. within the same genus), in all other cases you may loose many sequences because they are not similar enough to the reference sequence. Not only does this lead to a lower sequence turn-out but it also biases your results toward those seqeunces most similar to the reference sequence. The soultion to this is to use the sequence data we already assembled in the previous steps to create family-, genus- or even sample-specific reference libraries

2. **Intron/Exon structure**: Another reason for creating a new reference library is that you most likely have a set of exon sequences that were used to design the RNA baits for sequence capture. The more variable introns in between are not suitable for designing baits (too variable) but are extremely useful for most phylogenetic analyses (more information). There is a good chance that our contigs will contain parts of the trailing introns or in the best case even span across the complete intron, connecting two exon sequences. This is why we want to use these usually longer and more complete seqeunces for reference-based assembly, rather than the short, clipped exon sequences (**see image below**).

3. **Allelic variation**: Remapping the reads in the process of reference-based assembly will make visible the different alleles at a given locus. We will be able to see how many reads support allele a and how many support allele b and will have a better handle on separating sequencing errors from true variation (by assessing coverage). Further looking at the results of reference-based assembly can help distinguishing paralogues from allelic variation (if too many differences between the variants within one sample in one locus, it may likely be a case of paralogy).

4. **Coverage**: Reference-based assembly will give you a better and more intuitive overview over read-depth for all of your loci. There are excellent visualization softwares that help interpret the results.

```
In [26]: from IPython.display import IFrame
         print('Reference-based assembly is more powerful when using contigs as ref
         print('rather than the exon sequences from the bait-design step, as it spa
         print('accross adjacent intron sequences.')
         IFrame("../../images/exon_vs_contig_based_assembly.pdf", width=600, height
```

```
Reference-based assembly is more powerful when using contigs as reference
rather than the exon sequences from the bait-design step, as it spans
accross adjacent intron sequences.
```

```
Out[26]: <IPython.lib.display.IFrame at 0x1023901d0>
```

1.1 Creating new reference libraries

In the best scenario you have enough loci that could be recovered for all taxa. In that case the easiest and most efficient solution is to simply extract the contig sequences for all loci for each sample separately and use these as the new sample specific reference library. In any other case it is recommendable to split your samples by genus or family and create a genus/family specific reference sequence for each locus. Using the `--reference-type` flag you can decide which of these options you want to choose. The program also gives you the opportunity to provide a user-generated reference library. In our example data e.g. all samples belong to the same genus, so we decide to use the `alignment-consensus` option, which creates a consensus sequence of the alingment of all samples contig sequences for each locus. If you want to generate sample specific reference libraries, you need to first make a selection of loci that could be assembled for all loci (otherwise you may be lacking reference sequences for some samples).

1.2 Reference-based assembly

There are many additional flags available in order to control the settings for reference-based assembly step. The default settings are chosen to match the average sequence capture dataset. However, it is recommendable to check the resulting `.bam` files in a viewer (such as e.g. Tablet) and rerun the reference-based assembly until it results in a decent coverage with few-none mismatched reads and unnecessary gaps. To get an overview over the user-options type:

```
In [30]: %%bash
         source activate secapr_env
         secapr reference_assembly -h
```

```
usage: secapr reference_assembly [-h] --reads READS
                                 [--reference_type {alignment-consensus,sample-spec
                                 --reference REFERENCE --output OUTPUT
                                 [--keep_duplicates]
                                 [--min_coverage MIN_COVERAGE] [--k K] [--w W]
                                 [--d D] [--r R] [--c C] [--A A] [--B B]
                                 [--O O] [--E E] [--L L] [--U U]

Create new reference library and map raw reads against the library (reference-
based assembly)
```

```
optional arguments:
  -h, --help            show this help message and exit
  --reads READS         Call the folder that contains the trimmed reads,
                        organized in a separate subfolder for each sample. The
                        name of the subfolder has to start with the sample
                        name, delimited with an underscore [_] (default output
                        of clean_reads function).
  --reference_type {alignment-consensus,sample-specific,user-ref-lib}
                        Please choose which type of reference you want to map
                        the samples to. "alignment-consensus" will create a
                        consensus sequence for each alignment file which will
                        be used as a reference for all samples. This is
                        recommendable when all samples are rather closely
                        related to each other. "sample-specific" will extract
                        the sample specific sequences from an alignment and
                        use these as a separate reference for each individual
                        sample. "user-ref-lib" enables to input one single
                        fasta file created by the user which will be used as a
                        reference library for all samples.
  --reference REFERENCE
                        When choosing "alignment-consensus" or "sample-
                        specific" as reference_type, this flag calls the
                        folder containing the alignment files for your target
                        loci (fasta-format). In case of "user-ref-lib" as
                        reference_type, this flag calls one single fasta file
                        that contains a user-prepared reference library which
                        will be applied to all samples.
  --output OUTPUT       The output directory where results will be safed.
  --keep_duplicates     Use this flag if you do not want to discard all
                        duplicate reads with Picard.
  --min_coverage MIN_COVERAGE
                        Set the minimum read coverage. Only positions that are
                        covered by this number of reads will be called in the
                        consensus sequence, otherwise the program will add an
                        ambiguity at this position.
  --k K                 If the part of the read that sufficiently matches the
                        reference is shorter than this threshold, it will be
                        discarded (minSeedLen).
  --w W                 Avoid introducing gaps in read that are longer than
                        this threshold.
  --d D                 Stop extension when the difference between the best
                        and the current extension score is above |i-j|*A+INT,
                        where i and j are the current positions of the query
                        and reference, respectively, and A is the matching
                        score.
  --r R                 Trigger re-seeding for a MEM longer than
                        minSeedLen*FLOAT.
  --c C                     Discard a match if it has more than INT occurence in
                            the genome
  --A A                     Matching score. Acts as a factor enhancing any match
                            (higher value makes it less conservative = allows
                            reads that have fewer matches, since every match is
                            scored higher).
  --B B                     Mismatch penalty. The accepted mismatch rate per read
                            on length k is approximately: {.75 * exp[-log(4) *
                            B/A]}
  --O O                     Gap opening penalty
  --E E                     Gap extension penalty
  --L L                     Clipping penalty. During extension, the algorithm
                            keeps track of the best score reaching the end of
                            query. If this score is larger than the best extension
                            score minus the clipping penalty, clipping will not be
                            applied.
  --U U                     Penalty for an unpaired read pair. The lower the
                            value, the more unpaired reads will be allowed in the
                            mapping.
```

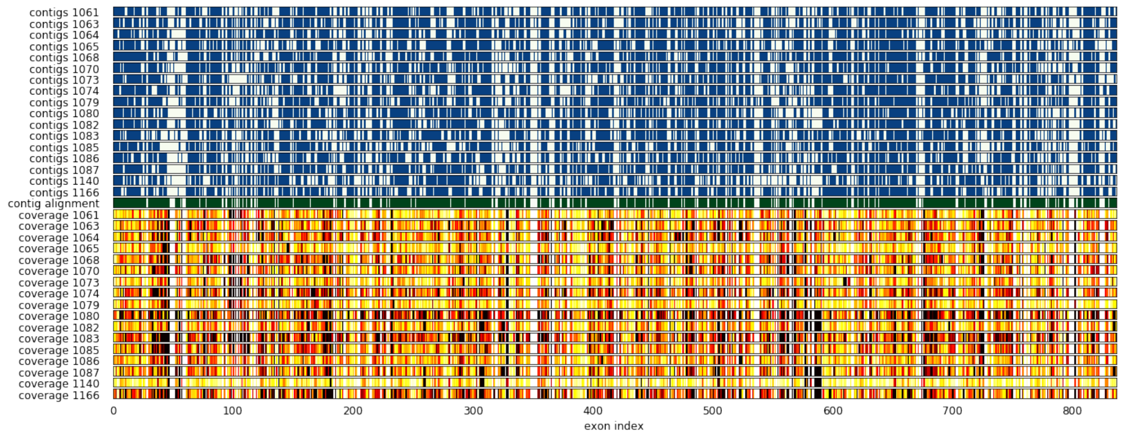
We finally run the command as follows:

```
secapr reference_assembly --reads ../../data/processed/cleaned_trimmed_reads --refe
```

1.3 Reviewing results of reference-based assembly

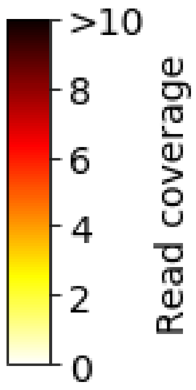
`secapr` provides several plotting options that help to review the results of the reference-based assembly. Below we plot the read-coverage for each sample below the presence/absence matrix of the contig data. Read-coverage is plotted according to a heatmap, with higher read-coverage resulting in 'hotter' colors. Below we use the flag `norm_value=10` in the plotting command in order to apply the heatmap scale for values between 0 and 10. All loci covered by more than 10 reads are colored with the same heatcolor as the value 10, which is black. As you see many loci were not very well covered in our example data (light colors, see legend below):

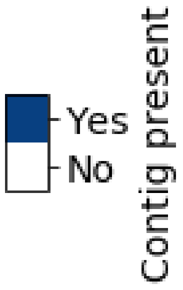
```
In [2]: import sys
        sys.path.append("../../src")
        import plot_contig_data_function as secapr_plot

        contig_input_file = '../../data/processed/target_contigs/match_table.txt'
        alignment_folder = '../../data/processed/alignments/contig_alignments'
        read_cov_file = '../../data/processed/remapped_reads/average_cov_per_locus.
        read_cov_folder = '/'.join(read_cov_file.split('/')[:-1])

        fig = secapr_plot.plot_contigs_alignments_read_cov(contig_input_file,alignm
        fig
        #fig.savefig(os.path.join(read_cov_folder,'contig_exon_coverage_matrix.png

Out[2]:
```

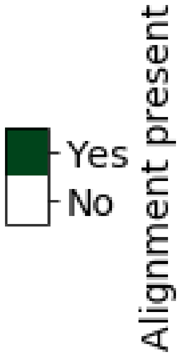


Here are the corresponding legends:

```
In [19]: import matplotlib.pyplot as plt
         %matplotlib inline
         import warnings
         warnings.filterwarnings('ignore')
         legend = secapr_plot.plot_heatmap_legend(0,10,width=.75,height=2, font_siz
         contig = secapr_plot.general_scale_bar(2,tick_labels=['No','Yes'],x0=.1,x1
         align = secapr_plot.general_scale_bar(2,tick_labels=['No','Yes'],x0=.1,x1=
         plt.show(contig)
         plt.show(align)
         plt.show(legend)
         #legend.savefig(os.path.join(read_cov_folder,'legend_exon_coverage.png'),
```



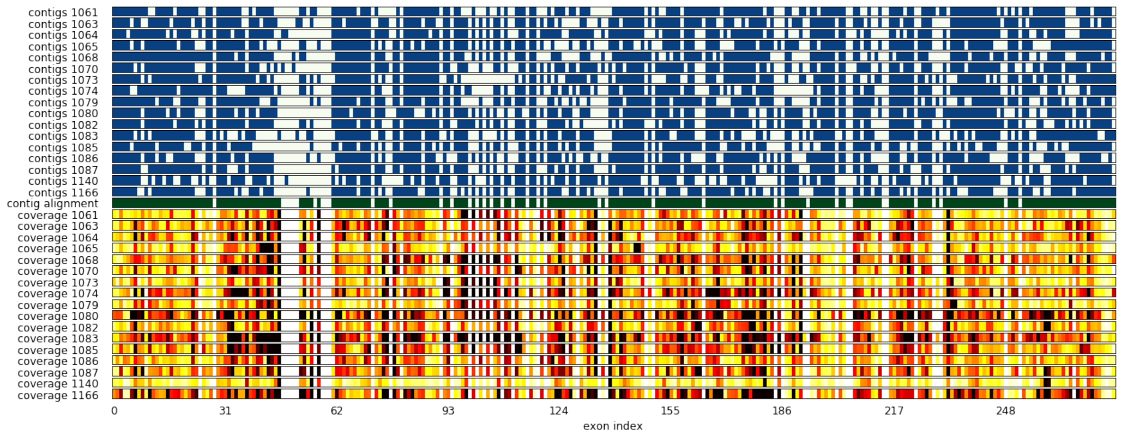


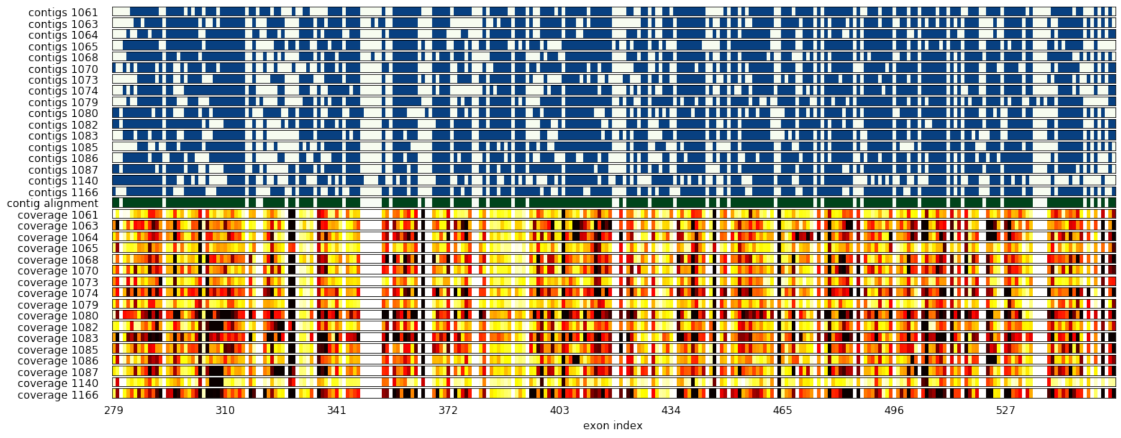


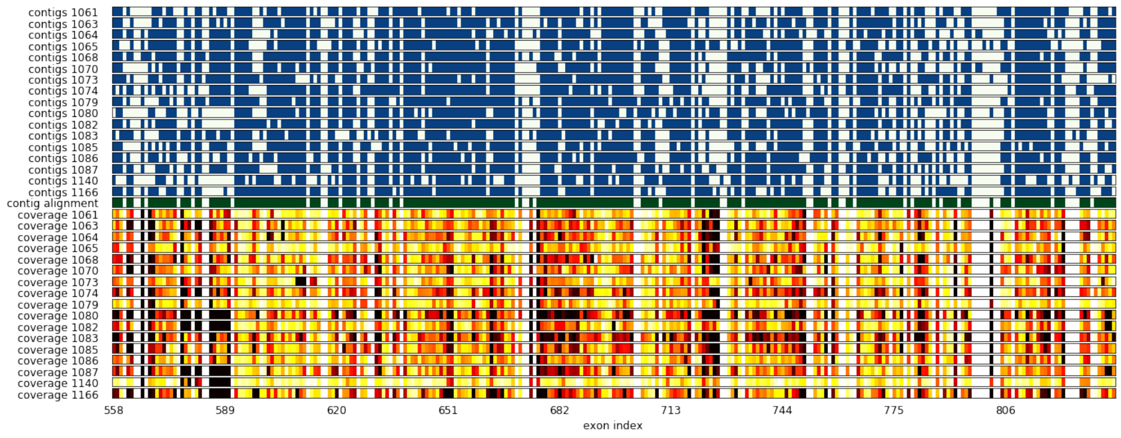
Once you are satisfied with the results of the reference-based assembly, you can move on to the allele phasing step. If there are many loci with very low read coverage (which will complicate allele phasing) it is recommendable to go through the process of locus selection first.

Tip: Split the plot into separate subplots for better readibility (see below).

```
In [23]: import sys
         sys.path.append("../../src")
         import plot_contig_data_function as secapr_plot

         contig_input_file = '../../data/processed/target_contigs/match_table.txt'
         alignment_folder = '../../data/processed/alignments/contig_alignments'
         read_cov_file = '../../data/processed/remapped_reads/average_cov_per_locus
         images = secapr_plot.plot_contigs_alignments_read_cov(contig_input_file,al
         print('Finished with splitting the plot.')

Finished with splitting the plot.
```







Previous page | Next page