MetaG ECCB'18 tutorial

Part II: large scale sequence analysis

Clovis Galiez, Milot Mirdita, Johannes Söding

Sept. 2018

1 Introduction

MMseqs2 (Many-against-Many sequence searching [4]) is a software suite to search and cluster huge protein sequence sets. MMseqs2 is open source GPL-licensed software implemented in C++ for Linux, MacOS, and (as beta version, via cygwin) Windows. The software is designed to run on multiple cores and servers and exhibits very good scalability. MMseqs2 can run 10000 times faster than BLAST. At 100 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: MMSeqs2, your new friend.

You will learn here the basic usage of MMSeqs2 as well as expert trick to take advantage of the ability of chaining different MMSeqs2 modules to produce custom workflows. We will show on an human gut metagenomic dataset (SRA run `ERR1384114`) what are

the advantages of using MMseqs2 and Plass (Protein Level Assembly) over the more conventional pipeline MegaHit[3]+Prodigal[2]+HMMER[1].

2 Set up and discover your environment

2.1 Command line interface and software

You will work on your own remote machine through a Bash terminal in your browser. Please connect through the following link: `https://17.llil.de/ide.html`. All software and data has been installed before this tutorial. The commands to reproduce the installation is available in the file `scripts/setup.sh` on GitHub of the tutorial `https://github.com/soedinglab/metaG-ECCB18-partII`.

All the software is already installed in `~/software` and intergated in your `$PATH` environment variable.

You can take a look to the helper of 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 Fastq reads file you are going to work on has been downloaded in the directory `~/data`.

3 Analysis of a human gut metaG dataset

3.1 Getting a catalog of proteins

We propose to use Plass to assemble a catalog of protein sequences directly from the reads, without the nucleic assembly step. The standard nucleic assemblies prevent many reads to assemble due to low **coverages** and **microdiversity**. To run this protein-level assembly, use the command

```
plass assemble readsFastqFile plassProteins.faa tmpDir
```

or type `plass assemble -h` for more options.

As a matter of comparison, run the usual pipeline using MegaHit for nucleic assembly (`megahit -r readsFastqFile -o megahitAssemblyDir`), and then extract the proteins using Prodigal (`prodigal -i contigFile.fna -a prodigal.faa`.

Take a look to the fasta files produced by Plass and Prodigal, or directly compare the number of detected proteins using:

```
cat file.faa|grep -c ">"
```

Redundancy-reduce There is actually some redundancy in the catalog generated by Plass. You can reduce this catalog by clustering it for instance to 90% of sequence identity and asking for the representative to cover at least 95% of the members:

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

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

Learn how to deal with FF-index The previous *easy-cluster* command is a shorthand for a workflow allowing to deal with fasta files as input and output. However, MMSeqs2 tools **do not** deal natively with Fasta format. They prefer the efficient FFindex format. Since the goal is to make you an expert in the MMSeqs2 workflow, we will now only use FFindex databases and create fasta files only for branching to external tools.

You can convert a Fasta to am FFindex using:

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

This generated several files 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.

The corresponding headers are stored in a separate FFindex database with `_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 identifier 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 to the index file structure with:

```
head plassProteinsDb.index
```

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

```
mmseqs cluster plassProteinsDb plassProteinsReduced\
    --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 vi)
vi plassProteinsReduced
```

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

Therefore, to count the number of proteins left in the clustered database, you can count the number of entries using:

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

3.2 Annotating the catalog

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

```
# first create a sequence database of
# the representative sequences
cut -f1 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
time mmseqs search plassProteinsNr ~/databases/pfamProfiles\
    searchOutProfile.mmseqs tmp -s 7 --no-preload

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

Learn how to filter databases You can post-process the annotation file to get the strong annotations:

```
# 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
```

There is a hacky way for extracting the entries that did not get a reliable annotation, it uses the fact that if there is no found hit for a given sequence, the corresponding entry will be empty, resulting in a data length of 2 (for \n and null byte):

```
# extract the keys of entries having
# no annotation better than 1e-5
awk '$3==2{print $1}' strongPfamAnnotations.index\
   >uncertainFunction.keys

# create a Fasta file of them for
# further investigations (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 with the standard annotation tool? You can compare (time and number of annotation) with the HMMER tool:

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

# check the number of annotated proteins
#   for MMSeqs2
awk '$3>2{print $1}' searchOutProfile.mmseqs.index|wc -l
#   for HMMER
tail -n+4 hmmer.tblout|cut -c 21-30|sort -u|wc -l
```

A full benchmark between the precision and sensitivity is available at `TODO:linktoMilot' spaperwhenavail`.

4 Build you own workflows

4.1 Cascaded profile clustering (deep clustering)

We will see how we can take advantage of the **MMSeqs2 modular structure** for creating 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 iteratively this procedure, one gets a deeper clustering of the original set.

Try to code yourself the following *partial* pseudo-code:

```
inputdb="plassProteinsReduced"
for step in 1:maxStep
    # 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 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). Code your own workflow that will be using the MMSeqs2 `result2profile` commands.

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
cat deepClusterDB.clusterSizes|tr -d '\0'|sort -n|uniq -c
```

You can compare it to your first cascaded clustering.

4.2 Abundance analysis

TODO, normalize by the gene length Let's check the most abundant genes in our dataset. To this end, we want to map the orfs from the reads on the protein catalog, and count the number of hits on each of the proteins. Code an MMSeqs2 workflow that:

- extracts the ORFs from the reads (`extractorfs`),
- map them on the proteins (`prefilter` with option `-s 1` 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` to have significant hits and fully covered by the protein sequence),
- sort the hits by evalue (`filterdb` and guess the right flags!),
- 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`).

To output the abundances as a TSV format, there is a hacky way:

```
# Create a flat file
mmseqs result2flat plassProteinsReduced plassProteinsReduced\
    mapAbundances mapAbundances.flat

# Check out the results
head mapAbundances.flat

# Transform this fasta-like format to TSV
awk '$0~/^>/{name=$0;next}{print name"\t" $0}'\
    mapAbundances.flat|\
    sort -n -k2,2|sed 's/>//' >abundances.tsv
```

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/ubuntu/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. You can also spot in the end 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 virtual machine constraints and that we were willing to spend time only on practicing 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

[1] Sean R Eddy. 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, 2009.

[2] Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC bioinformatics*, 11(1):119, 2010.

[3] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[4] Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026, 2017.