Metagenomics ECCB 2018 Tutorial

Part II: large scale sequence analysis

Clovis Galiez, Milot Mirdita, Martin Steinegger, Johannes Söding

Sept. 2018

1 Introduction

MMseqs2 (Many-against-Many sequence searching (Steinegger and Söding, 2017)) is a software suite to search and cluster huge protein sequence sets. MMseqs2 is open source GPLv3-licensed software implemented in C++ for Linux, macOS, and Windows. The software is designed to run on multiple cores and servers and exhibits very good scalability. MMseqs2 can run 10 000 times faster than BLAST. At 50 times its speed it achieves almost the same sensitivity. It can perform profile searches with the same sensitivity as PSI-BLAST at over 400 times its speed.

Figure 1: Your new friends! **MMseqs2** ultra fast and sensitive protein search, **Linclust** clustering huge protein sequence sets in linear time and **Plass** Protein-level assembly to increases protein sequence recovery from complex metagenomes

Here you will learn the basic usage of MMseqs2 as well as expert tricks to take advantage of the ability of chaining different MMseqs2 modules to produce custom workflows.

We will show on a human gut metagenomic dataset (SRA run `ERR1384114`) what the advantages of using Linclust (linear time clustering algorithm, Steinegger and Söding (2018)), Plass (Protein Level Assembly, Steinegger *et al.* (2018)) and MMseqs2 are over the more conventional pipeline with MegaHit(Li *et al.*, 2015), Prodigal(Hyatt *et al.*, 2010) and HMMER(Eddy, 2009).

2 Discover your environment

2.1 Command line interface and software

You will work on your own remote virtual machine through a Bash terminal in your browser. You will use the same environment as in the morning session. Please navigate to the **`$HOME/mmseqs2tutorial`** directory. All software and data has already been set up there. The commands to reproduce the installation are available in the file `scripts/setup.sh` on GitHub of the tutorial `https://github.com/soedinglab/metaG-ECCB18-partII`.

The software is installed in the `software` subdirectory and integrated in your `$PATH` environment variable.

You can take a look at the following commands:

```
plass
mmseqs
megahit -h
prodigal
```

The generic syntax for `mmseqs` and `plass` calls is always the following:

```
mmseqs <command> <db1> [<db2> ...] --flag --parameter 42
```

The help text of `mmseqs` shows, by default, only the most important parameters and modules. To see a full list of parameters and modules use the `-h` flag. You can auto complete commands and parameters of `mmseqs` by using the tab key.

The FASTQ read file you are going to work on is in the `data` directory.

3 Analysis of a human gut metagenomics dataset

3.1 Getting a protein catalogue of a metagenome

We use Plass to assemble a catalogue of protein sequences directly from the reads, without the nucleic assembly step. It recovers 2 to 10 times more protein sequences from complex metagenomes than other state-of-the-art methods and can assemble huge datasets.

The standard genomic assemblies prevent many reads to assemble due to **low coverages** and **micro-diversity**. To run this protein-level assembly, use the command

```
plass assemble data/ERR1384114.fastq plassProteins.faa tmpDir
```

or type `plass assemble -h` to see all available options.

As a matter of comparison, run the usual pipeline using MegaHit for genomic assembly:

```
megahit -r ERR1384114.fastq -o megahitAssemblyOutput
```

Then extract the proteins using Prodigal:

```
prodigal -i megahitAssemblyOutput/final.contigs.fa\
    -a prodigal.faa -p meta
```

Take a look at the FASTA files produced by Plass and Prodigal. To check the number of detected proteins, you can count the number of FASTA headers (lines beginning with the > character):

```
grep -c "^>" file.faa
```

Redundancy reduction Since Plass assembles with replacement of reads, the catalogue will contain some redundancy. You can reduce this catalogue by clustering it, for instance, to 90% of sequence identity, and asking for the representative sequence that cover at least 95% of the members. For this, you can either use the `easy-cluster` (sensitive clustering) or `easy-linclust` (linear time fast clustering) modules of MMseqs2:

```
mmseqs easy-cluster plassProteins.faa clusteredProts.faa \
    tmpDir --min-seq-id 0.9 -c 0.95 --cov-mode 1
```

Both the default MMseqs2 clustering and Linclust link two sequences by an edge based on three local alignment criteria:

- a maximum E-value threshold (option `-e`, default 10^{-3}) computed according to the gap-corrected Karlin-Altschul statistics;
- a minimum coverage (option `-c`, which is defined by the number of aligned residue pairs divided by either the maximum of the length of query/centre and target/non-centre sequences alnRes/max(qLen,tLen) (default mode, `--cov-mode 0`), by the length of the target/non-centre sequence alnRes/tLen (`--cov-mode 1`), or by the length of the query/centre alnRes/qLen (`--cov-mode 2`);
- a minimum sequence identity (`--min-seq-id`) with option `--alignment-mode` defined as the number of identical aligned residues divided by the number of aligned columns including internal gap columns, or, by default, defined by a highly correlated measure, the equivalent similarity score of the local alignment (including

gap penalties) divided by the maximum of the lengths of the two locally aligned sequence segments. The score per residue equivalent to a certain sequence identity is obtained by a linear regression using thousands of local alignments as training set.

You can count the number of entries in your clustered FASTA file `clusteredProts.faa_all_seqs.faa` again using the previous `grep` command.

Learn how to deal with MMseqs2's indexed databases The previous `easy-cluster` command is a shorthand to deal directly with FASTA files as input and output. However, MMseqs2's modules **do not** use the FASTA format internally. Since the goal of this tutorial is to make you an expert in using MMseqs2 workflows, we will explain and use the MMseqs2 database formats and create FASTA files only for downstream tools.

You can convert a FASTA file to the MMseqs2 database format using:

```
mmseqs createdb plassProteins.faa plassProteinsDb
```

This generates several files: a *data* file `plassProteinsDb` together with an *index* file `plassProteinsDb.index`. The first file contains all the sequences separated by a null byte `\0`. We coin more generally any data record *an entry*, and each entry is associated with a unique *key* (integer number) that is stored in the *index* file.

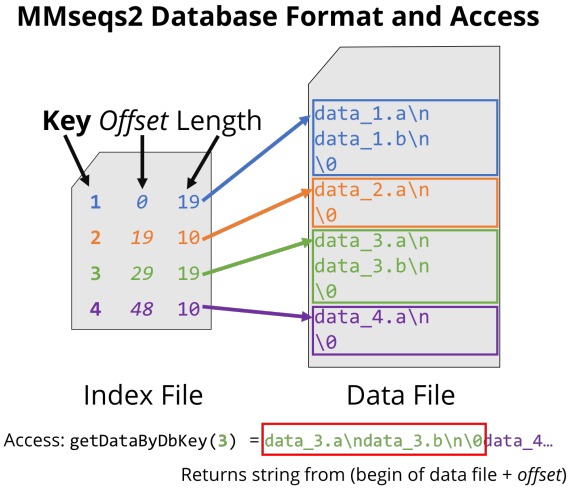

Figure 2: MMseqs2 database format, through which all MMseqs2 modules can be easily and efficiently chained.

The corresponding headers are stored in a separate database with a `_h` suffix (`plassProteinsDb_h`).

The `.dbtype` file helps to keep track of the database type (amino-acid, nucleic, profile, etc.).

The `.lookup` file is a lookup file linking the keys to their sequence accession as specified in their FASTA header.

The format of this index file is tab-separated and reports one line per entry in the database, specifying a unique key (column 1), the offset and the length of the corresponding data (columns 2 and 3 respectively). As we will make use of the efficient structure later on in the tutorial, you can already take a look at the index file structure with:

```
head plassProteinsDb.index
```

We provide a simple bash function to access a single entry of the file.

With it, you can access data for *an entry* with the `getentry` command. This command takes the data filename (e.g. plassProteinsDb) and the `ENTRYOFFSET` (second column of the index) and the `ENTRYLENGTH` (third column of the index). Please try to access a single entry of the datafile.

```
getentry FILENAME ENTRYOFFSET ENTRYLENGTH
```

Let's re-run the clustering of the catalogue database with our fresh database:

```
mmseqs cluster plassProteinsDb plassProteinsReduced tmpDir \
    --min-seq-id 0.9 -c 0.95 --cov-mode 1
```

This creates a *cluster database* where **each entry has the key of its representative sequence**, and whose data consists of the list of keys of its members:

```
# the index file contains entries whose
# keys are of those of their representative sequence
head plassProteinsReduced.index

# you will see the keys belonging to different clusters
# (one per line) and such that every cluster is
# separated by a null byte (shown as a • in your C9 IDE
# or as ^@ in vim
c9 open plassProteinsReduced
```

Note: This is a general principle in MMseqs2: the keys are always consistent between databases (input and output, clustering and sequence databases, and potentially intermediate files.

Therefore, to count the number of clusters, you can count the number of lines in the *index* file using:

```
wc -l plassProteinsReduced.index
```

3.2 Annotating the catalogue

To get domain annotation for your protein catalogue, we search the sequences against a database of Pfam profiles:

```
# first create a sequence database of
# the representative sequences
cat plassProteinsReduced.index > keysOfRepSeq
#  sequence db
mmseqs createsubdb keysOfRepSeq plassProteinsDb\
    plassProteinsNr
#  do the same for its headers
mmseqs createsubdb keysOfRepSeq plassProteinsDb_h \
    plassProteinsNr_h


# -s 7 for high sensitivity
mmseqs search plassProteinsNr \
    $HOME/mmseqs2tutorial/databases/pfamProfiles \
    searchOutProfile.mmseqs tmp -s 7 --no-preload

# you can make it human-readable in TSV format
mmseqs createtsv plassProteinsNr \
    $HOME/mmseqs2tutorial/databases/pfamProfiles \
    searchOutProfile.mmseqs searchOutProfile.mmseqs.tsv
```

Learn how to filter databases You can post-process the annotation file to retrieve only annotations of high confidence:

```
# check that the e-values are shown in column 4
# of the search result file
head searchOutProfile.mmseqs

# create a new database containing
# only annotations of e-value <= 1e-5
mmseqs filterdb searchOutProfile.mmseqs \
    strongPfamAnnotations --filter-column 4 \
    --comparison-operator le --comparison-value 1e-5
```

An advanced way to extract the entries that did not get a reliable annotation uses the fact that if no hit was found for a given sequence, the corresponding entry in the *data file* will be empty, resulting in a data length of 1 (for the null byte) in the *index file*:

```
# extract the keys of entries having
# no annotation better than 1e-5
awk '$3==1 {print $1}' strongPfamAnnotations.index \
   > uncertainFunction.keys

# create a FASTA file for further investigation
# in downstream tools (HHblits for instance)

# First extract sub-database containing
# the protein sequences of uncertain function
#   - sequence db
mmseqs createsubdb uncertainFunction.keys plassProteinsNr \
    plassProteinsNr.uncertainFunc
#   - do the same for its headers
mmseqs createsubdb uncertainFunction.keys plassProteinsNr_h \
    plassProteinsNr.uncertainFunc_h

# convert it to a fasta file
mmseqs convert2fasta plassProteinsNr.uncertainFunc \
    plassProteinsNr.uncertainFunc.faa
```

What is the difference to standard annotation tools? You can compare (time and number of annotation) with HMMER3:

```
time hmmscan --notextw --noali --tblout "hmmer.tblout" \
    --domtblout "hmmer.domtblout" \
    $HOME/mmseqs2tutorial/databases/Pfam-A.full_hmmer \
    clusteredProts.faa

# check the number of annotated proteins
#   for MMseqs2
awk '$3 > 1 {print $1}' searchOutProfile.mmseqs.index|wc -l
#   for HMMER
tail -n+4 hmmer.tblout | cut -c 21-30 | sort -u | wc -l
```

4 Build you own workflows

4.1 Cascaded profile clustering (deep clustering)

We will take advantage of MMseqs2's **modular architecture** to create a workflow (bash script) that calls MMseqs2 tools to deeply cluster a set of proteins.

Cascaded sequence clustering Let's first create a cascaded clustering workflow: after a first clustering step, the representative sequences of each of the clusters are searched against each other and the result of the search is again clustered. By repeating this procedure iteratively, one gets a deeper clustering of the original set.

Try to code complete the following script:

```
#!/bin/bash -e

inputdb="plassProteinsReduced"
for step in {1..maxStep}; do
    # we do not use the cluster mmseqs command as it
    # already is a cascaded workfolw clustering
    mmseqs search $inputdb $inputdb searchStep$step tmpDir
    mmseqs clust $inputdb searchStep$step clusteringStep$step
    mmseqs result2repseq ...
    inputdb=...
done

# Then merge the clustering steps
mmseqs mergeclusters plassProteinsReduced deepClusterDB ...
```

Try your script with 3 steps, and check the clustering depth (number of clusters) at each step:

```
wc -l clusteringStep*.index
```

What do you notice ?

Deeper clustering using profiles To make a deeper clustering of your protein set, one idea is to create a cascaded clustering where the sequence search at every iteration is replaced by a profile to sequence search (more sensitive search than sequence to sequence searches). Write your own workflow that will be using the `result2profile` module. After adjusting your workflow to handle profiles also add the `--add-self-matches` parameter to the search to assure that the query is contained in each search results.

Did you manage to cluster more deeply?

You can get the distribution of your cluster sizes by calling:

```
# get the size of every cluster
mmseqs result2stats plassProteinsReduced plassProteinsReduced \
    deepClusterDB deepClusterDB.clusterSizes --stat linecount
# show the distribution
tr -d '\0' < deepClusterDB.clusterSizes | sort -n | uniq -c
```

Compare this clustering to your first cascaded clustering.

4.2 Abundance analysis

Let's check the most abundant genes in our dataset. To this end, we want to map the ORFs from the reads to the protein catalogue, and count the number of hits on each of the proteins.

Write an MMseqs2 workflow that:

- creates a read sequence database from the FASTA files (`createdb`)
- extracts the ORFs from the read database (`extractorfs`),
- translate the nucleotide ORFs to protein sequences (`translatenucs`),
- map them on the proteins (`prefilter` with option `-s 2` since mapping calls for high sequence identity),
- score the prefilter hits with a gapless alignment (`rescorediagonal` with option `-c 1 --cov-mode 2 --min-seq-id 0.95 --rescore-mode 2 -e 0.000001 --sort-results 1` to have significant hits and fully covered by the protein sequence),
- keep the best mapping target (`filterdb` with flag `--extract-lines 1`),
- in the database output from the last step, every entry is a read-ORF. Now you should transpose the database so that each contig is an entry, and the data is the list of mapped read-ORFs (`swapresults`),
- count the number of mapped read-ORF (`result2stats`). Advanced: instead of just count the reads, sum up the aligned length and divide it by the contig length `apply` and `awk`. The contig length and alignment length (alnEnd - alnStart) is in the alignment format.

We can use `createtsv` to output the abundances as a TSV formatted file:

```
# Create a flat file
mmseqs createtsv plassProteinsReduced plassProteinsReduced\
    mapAbundances mapAbundances.tsv --target-column 0
```

4.3 Re-create the linclust workflow... if you have time :)

Based on the explanation of the Linclust algorithm, try to code its workflow using:

- `kmermatcher`
- `rescorediagonal`
- `clust`

Take a look at the real Linclust workflow in `\$HOME/mmseqs2tutorial/software/MMseqs2/data/linclust.sh`. This version is slightly more involved as it integrates a redundancy reduction step (the `pre_clust` prefiltering by high Hamming distance), and uses a trick using `filterdb` with the flag `--filter-file` to apply the workflow you just built *only* on the *non-redundant* entries. At the end of the file, you can also spot a merging step to recover the redundant part in the final clustering.

5 Conclusion

We hope that you are more familiar with the MMseqs2 environment, and that you enjoy its modularity and flexibility for creating new workflows. Due to time and virtual machine constraints we chose a rather small metagenomic dataset, but using MMseqs2 on bigger datasets should convince you of its scalability.

Please help us by giving critical feedback on this tutorial!

References

Eddy, S.R. (2009). A new generation of homology search tools based on probabilistic inference. In *Genome Informatics 2009: Genome Informatics Series Vol. 23*, pages 205–211. World Scientific.

Hyatt, D. et al (2010). Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, **11**(1), 119.

Li, D. et al (2015). Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, **31**(10), 1674–1676.

Steinegger, M. and Söding, J. (2017). Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, **35**(11), 1026.

Steinegger, M. and Söding, J. (2018). Clustering huge protein sequence sets in linear time. *Nat. Commun.*, **9**(1), 2542.

Steinegger, M. et al (2018). Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold. *bioRxiv*.