HybSeq course

Practical processing of HybSeq target enrichment sequencing data on computing grids like MetaCentrum

Vojtěch Zeisek, Roswitha Elisabeth Schmickl and Tomáš Fér

Department of Botany, Faculty of Science, Charles University, Prague
Institute of Botany, Czech Academy of Sciences, Průhonice
https://trapa.cz/, zeisek@natur.cuni.cz

January 20 to 23, 2020

Outline I

Outline II

Resources before we start

- Course git and information: https://trapa.cz/en/hybseq-course-2020
- Most of the work is done in Linux/UNIX (macOS, ...) command line, so that good knowledge of work in command line is essential, good starting point can be my Linux and MetaCentrum course https://soubory.trapa.cz/linuxcourse/
- Many tasks are done in R, so that at least basic knowledge of R is needed, good starting point can be my R course https://soubory.trapa.cz/rcourse/
- As HybPiper is written in Python, so that at least minimal knowledge of this language is advantageous
- Processing HybSeq data is computationally demanding (it requires plenty of resources), during the course we use MetaCentrum, Czech National Grid Infrastructure (slide 13), but any computing cluster or powerful desktop (for patient users;-) can be used

HybSeq and its data

- HybSeq combines target enrichment and genome skimming (see lesson by RS) and especially in lager plant genomes it allows to select only ∼1000 single/low copy genes
- It requires sequencing probes, general or group specific (can be design using pipelines like Sondovač)
- From sequencing laboratory we get demultiplexed raw FASTQ files
- Steps leading to lists of gene trees require plenty of computing resources and disk space
  - Even simple operation can take significant time — think twice before every step
  - User can select how much resources provide for each step — depends on data size and available resources (more resources like CPU and memory will speed up processing)

Oxalis test data set

- Genus *Oxalis* has altogether ca. 500 species, over 200 in South Africa
- We selected 24 South African species (following Schmickl et al. 2016) as test data
- The probes used for sequencing were introduced in Schmickl et al. 2016, but were reduced (some probes with poor sequencing results were removed)
- For data structure see slide 28
- If you did not yet do so, download data (slide 26) — it can take long time...

Steps from sequencing files to species trees I

1. Trimming of raw sequencing FASTQ files (removal of adaptors, ...) e.g. by Trimmomatic
2. Deduplication of FASTQ reads e.g. by BBMap
   - Not strictly required, duplicates mainly provide wrong insight into real coverage of particular loci
3. Checking of FASTQ files in FastQC or similar tool and removal of low-quality files
4. Preparing probe reference FASTA file and list of samples for processing by HybPiper
5. Processing every sample with with HybPiper (or alternatively HybPhyloMaker or similar tool)
   1. Mapping of FASTQ reads with BWA to FASTA reference
   2. Distributing (sorting) of reads according to successful hits (using Samtools) into FASTA files for assembly
   3. Assembly of sorted reads with SPAdes
   4. Alignment of SPAdes contigs against the target sequence
      - Contigs are not expected to overlap much

Steps from sequencing files to species trees II

- Initial exonerate search is filtered for hits that are above a certain threshold
- Contigs that pass this filter are arranged in order along the alignment
- All contigs that pass the previous steps are concatenated into a "supercontig" and the exonerate search is repeated

5. Search for paralogs — if SPAdes assembler generates multiple contigs that contain coding sequences representing 75% of the length of the reference protein, HybPiper will print a warning for that gene
6. Recovering of the individual sequences
7. Statistics of the recovery
8. Cleanup of temporal files (especially SPAdes produces huge amount of data unneeded for further processing)

6. Statistics of sequence lengths in all samples and more information about recovered contigs
7. Creation of heatmaps (using R and packages gplots and heatmap.plus, or ggplot2 and reshape2)

Steps from sequencing files to species trees III

8. Retrieve of sequences of exons, introns and supercontigs for all samples
9. Alignment of all contigs (e.g. by MAFFT; or MUSCLE, Clustal, ... e.g. using R and packages ape and/or ips)
   - All alignments must be trimmed — columns/rows with too much missing data (e.g. beginning and end of the alignment) must be removed (e.g. using R and package ape)
   - It is also useful to create simple NJ tree graphical check of alignment (e.g. using R and package ape)
10. Sorting of alignments, statistics of their length and quality, discarding of poor (too short, too few individuals, too few variable positions, ...) alignments
11. Reconstruction of gene trees from all aligned contigs (e.g. using IQ-TREE, or ExaML, MrBayes, PhyML, RAxML, ...)
12. Post-processing of gene trees
    - Identification, inspection and possible removal of gene trees with significantly different topology (e.g. by R and packages ape and kdetrees, TreeShrink)

Steps from sequencing files to species trees IV

- Comparison of gene trees (e.g. heatmaps and PCoA by R and packages ade4, ape, distory, phytools)
- Comparison of (several) (species) trees (e.g.by R and packages ape or phytools)

13. Construction of species trees (e.g. by ASTRAL)
    - Comparison of species tree and gene trees (e.g. by phyparts and MJPythonNotebooks)
14. Phylogenetic networks (e.g. by PhyloNet)

Note...

- This general scheme can be significantly altered...
- There are plenty of technical as well as biological problems (HGT, ILS, ...) and new software keep being developed...
- Much more analysis possible...

List of software used during the course I

- ASTRAL (see lesson by TF) — species trees from gene trees
- BASH 4 or later and GNU core utils (“Linux command line”)
- BBMap — deduplication of FASTQ
- BLAST+ (used by HybPiper)
- BWA (used by HybPiper)
- Dendroscope — visualize outputs of PhyloNet
- Exonerate (used by HybPiper)
- GNU Parallel (used by HybPiper and in BASH scripts)
- HybPiper — recovering genes from targeted sequence capture data
- IQ-TREE — gene trees
- MAFFT — alignment

List of software used during the course II

- MJPythonNotebooks (used by phyparts)
- PhyloNet — phylogenetic networks
- phyparts — comparison of species tree vs. gene trees
- Python 2.7 or later and Biopython 1.59 or later (used by HybPiper)
- QuartetScores (see lesson by TF) — support scores for internodes
- R 3.5 or later and packages ade4, adegenet, ape, corrplot, distory, ggplot2, gplots, heatmap.plus, ips, kdetrees, pegas, phangorn, phytools and reshape2
  - Used by HybPiper, for alignment of contigs, post-processing of alignments, post-processing and comparison of gene trees, etc.
- Samtools (used by HybPiper)
- SPAdes (used by HybPiper)
- TreeShrink — detection of outlier long branches in collections of phylogenetic trees
- Trimmomatic — trimming of FASTQ

CESNET and MetaCentrum I

- CESNET is organization of Czech universities, Academy of Science and other organizations taking care about Czech backbone Internet, one of world leading institutions of this type
- CESNET provides various services
  - Massive computations — MetaCentrum — this we need to process our HybSeq data
  - Large data storage — this we need to store our HybSeq data
  - FileSender to be able to send up to 1.9 TB file
  - Cloud — computing (HPC) cloud similar to e.g. Amazon Elastic Compute Cloud (EC2) or Google Compute Engine
  - ownCloud to backup and/or sync data across devices (default capacity is 100 GB, user may ask for more) — similar to e.g. Dropbox
    - It is possible to connect by webDAV to ownCloud— many applications support it
    - It is possible to share calendars and/or address books via calDav and cardDav among devices and/or people
- Services accessible without registration
  - ownCloud https://owncloud.cesnet.cz/

CESNET and MetaCentrum II

- FileSender https://filesender.cesnet.cz/
- Go to web and log in with your institutional password

- Services requiring registration (and approval)
  - To use MetaCentrum fill registration form https://metavo.metacentrum.cz/en/application/form
  - To use data storage fill registration form https://einfra.cesnet.cz/perun-registrar-fed/?vo=storage
  - After registration for MetaCentrum, user can join MetaCloud via https://perun.metacentrum.cz/fed/registrar/?vo=meta&group=metacloud
  - Users not having access to EduID have to register first at HostelID http://hostel.eduid.cz/en/index.html
- Information about data storage https://du.cesnet.cz/en/start contains detailed usage instructions
- Information about MetaCentrum https://www.metacentrum.cz/en/

CESNET and MetaCentrum III

- Information about MetaCloud
  https://wiki.metacentrum.cz/wiki/Kategorie:Clouds
- Most of practical information for users are at wiki
  https://wiki.metacentrum.cz/w/index.php?&setlang=en

MetaCentrum vs. other clusters...

I show processing on MetaCentrum Czech National Grid Infrastructure, as it is readily available, well maintained and contains all needed applications, but it's possible to use any computing cluster in similar way.

MetaCentrum

- Find all needed information at https://wiki.metacentrum.cz/wiki/Main_Page
- Current state and usage as available at https://metavo.metacentrum.cz/en/
- Manage your user account at http://metavo.metacentrum.cz/en/myaccount/
- Personal view on actual resources and running tasks is at https://metavo.metacentrum.cz/pbsmon2/person
- List of available applications https://wiki.metacentrum.cz/wiki/Kategorie:Applications
- It has 9 frontends where users log and thousands of computers doing the calculations — they are not accessed directly to run task
- Most of computers are running Debian GNU/Linux

MetaCentrum usage

- User can transfer data on one of frontends or to data storage by e.g. `scp` or WinSCP from Windows or FileZilla from anywhere
- Same credentials are used for all frontends, computing nodes as well as data storage, for SSH login as well as file transmissions

```
# Login to selected server (tarkil is located in Prague)
ssh USER@tarkil.metacentrum.cz
# Continue as in any other command line...
qsub ... # Submit the job (see later)
```

- In home directory on the server prepare all needed data and non-interactive script (interactive are more complicated) which will do the calculations
- Tasks are not launched immediately, but using `qsub` the task is submitted into queue and system decides when it will be launched

File transfers to MetaCentrum

- Graphical applications: WinSCP, FileZilla or from most of file managers
- Protocol is SSH/SSH2/SFTP/SCP, port 22, server is selected frontend's address (e.g. `tarkil.metacentrum.cz`) — it is recommendable to use all the time same frontend
- All servers are accessible under domain `*.metacentrum.cz`: `skirit`, `perian`, `onyx`, `zuphux` (located in Brno), `alfrid`, `nympha`, `minos` (in Pilsen), `tarkil` (in Prague), and `tilia` (in Průhonice) — so that e.g. `tarkil.grid.cesnet.cz` is synonymous to `tarkil.metacentrum.cz`
- In the same way use SCP/SFTP/rsync/SSH to `ssh.du4.cesnet.cz` to access the storage, see help

Launching of tasks

- https://wiki.metacentrum.cz/wiki/How_to_compute/Requesting_resources
- Personal view https://metavo.metacentrum.cz/pbsmon2/person has nice overview of available resources and tasks and allows comfortable construction of submission command

```
# We will run up to 5 days (120 h), require one physical
# computer with 8 CPU threads, 24 GB of RAM, 10 GB of disk
# space and we get all information mails
qsub -l walltime=120:0:0 -l select=1:ncpus=8:mem=24gb:
  scratch_local=10gb -m abe metacentrum.sh
# Check how the task is running (above web) and
qstat -u $USER # Information about $USER's jobs
qstat 123456789 # The task ID is available from qstat
qstat -f 123456789 # Print a lot of details
qdel 123456789 # Terminate scheduled or running task
```

Key MetaCentrum commands

- MetaCentrum is "just" normal Linux server — work as usually
- Command `module` loads/unloads selected application
- Tasks (BASH scripts) are submitted for computing by `qsub` — the script must copy the data into `$SCRATCHDIR` and do all calculations there
  - It has plenty of options how to specify requirements (see help)
- Queued and running jobs can be seen by `qstat -u $USER` (`qstat` has much more options) and any job can be terminated by `qdel 123456789` (number from `qstat`)

```
module add <TAB><TAB> # Load some module
module rm XXX # Unload selected module
module list # List of currently loaded modules
qsub ... # Submit task for computing
qstat -u $USER # See $USER's running and queued jobs
qdel 123456789 # Termite task (number from qstat)
```

Scheduling details I

- Specify needed time
  - Always `hours:minutes:seconds`, so e.g. for 4 weeks use
    `-l walltime=672:0:0` ($28 \cdot 24$), for two days and 12 hours
    `-l walltime=60:0:0`
  - User may ask to prolong the walltime — it is needed to write in advance
- Ask for as much RAM as you need (e.g. `-l mem=8gb` to request 8 GB of memory)
  - If the task is going to require more, than allowed, system kills it…
  - If user doesn't use all required RAM, the system temporarily lowers priority for future tasks
  - It can be hard to estimate…
- Disk space is relatively free resource, user can ask more to have some reserve (e.g. `-l scratch_local=10gb` to request 10 GB)

Scheduling details II

- Specify how many physical computer(s) you are going to use (e.g. `-l select=1` for one machine) and number of CPU threads on each machine (e.g.
  `-l select=1:ncpus=8` for 1 machine with 8 cores or
  `-l select=2:ncpus=4` for 2 machines, each with 4 CPU threads)
  - It use to be necessary to specify correct number of threads for the application (e.g. `parallel -j 4`) — the application sees all CPUs on the machine, but can't use them
  - If the application consumes less than required, the system temporarily lowers priority for future tasks, if it try to use more, it will be very slowed down or killed by the server
- If requesting e-mails (e.g. `-m abe` to get mail about abort, beginning and exit of the task) and submitting plenty of tasks by some script, it can result in hundreds of mails — receiving mail servers don't like it...

Scheduling details III

- Every user has certain priority highered by acknowledgments in publications to MetaCentrum and lowered by intensive usage of the service (the usage is calculated from past month)
- After submission of the task, check in the queue in which state it is — sometimes it can't start because of impossible combination of resources or so
- User can check load of machines
- For more options read `https://wiki.metacentrum.cz/wiki/How_to_compute/Requesting_resources`
  - Request special CPU (AMD, graphical, ...), e.g. CPU with AVX2
    ```
    -l select=cpu_flag=avx2
    ```
  - Request particular location, ...
  - `https://metavo.metacentrum.cz/pbsmon2/qsub_pbspro` helps with preparation of `qsub` command

Get to task's working directory

- Go to https://metavo.metacentrum.cz/pbsmon2/person and click to list of your tasks and click to selected task
- Search for information **exec_host** (address of node doing the task) and **SCRATCHDIR** (temporal directory for all data and results)
- Sometimes one needs to monitor task progress or influence it
- It is not possible to directly modify running task, but at least check (and possibly modify) input data and see outputs

```
# From MetaCentrum frontend login to node running the task
ssh exec_host # No need to specify user name; e.g. mandos9
# Go to SCRATCH directory
cd SCRATCHDIR # e.g. /scratch/gunnera/job_90220.meta-pbs...
# There are working data of currently running task...
# Check whatever you need...
```

Running R tasks on MetaCentrum

- There are no R packages, user must create local package library and provide path — Be careful about paths!
- In the `metacentrum.sh` script load R `module add R-3.5.1-gcc` and start R script as usually `R CMD BATCH script.r`

1. Login to selected frontend via SSH
2. Go to working directory `cd workdir`
3. Create new directory for R packages `mkdir rpackages`
4. Start R `R` and install **all** R packages needed for the task — install them into directory `rpackages`: `install.packages(pkgs=..., lib="rpackages")`
5. In the R script load the packages from the `rpackages` directory `library(package=..., lib.loc="rpackages")`
6. Ensure all needed outputs are saved from the R script

Data download

- It's not possible to store such large data on MetaCentrum frontend, it must be download to the CESNET storage
- Servers powering CESNET storage have only very limited set of command line tools available
- We can login to any MetaCentrum frontend and then go to directory where the storage is accessible

```
# Login to any MetaCentrum frontend
ssh USER@tarkil.grid.cesnet.cz # Or any other fronted
# Go to your directory on the CESNET data storage
cd /storage/ostrava2-archive/tape_tape/backup/VO_storage/
  home/$USER/ # $USER's home on the storage
# Download the course data
wget ftp://botany.natur.cuni.cz/hybseq_course.zip
# Unpack it
unzip hybseq_course.zip
```

Scripts and other resources to process the data

- The archive `hybseq_course.zip` contains test data as well as needed scripts, R packages, HybSeq reference, ...
- The other resources (not the data in the `oxalis` directory) should be moved to frontend
- Inspect content of the `bin` and `hybseq` directories
- Scripts must be updated to point to correct location of the `oxalis`

Do not blindy copy-paste commands...

Commands show typical way how to proceed, but should not be using without understanding, and onther ways how to work are possible...

```
# Login to any MetaCentrum frontend
ssh USER@tarkil.grid.cesnet.cz # Or any other fronted
# Move scripts and other resources into home directory
mv /storage/ostrava2-archive/tape_tape/backup/VO_storage/
  home/$USER/{bin,hybseq} ~/ # Note $USER/{...,...} syntax
```

Oxalis test data directory structure I

```
├── bin # BASH script to be submitted by qsub
│   └── HybPiper # HybPiper Python and R scripts
├── hybseq # BASH scripts to process the data
│   ├── dedup # Output directory for deduplicated sequences
│   ├── qual_rep # Output directory for FastQC reports
│   ├── ref # References for HybPiper
│   ├── rpackages # R library with installed R packages
│   └── trimmed # Output directory for trimmed sequences
└── oxalis # Oxalis test data
    ├── 1_data # Sequencing libraries
    │   ├── lib_01 # Sequencing library 1
    │   │   ├── 0_data # Raw FASTQ sequences
    │   │   ├── 1_trimmed # Trimmed FASTQ sequences
    │   │   ├── 2_dedup # Deduplicated FASTQ sequences
    │   │   └── 3_qual_rep # FastQC quality reports
    │   └── lib_02 # Sequencing library 2
    ... ... ├... # Next slide...
```

Oxalis test data directory structure II

```
... ... ... # Previous slide...
▯   ▯   ├── 0_data # Raw FASTQ sequences
▯   ▯   ├── 1_trimmed # Trimmed FASTQ sequences
▯   ▯   ├── 2_dedup # Deduplicated FASTQ sequences
▯   ▯   └── 3_qual_rep # FastQC quality reports
▯   ├── 2_seqs # Outputs of HybPiper
▯   ▯   ├── o_amblyodonta_S524.dedup # Sample output dir
▯   ▯   ▯   ├── Assembly_10176 # Gene output directory
▯   ▯   ▯   ├── Assembly_10307 # Gene output directory
▯   ▯   ▯   ├── ... # Gene output directory
▯   ▯   ├── ... # More samples...
▯   ▯   └── o_zeekoevleyensis_S518.dedup # Sample output dir
▯   ▯   ▯   ├── Assembly_10176 # Gene output directory
▯   ▯   ▯   ├── Assembly_10307 # Gene output directory
▯   ▯   ▯   ├── ... # Gene output directory
▯   ├── 3_aligned # Aligned contigs
... ├... # Next slide...
```

Oxalis test data directory structure III

```
... ... # Previous slide...
▯   ├── exons # Aligned exons
▯   ├── introns # Aligned introns
▯   └── supercontigs # Aligned supercontigs
├── 4_gene_trees # Reconstructed gene trees
▯   ├── exons # Gene trees of exons
▯   ├── introns # Gene trees of introns
▯   └── supercontigs # Gene trees of supercontigs
└── 5_species_trees # Processing of the trees
    ├── distances_kdetrees_comparisons
    ├── phyparts # Comparison of species vs. gene trees
    ├── phylonet # Phylogenetic networks
    └── treeshrink # Identification of outlied branches
    ... # More downstream analysis...
```

- Of course, every user can figure different directory structure, but HybSeq produces a lot of data and plenty of software packages are used, so keep some logical structure...

Trimming and deduplication

- Raw demultiplexed FASTQ sequences must be trimmed (sequencing adaptors removed, ...) and should be deduplicated (removal of artificial duplicates to get correct statistics of coverage)
- There are plenty of software packages available, Trimmomatic use to be used for trimming and e.g. BBMap for deduplication
- Usually, libraries are processed as they are delivered from sequencing company
- Quality of all FASTQ files should be checked by e.g. FastQC
- It is practical to obtain simple statistics — number of sequences in original files, after trimming and after deduplication
- Low quality files should be discarded...
- Everything can be easily coded into simple BASH script processing all files (see following slides)

Scripts to trim and deduplicate FASTQ sequences

- Script `~/hybseq/hybseq_1_prep.sh`
  - See `./hybseq_run_1_prep.sh -h` for usage help
  - Script checks if all needed parameters and tools are available and in simple `for` loop trimms all sequences (one by one), deduplicates them, does quality checking, prints simple statistics, and prepares list of samples for HybPiper
- Script `~/bin/hybseq_run_1_prep.sh`
  - Submits via `qsub` script `~/bin/hybseq_run_1_prep.sh` for calculation
  - Variables `WORKDIR` and `DATADIR` must point to correct existing location
  - The script is started twice to process both sequencing libraries
  - If changing parameters for `hybseq_1_prep.sh`, parameters for `qsub` must be changed accordingly

Submission of hybseq_1_prep.sh

- To prepare data for HybPiper
- `/bin/hybseq_run_1_prep.sh` must be edited before submission via `qsub`
- After it runs for a while (everything had been copyied to the computing node), it is possible to change `DATADIR` and submit processing of the second library

```
# After edition of WORKDIR and DATADIR run
qsub -l walltime=12:0:0 -l select=1:ncpus=4:mem=16gb:
  scratch_local=100gb -m abe ~/bin/hybseq_run_1_prep.sh
# Or similar command
# Monitor running $USER's tasks with details
qstat -w -n -1 -u $USER # Last column contains machine name
# See your processes running the machine (from above list)
ssh exec_node "ps ux" # Replace exec_host by hostname!
```

Trimm and deduplicate all FASTQ files

Tasks to pre-process FASTQ data for HybPiper

1. Inspect `~/bin/hybseq_run_1_prep.sh` and `~/hybseq/hybseq_1_prep.sh` and be sure to understand what the scripts do, including syntax used.
2. Edit declaration of variables `WORKDIR` and `DATADIR` so that they point to correct locations.
3. Submit via `qsub` `~/bin/hybseq_run_1_prep.sh` to process both libraries and monitor the jobs during processing.
4. Inspect outputs of `~/bin/hybseq_run_1_prep.sh`, including statistics and FASTQ checks. What do they show?
5. Can you run the task on your computer (desktop or notebook) without `qsub`? If so, how?

Requirements to run HybPiper

- See https://github.com/mossmatters/HybPiper/ to see software requirements to run HybPiper
- It is possible to run HybPiper on your computer (with Linux or macOS), but it requires plenty of CPU and memory and creates huge output directories...
- To process multiple files (by `while` loop or by `~/bin/hybseq_run_2_hybpiper_1_submitter.sh`) there must be list of sample base names without suffixes like `*[._]R{1,2}.f*q*` (here created by `hybseq_1_prep.sh`)
- Reference bait FASTA file **must** have sequences named as `Species_name-gene_id` (see help) (note order and dash in between)
- HybPiper is processing individual files with given baits FASTA sequences — batch processing must be scripted

Before running HybPiper

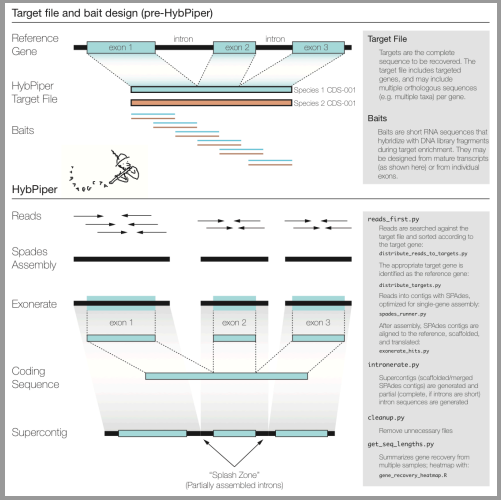
- Directory `~/hybseq/ref/` contains prepared reduced bait file `new_soa_probes_gen_comp.fasta` (it will be used to process test data) and unreduced `input_seq_without_cpdna_1086_loci_renamed.f...` (with renamed sequences from `input_seq_without_cpdna_1086_loci.fa`)
- To run HybPiper we need both files above and all required software, and BASH script to process all input files in batch — see `hybseq_run_2_hybpiper_1_submitter.sh` (submitting via `qsub` all input files), `hybseq_run_2_hybpiper_2_qsub.sh` (preparing individual job to run) and `hybseq_2_hybpiper.sh` (processing every input file — doing the job)

Prepare to run HybPiper

Tasks before running HybPiper

1. Check `samples_list.txt` in `2_dedup` output directories of both libraries after running `~/hybseq/hybseq_1_prep.sh`, and check how it was created.
2. In `input_seq_without_cpdna_1086_loci.fa` (output of Geneious assembler) consider `Contig_#` (`#` stands for number) as standing for `Oxalis_obtusa_#` (mutliple *Oxalis obtusa* individuals were used) and `Assembly_#` as gene ID and think how to transfor it to form required by HybPiper (also discard final `_#`). Use any good text editor (or `sed` and another command line tools) and regular expressions.
3. Do we have everyhting needed to run HybPiper?

Overview of HybPiper



Understanding how presented scripts run HybPiper I

- Script `~/bin/hybseq_run_2_hybpiper_1_submitter.sh` goes to directory set by `DATADIR` and for every sample (forward and reverse FASTQ files) listed in `samples_list.txt` (variable `SAMPLES`, created by `hybseq_2_hybpiper.sh`) submits via `qsub` individual task
- Script `~/bin/hybseq_run_2_hybpiper_2_qsub.sh` drives submission and manipulation of processing of each individual file — it checks if all required data are provided, copy input files to `SCRATCHDIR`, loads required software modules, runs processing itself by `hybseq_2_hybpiper.sh` and copy results back
- Script `~/hybseq/hybseq_2_hybpiper.sh` runs all HybPiper steps for individual samples (as submitted by previous scripts)

Understanding how presented scripts run HybPiper II

- Outputs of `~/hybseq/hybseq_2_hybpiper.sh` should be checked and moved into special directory (`oxalis/2_seqs`) where new list of samples must be created
- Script `~/bin/hybseq_run_3_hybpiper_postprocess.sh` can be then used to run `~/hybseq/hybseq_3_hybpiper_postprocess.sh` which uses HybPiper to retrieve sequences of exons, introns and supercontigs, and prints statistics and creates heatmaps
- Retrieved sequences can then be aligned

Submitting jobs to run HybPiper

- To retrieve probe sequences from every input FASTQ file
- `~/bin/hybseq_run_2_hybpiper_1_submitter.sh` must be edited before running it
- After submission of all input files is done, it is possible to change `DATADIR` and submit processing of the second library
- It will start job for every sample, so that output of `qstat` will notably prolong...

```
# After edition of HYBPIPDIR, WORKDIR, DATADIR, SAMPLES,
# BAITFILE and NCPU run simply
~/bin/hybseq_run_2_hybpiper_1_submitter.sh
# Monitor running $USER's tasks with details
qstat -w -n -1 -u $USER # Last column contains machine name
# See your processes running the machine (from above list)
ssh exec_node "ps ux" # Replace exec_host by hostname!
```

Running HybPiper

Run HybPiper

1. Inspect scripts `~/bin/hybseq_run_2_hybpiper_1_submitter.sh`, `~/bin/hybseq_run_2_hybpiper_2_qsub.sh` and `~/hybseq/hybseq_2_hybpiper.sh` and be sure to understand what they are doing, including syntax used.
2. In `~/bin/HybPiper/` check `./reads_first.py -h`.
3. Edit declarations of variables `HYBPIPDIR`, `WORKDIR`, `DATADIR`, `SAMPLES` and `BAITFILE` in `hybseq_run_2_hybpiper_1_submitter.sh` so that they point to correct locations.
4. Process both libraries and inspect outputs and log files.

Other tasks with HybPiper

HybPiper tasks for advanced users

1. Think how to process all samples on single computer. Use `while` BASH loop and feed it by `samples_data.txt`. How would such script look like? What had to be changed in `qsub`?
2. Check requirements of software used by HybPiper (BWA, SPAdes, ...) and think if changing of number of CPU threads and memory would significantly speed up processing of individual files.
3. See `man qsub` and think if there is another option how to pass options from `hybseq_run_2_hybpiper_1_submitter.sh` for individual `qsub` commands (apart of usage of exported variables as is used now).
4. Think about HybPiper parameters in `hybseq_2_hybpiper.sh`.

After running HybPiper for every sample...

- Previous step processed by HybPiper every single sample individually and independently
- HybPiper postprocessing will retrieve contigs for every exon, intron and supercontig containing all individuals where particular genetic region was found; and do some statistics
- Script `~/bin/hybseq_run_3_hybpiper_postprocess.sh` is submitted via `qsub` and it runs `~/hybseq/hybseq_3_hybpiper_postprocess.sh` which uses HybPiper to obtain table of contig lengths, some statistics, heatmaps, and retrieves from individual directores contigs to be aligned
- New list of samples for all samples (all libraries) must be prepared (next slide)

Sorting data after running HybPiper for all samples

- Outputs of HybPiper are in same directory as where input files were — outputs from all libraries must be moved into dedicated directory for post-processing and sequence retrieval
- Code below is exemplary and requires edits, as well as script `~/bin/hybseq_run_3_hybpiper_postprocess.sh`

```
# Move from both directories oxalis/1_data/lib_0[12]/2_dedup
# all outputs of HybPiper to oxalis/2_seqs/ for next steps
mv HybPiper.* hybseq_hybpiper.* *.dedup ../../../2_seqs/
# Create in directory oxalis/2_seqs new samples_list.txt
find . -maxdepth 1 -type d | sed 's/^\.\///' | sort | tail -n+2 >
  samples_list.txt
# Edit HYBPIPDIR, WORKDIR, BAITFILE and DATADIR in
# ~/bin/hybseq_run_3_hybpiper_postprocess.sh
# When ready, submit task to post-process HybPiper results
qsub -l walltime=12:0:0 -l select=1:ncpus=1:mem=2gb:scratch_local=100gb
   -m abe ~/bin/hybseq_run_3_hybpiper_postprocess.sh
```

Post-processing HybPiper outputs and retrieving contig sequences I

Tasks to post-process HybPiper outputs I

1. Inspect `~/hybseq/hybseq_3_hybpiper_postprocess.sh` and `~/bin/hybseq_run_3_hybpiper_postprocess.sh` and be sure to understand what the scripts do, including syntax used.
2. Edit declaration of variables `HYBPIPDIR`, `WORKDIR`, `BAITFILE` and `DATADIR` so that they point to correct locations.
3. Submit via `qsub` `~/bin/hybseq_run_3_hybpiper_postprocess.sh` to post-process all HybPiper outputs.
4. Inspect outputs of, including statistics, heatmaps and log. What do they show?

Post-processing HybPiper outputs and retrieving contig sequences II

Tasks to post-process HybPiper outputs II

1. Which part does take the longest time on this task? Does this task take plenty of resources (CPU, memory)?
2. If not satisfied with heatmaps, edit `R` scripts in `~/bin/HybPiper/` and run them manually (can be done in your notebook).
3. Can you run the task on your computer (desktop in office or notebook) without `qsub`? If so, how? Can you run the task directly on MetaCentrum fronted?

Alignment of all contigs I

- All sequences retrieved in the previous step with HybPiper must be aligned (by any aligner)
- Alignments must be post-processed
  - Rows (individuals) and/or colums (positions within alignment) with more than ∼10–20% of missing data must be removed (e.g. beginning and the end of the alignment)
  - Too short alignments or alignments with too few variable sites or too few individuals should be removed
  - Exact thresholds are to be discussed...
- Script `~/bin/hybseq_run_4_alignment_1_submitter.sh` goes to directory set by `DATADIR` and for every contig (all `*.fasta` or `*.FNA` files) in that directory (retrieved by `hybseq_run_3_hybpiper_postprocess.sh`) submits via `qsub` individual alignment task

Alignment of all contigs II

- Script `~/bin/hybseq_run_4_alignment_2_qsub.sh` drives submission and manipulation of processing of each individual file — it checks if all required data are provided, copy input files to `SCRATCHDIR`, loads required software modules, runs processing itself by `hybseq_4_alignment.r` `R` script and copy results back
- `R` script `~/hybseq/hybseq_4_alignment.r` alignes every input contig with MAFFT, trimms the alignment, creates NJ tree (NWK and PNG), creates image of alignment and reports alignment details
- Script `~/bin/hybseq_run_4_alignment_3_postprocess.sh` is short, can be runned on the fronted, it will sort outputs into subdirectories for exons, intron and supercontigs, create statistics of alignments and lists of NJ gene trees

Submitting alignment jobs

- To align all retrieved contigs (sequences)
- `~/bin/hybseq_run_4_alignment_1_submitter.sh` must be edited before running it
- It will start job for every sample, so that output of `qstat` will be very long (3 jobs — respective exon, intron and supercontig — for every of ∼1000 probes)...

```
# After edition of WORKDIR and DATADIR run simply
~/bin/hybseq_run_4_alignment_1_submitter.sh
# Monitor running $USER's tasks with details
qstat -w -n -1 -u $USER # Last column contains machine name
# See your processes running the machine (from above list)
ssh exec_node "ps ux" # Replace exec_host by hostname!
# Something went wrong? Cancel or running or queued tasks by
qdel $(qstat -u $USER | grep -o "^[0-9]\+" | tr "\n" " ")
```

Running alignments

Run alignments

1. Inspect scripts `~/bin/hybseq_run_4_alignment_1_submitter.sh`, `~/bin/hybseq_run_4_alignment_2_qsub.sh` and `~/hybseq/hybseq_4_alignment.r` and be sure to understand what they are doing, including syntax used.
2. Edit declarations of variables `WORKDIR` and `DATADIR` in `hybseq_run_4_alignment_1_submitter.sh` so that they point to correct locations.
3. Process all `*.FNA` and `*.fasta` files.
4. Monitor progress of the jobs
5. Inspect outputs (including images) and log files.

Other tasks with alignments

Alignment tasks for advanced users

1. Think how to process all samples on single computer. Use `find` to list all `*.FNA` and `*.fasta` files and pass them to GNU Parallel. How would such script look like? What had to be changed in `qsub`?
2. Check requirements of MAFFT and/or another aligners and think if changing of number of CPU threads and memory would significantly speed up processing of individual files.
3. In `~/hybseq/hybseq_4_alignment.r` replace usage of MAFFT by another aligner like e.g. MUSCLE or Clustal.
4. Think about parameters for functions `deleteGaps()`, `del.rowgapsonly()` and `del.colgapsonly()`. How do they influence output (`*.aln.fasta` files)?

After all alignment jobs are done

- Results (alignments named `*.aln.fasta` and other files) are in newly created `aligned` directory created by `hybseq_run_4_alignment_1_submitter.sh` in `DATADIR`
- Other outputs are images with alignment checks (`*.aln.check.png` and `*.aln.png`), NJ trees (`*.nwk` and `*.tree.png`) and logs (`*.log` from `R` and `HybSeq.alignment.*.[eo]*` from `qsub`)
- Outputs should be sorted by `hybseq_run_4_alignment_3_postprocess.sh` — it requires only path to the `aligned` directory

Sorting data after running alignments for all samples

- Outputs of alignments are in directory `aligned` which was created in the input directory, files with alignments itself are named `*.aln.fasta`
- It should be sorted by `hybseq_run_4_alignment_3_postprocess.sh`
- All outputs should be then moved to dedicated directory (cf. slide 28)
- Code below is exemplary and requires edits

```
# Post-process (sort into subdirectories and get simple
# statistics) all alignments - provide path to directory
# with aligned files
~/bin/hybseq_run_4_alignment_3_postprocess.sh oxalis/2_seqs/aligned/
```

Post-processing of alignments

Tasks to post-process alignments

1. Inspect `~/bin/hybseq_run_4_alignment_3_postprocess.sh` and be sure to understand what the script does, including syntax used.
2. Run `~/bin/hybseq_run_4_alignment_3_postprocess.sh` with correct path to post-process all aligned outputs.
3. Inspect outputs of, including statistics, images and logs. Open in spreadsheet (e.g. LibreOffice Calc) the `*.tsv` tables. What do they show?

Gene trees from all alignments I

- Gene trees must be computed from all aligned sequences
- Gene trees must be post-processed
  - Trees should be sorted into subdirectories for exons, introns and supercontigs and lists of gene trees created
  - Trees with significantly different topology must be identified and inspected (and possibly removed) — this will be later done in `R`
  - Long branches within trees must be identified and respective trees inspected — artificial long branches can be discarded from given trees (e.g. by TreeShrink)
- Script `~/bin/hybseq_run_5_gene_trees_1_submitter.sh` goes to directory set by `DATADIR` and for every aligned contig (all `*.aln.fasta` files) in that directory (created in previous step) submits via `qsub` individual task to reconstruct gene tree

Gene trees from all alignments II

- Script `~/bin/hybseq_run_5_gene_trees_2_qsub.sh` drives submission and manipulation of processing of each individual file — it checks if all required data are provided, copy input files to `SCRATCHDIR`, loads required software modules, runs processing itself by `hybseq_5_gene_trees.sh` and copy results back
- Script `~/hybseq/hybseq_5_gene_trees.sh` computes gene tree for every input file with IQ-TREE
- Script `~/bin/hybseq_run_5_gene_trees_3_postprocess.sh` is short, can be runned on the fronted, it will sort outputs into subdirectories for exons, intron and supercontigs and create lists of maximum likelihood and consensus (after bootstrapping) gene trees

Submitting gene trees jobs

- To reconstruct gene trees from every aligned and trimmed sequence
- `~/bin/hybseq_run_5_gene_trees_1_submitter.sh` must be edited before running it
- It will start job for every sample, so that output of `qstat` will be very long (3 jobs — respective exon, intron and supercontig — for every of ~1000 probes)...

```
# After edition of WORKDIR and DATADIR run simply
~/bin/hybseq_run_5_gene_trees_1_submitter.sh
# Monitor running $USER's tasks with details
qstat -w -n -1 -u $USER # Last column contains machine name
# See your processes running the machine (from above list)
ssh exec_node "ps ux" # Replace exec_host by hostname!
```

Running gene trees

Run gene trees

1. Inspect scripts `~/bin/hybseq_run_5_gene_trees_1_submitter.sh`, `~/bin/hybseq_run_5_gene_trees_2_qsub.sh` and `~/hybseq/hybseq_5_gene_trees.sh` and be sure to understand what they are doing, including syntax used.
2. Edit declarations of variables `WORKDIR` and `DATADIR` in `hybseq_run_5_gene_trees_1_submitter.sh` so that they point to correct locations.
3. Run it to process all `*.aln.fasta` files.
4. Monitor progress of the jobs
5. Inspect outputs and log files.

Other tasks with gene trees

Gene trees tasks for advanced users

1. See IQ-TREE help, check its parameters in `~/hybseq/hybseq_5_gene_trees.sh` and think about possible changes to fit better your needs.
2. Think how to process all samples on single computer. Use `find` to list all `*.aln.fasta` files and pass them to GNU Parallel. How would such script look like? What had to be changed in `qsub`?
3. Check requirements of IQ-TREE and/or another tree builder and think if changing of number of CPU threads and memory would significantly speed up processing of individual files.
4. Replace usage of IQ-TREE in `~/hybseq/hybseq_5_gene_trees.sh` by another tree builder like e.g. ExaML, MrBayes, PhyML or RAxML.

After all gene trees jobs are done

- Results are in newly created `trees` directory created by `hybseq_run_5_gene_trees_1_submitter.sh` in `DATADIR`
  - Records of what IQ-TREE did (`*.log`) and `HybSeq*` from `qsub`
  - Results are in `*.iqtree` (IQ-TREE report), maximum likelihood tree is in `*.treefile` and likelihood distances in `*.mldist`
  - Ultrafast bootstrap approximation results contain split support values in `*.splits.nex`, consensus tree in `*.contree`, bootstrap trees `*.ufboot` and likelihood mapping plots in `*.svg` and `*.eps`
- Outputs should be sorted by `hybseq_run_5_gene_trees_3_postprocess.sh` — it requires only path to the `trees` directory

Sorting data after running gene trees for all samples

- Outputs of gene trees are in directory `trees` which was created in the input directory, files with gene tree itself are named `*.contree`
- It should be sorted by `hybseq_run_5_gene_trees_3_postprocess.sh`
- All outputs should be then moved to dedicated directory (cf. slide 28)
- Code below is exemplary and requires edits

```
# Post-process (sort into subdirectories and get lists of
# gene trees) all gene trees - provide path to directory
# with gene trees files
~/bin/hybseq_run_5_gene_trees_3_postprocess.sh oxalis/3_aligned/trees/
```

Post-processing of gene trees

Tasks to post-process gene trees

1. Inspect `~/bin/hybseq_run_5_gene_trees_3_postprocess.sh` and be sure to understand what the script does, including syntax used.
2. Run `~/bin/hybseq_run_5_gene_trees_3_postprocess.sh` with correct path to post-process all gene trees outputs.
3. Inspect outputs of, including logs. What do they show?

Seeing trees in forest

- Comparison of gene trees start with identifying trees with significantly different topology
- There are several distance matrices allowing compare topological differences among trees (and subsequently plot heatmap, PCoA, etc.)

How to recognize artifact and real biologial feature?

- Without good reference genome it is hard to tell if long branches, weird topologies, etc. are some artifacts or biological reality...
- Problems commonly start with low-quality DNA in lab and subsequent high number of missing data
- Statistically, most of "weird" gene trees topologies are rather from technical issues, so that most of people filter them out...
- Results can vary according to strictness with trimming raw FASTQ, sensitivity of various HybPiper settings, settings of aligner and tree builder...

Distances comparing trees I

Single number to compare each pair of complex topologies?

- To compare topology of trees, we need some apropriate distance matrix
- There is no general agreement which is the best, all have issues...
- If the distance matrix is not Euclidean, we run into another issues...

- The tasks will be done in R
- Download e.g. `trees_ml_exons.nwk` (or another final list of gene trees) and work in `R` in your notebook
- Robinsons-Foulds distance in `phytools::multiRF`
  - The index adds 1 for each difference between pair of trees
  - Well defined only for fully bifurcating trees — if not fulfilled, some results might be misleading
  - Allow comparison of trees created by different methods

Distances comparing trees II

- If the difference is very close to root, RF value can be large, even there are not much differences in the tree at all — `dist.multiPhylo` from package distory can be an alternative, although interpretation of that geodesic distance is sometimes not so straightforward as simple logic of RF

- Methods implemented in `ape::dist.topo` allow comparison of trees with polytomies (`method="PH85"`) or use of squared lengths of internal branches (`method="score"`)
- Final matrices are commonly not Euclidean — may be problematic for usage in methods like PCoA
  - Test it with `ade4::is.euclid` , can be scaled (forced to became Euclidean) by functions like `quasieuclid` or `cailliez` in `ade4` — carefully, it can damage meaning of the data
  - We get matrix of pairwise differences among trees (from multiple genes), we need display and analyze it

Preparing lists of trees for import into R

- If the trees should be rooted, only trees containing the outgroup should be kept
- `trees_ml_exons.nwk` is shown as an example, but other trees can be used as well
- `grep` will easily keep only trees having outgroup `o_purpurascens_S482`

```
# Extract only trees having particular taxon
grep o_purpurascens_S482 trees_ml_exons.nwk > trees_ml_exons.out.nwk
# See how many trees were lost
wc -l trees_ml_exons.nwk trees_ml_exons.out.nwk
```

Loading trees into R

```
# Load libraries
library(ape)
library(ade4)
library(distory)
library(gplots)
library(ggplot2)
library(kdetrees)
library(phangorn)
# Set working directory
setwd("/home/vojta/dokumenty/vyuka/hybseq/")
# Load the list of trees
trees <- read.tree(file="trees_ml_exons.out.nwk")
trees # See it
# Root all trees
trees <- root.multiPhylo(phy=trees, outgroup="o_purpurascens_S482",
  resolve.root=TRUE)
print(trees, details=TRUE)
```

Heatmap of topological distances

- There are several heatmap functions, try also at least `heatmap` and `heatmap.plus::heatmap.plus`
- Edit settings to fit your needs and preferences

```
# Compute distance of topological similarities
trees.d <- dist.topo(x=trees, method="score")
# Plot the heatmap (package gplots)
png(filename="trees_dist.png", width=10000, height=10000)
  heatmap.2(x=as.matrix(trees.d), Rowv=FALSE, Colv="Rowv",
    dendrogram="none", symm=TRUE, scale="none", na.rm=TRUE,
    revC=FALSE, col=rainbow(15), cellnote=as.matrix(trees.d),
    notecex=1, notecol="white", trace="none",
    labRow=rownames(as.matrix(trees.d)), labCol=colnames
    (as.matrix(trees.d)), key=FALSE, main="Correlation
    matrix of topographical distances")
  dev.off() # Saves the image
```

PCoA of topological distances

- Requires Euclidean distance matrix (`is.euclid()`)
- Non-Euclidean matrices can be forced to became Euclidean by e.g. `quasieuclid()` or `cailliez()`
- There are plenty of options how to display it

```
# Test if the distance matrix is Euclidean
is.euclid(distmat=as.dist(trees.d), plot=TRUE, tol=1e-05)
# PCoA
trees.pcoa <- dudi.pco(d=trees.d, scannf=FALSE, nf=5)
trees.pcoa
# Plot PCoA
s.label(dfxy=trees.pcoa$li)
s.kde2d(dfxy=trees.pcoa$li, cpoint=0, add.plot=TRUE)
add.scatter.eig(trees.pcoa[["eig"]], 3,1,2, posi="bottomleft")
title("PCoA of matrix of pairwise trees distances")
```

Kdetrees

- Finds discordant phylogenetic trees
- Produces relative scores — high are relatively similar to each other, low dissimilar (discordant with the others)
- Produces scores, list of passing/discarded trees and graphical outputs
- In `kdetrees()`, value of `k` is responsible for threshold for removal of outliers — play with it

```
# Run kdetrees to detect outliers - play with k
?kdetrees # See options for kdetrees
trees.kde <- kdetrees(trees=trees, k=0.36, distance="dissimilarity",
  topo.only=FALSE, greedy=TRUE)
 # See text results with list of outlying trees
trees.kde
```

Outputs of kdetrees

```
# See graphical results
plot(x=trees.kde)
hist(x=trees.kde)
# See removed trees
plot.multiPhylo(trees.kde[["outliers"]])
# Save removed trees
write.tree(phy=trees.kde[["outliers"]], file="trees_outliers.nwk")
# Save kdetrees report
write.table(x=as.data.frame(x=trees.kde), file="trees_scores.tsv",
  quote=FALSE, sep="\t")
# Extract passing trees
trees.good <- trees[names(trees) %in% names(trees.kde[["outliers"]])
  == FALSE]
trees.good
# Save passing trees
write.tree(phy=trees.good, file="trees_good.nwk")
```

TreeShrink

- Algorithm for detecting abnormally long branches in one or more phylogenetic trees Mai et Mirarab 2018
- Requires R to be installed
- See more usage options

```
# Go to ~/bin directory
cd ~/bin/ || { mkdir ~/bin && cd ~/bin/; }
# Download TreeShrink
git clone https://github.com/uym2/TreeShrink.git
cd TreeShrink/
# Install it
python3 setup.py install --user # Or if using conda
conda install -c smirarab treeshrink
# Go to directory with trees_good.nwk and run TreeShrink
python3 ~/bin/TreeShrink/run_treeshrink.py -h # See help
python3 ~/bin/TreeShrink/run_treeshrink.py -r ~/bin/TreeShrink/
  -t trees_good.nwk
```

Outputs of TreeShrink

- Output (2 files) is saved into directory (in our case) `trees_good_treeshrink`
- File `*.nwk` contains new list of phylogenetic trees in `NEWICK` which can be then used as an input for any species tree reconstruction software
- File `*_RS_*.txt` is bit hard to read, it has one line for every tree in the input list and every line contains list of removed tips
  - If there is an empty line, no tip was removed from that particular tree
  - Trees are not named, only in same order as in the original input file

```
# Find out how many times particular sample was removed from the list of
# the trees
grep -o "\<[[:graph:]]\+\>" trees_good_RS_0.05.txt | sort | uniq -c |
  sort -r
```

Parsimony super tree

- Parsimony has plenty of implementation, example below is from `R` package `phangorn`

```
# Compute parsimony super tree
?superTree # See help first...
tree.sp <- superTree(tree=trees.good, method="NNI", rooted=TRUE,
  trace=2, start=NULL, multicore=TRUE)
# Rooting the species tree
tree.sp <- root(phy=tree.sp, outgroup="o_purpurascens_S482",
  resolve.root=TRUE)
tree.sp # See details
# Save parsimony super tree
write.tree(phy=tree.sp, file="parsimony_sp_tree.nwk")
# Plot parsimony super tree
plot.phylo(x=tree.sp, type="phylogram", edge.width=2,
  label.offset=0.01, cex=1.2)
add.scale.bar()
# Tune display of the tree...
```

Other options for species tree estimation in R

- Similar approach as `superTree` is implemented in `phytools::mrp.supertree`
- Distance-based tree reconstruction is in `ape::speciesTree`
- Coalescence model handling multiple individuals per species is in `phangorn::coalSpeciesTree`

```
# See help for mrp.supertree and coalSpeciesTree
?phytools::mrp.supertree
?phangorn::coalSpeciesTree
# All trees must be ultrametric - chronos scale them
trees.ultra <- lapply(X=trees, FUN=chronos, model="correlated")
class(trees.ultra) <- "multiPhylo"
# Calculate the species tree
tree.sp.mean <- speciesTree(x=trees.ultra, FUN=mean)
tree.sp2 <- mrp.supertree(tree=trees, method="optim.parsimony",
  rooted=TRUE)
```

Consensus network

- Available in `R` package `phangorn`
- Requires same set of tips in all trees

```r
# See help
?consensusNet
# Compute consensus network
tree.net <- consensusNet(obj=trees, prob=0.25)
# Plot 2D or 3D
plot(x=oxalis.tree.net, planar=FALSE, type="2D", use.edge.length=TRUE,
  show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
  show.nodes=TRUE, edge.color="black", tip.color="blue") # 2D
plot(x=oxalis.tree.net, planar=FALSE, type="3D", use.edge.length=TRUE,
  show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
  show.nodes=TRUE, edge.color="black", tip.color="blue") # 3D
```

Phylonet

- Requires as input NEXUS file with settings describing PhyloNet commands (see example below)

```
#NEXUS
BEGIN TREES;
... list of trees from trees_good.nwk newick file ...
# Ever tree starts with:
Tree TreeID = (tree in NWK)
... # All other trees ...
END;
BEGIN PHYLONET;
InferNetwork_MP (all) 1 -b 50 -x 5 -pl 2 -di;
END;

# Download binary JAR file (ready to run)
wget https://bioinfocs.rice.edu/sites/g/files/bxs266/f/kcfinder/files/
  PhyloNet_3.8.0.jar
java -Xmx8g -jar PhyloNet_3.8.0.jar file.nex | tee file.log
```

Running phylonet

- Prepare the list of trees for the NEXUS file e.g. in spreadsheet
- The `PHYLONET` section of the input NEXUS contains settings according to list of commands
- TreeID can be completely random, or simple consecutive sequence like GT0001–GT####
- PhyloNet can be computationally very demanding, calculating more than 1–3 reticulations can be unrealistic in terms of time needed...
- It does not save output file, the network in special NWK format for Dendroscope is on the end — copy it from terminal (after `Visualize in Dendroscope :`) or log file and save as tiny TXT, which can be opened in Dendroscope

Installing phyparts and other tools

- Requires `maven` and several Python packages, installation can be complicated...

```
cd ~/bin/
# Install Phyparts
git clone https://bitbucket.org/blackrim/phyparts.git
cd phyparts/
# Install dependencies
./mvn_cmdline.sh
# Install PhyParts_PieCharts
git clone https://github.com/mossmatters/MJPythonNotebooks.git
# Split list of trees into individual files
mkdir trees_good
split -a 4 -d -l 1 trees_good.nwk trees_good/trees_good_
ls trees_good/
```

Producing phyparts and phypartspiecharts.py outputs

```
# Remove IQTREE ultrafast bootstrap values from gene trees
sed -i 's/\/[0-9]\{1,3\}//g' trees_good/trees_*
# Analysis with phyparts
java -jar ~/bin/phyparts/target/phyparts-0.0.1-SNAPSHOT-
  jar-with-dependencies.jar --help
java -jar ~/bin/phyparts/target/phyparts-0.0.1-SNAPSHOT-
  jar-with-dependencies.jar -a 1 -d trees_good -m
  parsimony_sp_tree.nwk -o trees_good_res -s 0.5 -v
# Copy phypartspiecharts.py to directory with trees
cp ~/bin/phyparts/MJPythonNotebooks/phypartspiecharts.py .
# See help for phypartspiecharts.py
python phypartspiecharts.py --help
# Pie chart: concordance (blue) top conflict (green),
# other conflict (red), no signal (gray)
# Run phypartspiecharts.py to get the graphical output
python phypartspiecharts.py --svg_name trees_good_res.svg
  parsimony_sp_tree.nwk trees_good_res 144
```

Comparing two trees — cophyloplots

- Slightly different implementation in `R` packages `ape` (`cophyloplot`) and `phytools` (`cophylo`)
- See help pages and play with graphical parameters

```
# We need 2 column matrix with tip labels
tips.labels <- matrix(data=c(sort(tree.sp[["tip.label"]]), sort
  (tree.sp2[["tip.label"]])), nrow=length(tree.sp[["tip.label"]]), ncol=2)
# Draw the tree, play with graphical parameters
# Click to nodes to rotate them to get better display
cophyloplot(x=tree.sp, y=tree.sp2, assoc=tips.labels, use.edge.length=
  FALSE, space=60, length.line=1, gap=2, type="phylogram", rotate=TRUE,
  col="red", lwd=1.5, lty=2)
# Slihtly better display in phytools::cophylo
trees.cophylo <- cophylo(tr1=tree.sp, tr2=tree.sp2, assoc=tips.labels,
  rotate=TRUE)
plot.cophylo(x=trees.cophylo, lwd=2, link.type="curved")
```

Density tree

- The trees should be (otherwise plotting works, but may be more ugly) rooted, ultrametric and binary bifurcating
- implementations are in `phangorn` (`densiTree`) and `phytools` (`densityTree`)

```
is.rooted.multiPhylo(trees.ultra) # rooted
is.ultrametric.multiPhylo(trees.ultra) # ultrametric
is.binary.multiPhylo(trees.ultra) # binary bifurcating
# See help page
?phangorn::densiTree
# Plotting density trees
densiTree(x=trees.ultra[1:10], direction="downwards", scaleX=TRUE
  col=rainbow(3), width=5, cex=1.5)
densiTree(x=trees.ultra, direction="upwards", scaleX=TRUE, width=5)
densiTree(x=trees.ultra, scaleX=TRUE, width=5, cex=1.5)
```

Different display for multiple trees

- `phytools::densiTree` requires same number of tips in all trees
- Note various ways how to select trees to display
- Nodes of the trees are not rotated (the display might be suboptimal)

```
# See help page
?phytools::densityTree
# Plotting density trees
densityTree(trees=c(tree.sp, tree.sp2), fix.depth=TRUE, use.gradient=TRUE,
  alpha=0.5, lwd=4)
densityTree(trees=trees.ultra, fix.depth=TRUE, use.gradient=TRUE,
  alpha=0.5, lwd=4)
densityTree(trees=trees.ultra[1:3], fix.depth=TRUE, use.gradient=TRUE,
  alpha=0.5, lwd=4)
densityTree(trees=trees.ultra[c(2,4,6,7)], fix.depth=TRUE,
  use.gradient=TRUE, alpha=0.5, lwd=4)
```

This is just beginning of long journey...

- I presented common basic tools how to process HybSeq data and deal with various problems, but new tools keep emerging...
- Stay updated and keep trying new tools
- The scripts presented are not the only rigid way how to proceed, rather very general guideline, which should be subject of heavy modificcations according to your needs...

The end

Our course is over...

...I hope it was helpful for You...

...any feedback is welcomed...

...happy playing with the data...

...any final questions?