MMseqs user guide

MMseqs suite for fast and sensitive batch searching and clustering of huge protein sequence sets

(c) 2014 Maria Hauser, Martin Steinegger, and Johannes Soeding

1 Summary

MMseqs (Many-against-Many sequence searching) is a software suite for very fast protein sequence searches and clustering of huge protein sequence data sets. MMseqs is around 1000 times faster than protein BLAST and sensitive enough to capture similarities down to less than 30% sequence identity.

At the core of MMseqs are two modules for the comparison of two sequence sets with each other - the prefiltering and the alignment modules. The first, prefiltering module computes the similarities between all sequences in one set with all sequences in the other based on a very fast and sensitive alignment-free metric, the sum of scores of similar k-mers. The alignment module implements an SSE2-accelerated Smith-Waterman-alignment of all sequences that pass a cut-off for the prefiltering score in the first module. Both modules are parallelized to use all cores of a computer to full capacity. Due to its unparalleled combination of speed and sensitivity, searches of all predicted ORFs in large metagenomics data sets through the entire UniProtKB or NCBI-NR databases are feasible. This could allow for assigning to functional clusters and taxonomic clades many reads that are too diverged to be mappable by current software.

MMseqs' clustering module can cluster sequence sets efficiently into groups of similar sequences. It takes as input the similarity graph obtained from the comparison of the sequence set with itself in the prefiltering and alignment modules. MMseqs further supports an updating mode in which sequences can be added to an existing clustering with stable cluster identifiers and without the need to recluster the entire sequence set. We will use MMseqs to regularly update versions of the UniProtKB database clustered down to 20-30% sequence similarity threshold.

2 Installation

First, set environment variables:

```
$ export MMDIR=$HOME/path/to/mmseqs/
$ export PATH=$PATH:$MMDIR/bin
```

MMseqs uses ffindex, a fast and simple database for wrapping and accessing huge amounts of small files. Setting the environment variable `LD_LIBRARY_PATH` ensures that ffindex binaries are in the path:

```
$ export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:$MMDIR/lib/ffindex/src
```

Then create MMseqs binaries:

```
$ cd $MMDIR/src
$ make
```

MMseqs binaries are now located in `$MMDIR/bin.`

3 Getting started

Here we explain how to run a search for matches of sequences in the query database in the target database and how to cluster a database. Test data (a query and a target database for the sequence search and a database for the clustering) are stored in `$MMDIR/data.`

Search

You can use the query database `queryDB.fasta` and target database `targetDB.fasta` to test the search workflow.

Before clustering, you need to convert your database containing query sequences (`queryDB.fasta`) and your target database (`targetDB.fasta`) into ffindex format:

```
$ fasta2ffindex queryDB.fasta queryDB
$ fasta2ffindex targetDB.fasta targetDB
```

It generates ffindex database files, e. g. `queryDB` and ffindex index file `queryDB.index` from `queryDB.fasta`. Then, generate a directory for tmp files:

```
$ mkdir tmp
```

Please ensure that in case of large input databases `tmp` provides enough free space. For the disc space requirements, see the section

To run the search, type:

```
$ mmseqs_search queryDB targetDB outDB tmp
```

Then, convert the result ffindex database into a FASTA formatted database:

```
$ ffindex2fasta outDB outDB.fasta
```

Clustering

Before clustering, convert your FASTA database into ffindex format:

```
$ fasta2ffindex DB.fasta DB
```

Then, generate a directory for tmp files:

```
$ mkdir tmp
```

Run the cascaded clustering of your database and output the result into the database files `DB_clu`, `DB_clu.index`:

```
$ mmseqs_cluster DB DB_clu tmp --cascaded
```

To generate a FASTA-style formatted output file from the ffindex output file, type:

```
$ ffindex2fasta DB_clu DB_clu.fasta
```

To run the more sensitive cascaded clustering and convert the result into FASTA format, type:

```
$ mmseqs_cluster DB DB_clu_s7 tmp --cascaded -s 7
$ ffindex2fasta DB_clu_s7 DB_clu_s7.fasta
```

4 System requirements

MMseqs runs under Linux only. Alignment and prefiltering modules are fully parallelized with SSE2 and OpenMP, i. e. MMseqs runs fastest on a computer with many (e.g. 16-32) cores. Besides, MMseqs needs much memory (you need 128G of memory in order to cluster the current UniProtKB version containing 54 M sequences). We offer an option for limiting the memory use at the cost of longer runtimes. The database is split into chunks and the program only holds one chunk in memory at any time. For clustering large databases containing tens of millions of sequences, you should provide enough free disc space (≈500 GB). In section 11, we will discuss the runtime, memory and disc space consumption of MMseqs and how to reduce resource requirements for large databases.

5 ffindex database format

All modules take ffindex databases as input and produce ffindex databases as output. ffindex was developed to avoid drastically slowing down the file system when millions of files need to be written and accessed. ffindex hides the files from the file system by storing them as unstructured data records in a single *data file*. In addition to this data file, an ffindex database includes a second file: This *index file* stores for each unique accession code the start position in bytes of the data record in the ffindex data file. When transforming a FASTA file with multiple sequences into an ffindex database, the accession code is the ID of the sequence parsed from the header. If no ID can be identified, the accession code is the whole header without the > character before the first blank space.

The binaries `fasta2ffindex` and `ffindex2fasta` located in `mmseqs/bin` do the format conversion from and to the ffindex database format. `fasta2ffindex` generates a ffindex database from a FASTA sequence database. `ffindex2fasta` converts an ffindex database to a FASTA formatted text file: the headers are ffindex accession codes preceded by >, with the corresponding dataset from the ffindex data file following.

However, for a fast access to the particular datasets in very large databases it is advisable to use the ffindex database directly without converting. We offer the binary `ffindex_get` for direct access to the datasets stored in an ffindex database.

6 Overview of folders in MMseqs

- `bin`: MMseqs binaries, `fasta2ffindex` and `ffindex2fasta` binaries
- `data`: BLOSUM matrices and test data
- `lib`: ffindex sources and binaries
- `src`: MMseqs sources and the Makefile

7 Overview of MMseqs commands

MMseqs contains six binaries. Three commands execute workflows that combine MMseqs core modules. The other three commands execute the single modules which are used by the workflows and should be used by more advanced users.

Workflows:

- `mmseqs_search:` Compares all sequences in the query database with all sequences in the target database.
- `mmseqs_cluster:` Clusters the sequences in the input database by sequence similarity.
- `mmseqs_update:` Given an the existing clustering of a sequence database and a new version of the sequence database with some new sequences being added and others having been deleted, MMseqs incrementally updates the clustering.

Single modules:

- `mmseqs_pref:` Computes k-mer similarity scores between all sequences in the query database and all sequences in the target database.
- `mmseqs_aln:` Computes Smith-Waterman alignment scores between all sequences in the query database and the sequences of the target database whose prefiltering scores computed by `mmseqs_pref` pass a minimum threshold.
- `mmseqs_clu:` Computes a similarity clustering of a sequence database based on Smith Waterman alignment scores of the sequence pairs computed by `mmseqs_aln`.

8 Description of workflows

8.1 Batch sequence searching using `mmseqs_search`

For searching a database, you need your query and target database in ffindex format and an empty directory for MMseqs temporary files. Then, you can run the search by typing

```
$ mmseqs_search queryDB targetDB outDB tmp
```

To get more sensitive results, increase the search sensitivity (`-s` option):

```
$ mmseqs_search queryDB targetDB outDB tmp -s 7
```

The default sensitivity is 4, sensitivity can be set in the range $[1 : 9]$.

This workflow combines the prefiltering and alignment modules into a fast and sensitive batch protein sequence search that compares all sequences in the query database with all sequences in the target database. Query and target databases may be identical. The program outputs for each query sequence all database sequences satisfying the search criteria such as sensitivity of the search.

The underlying algorithm is explained in more detail in section 9.1, and the full parameter list can be found in section 12.1.

8.2 Clustering databases using `mmseqs_cluster`

For clustering a database, your need your sequence database in ffindex format and an empty directory for MMseqs temporary files. Then, you can run the clustering with

```
$ mmseqs_cluster inDB outDB tmp
```

and cascaded clustering with

```
$ mmseqs_cluster inDB outDB tmp --cascaded
```

For more sensitive clustering, adjust the sensitivity (`-s` option):

```
$ mmseqs_cluster inDB outDB tmp --cascaded -s 7
```

The clustering workflow combines the prefiltering, alignment and clustering modules into a simple clustering or cascaded clustering of a database. There are two possibilities to run the clustering:

- Simple clustering runs the prefiltering, alignment and clustering modules with predefined parameters.
- Cascaded clustering clusters the sequence database using prefiltering, alignment and clustering modules incrementally in three steps. In the first step, the prefiltering runs with a low sensitivity of 1 and a very high results significance threshold in order to accelerate the calculation and search only for hits with a high sequence identity. Then alignments are calculated and the database is clustered. The second step takes the representative sequences of the first clustering step and repeats the prefiltering, alignment and clustering steps. This time, the prefiltering is run with a higher sensitivity

and a lower result significance threshold for catching sequence pairs with lower sequence identity. In the third step, the whole process is repeated again with the target sensitivity defined by the `-s` parameter. Eventually, the clustering results are merged and the resulting clustering is written to the output ffindex database.

Cascaded clustering yields more sensitive results than simple clustering. Also, it allows very large cluster sizes in the end clustering resulting from cluster merging (note that cluster size can grow exponentially in the cascaded clustering workflow), which is not possible with the simple clustering workflow because of the limited maximum number of sequences passing the prefiltering and the alignment. Therefore, we strongly recommend to use cascaded clustering especially to cluster larger databases and to obtain maximum sensitivity.

The underlying algorithm is explained in more detail in section 9.1, and the full parameter list can be found in section 12.2.

8.3 Updating a database clustering using `mmseqs_update`

To run the updating, you need the old and the new version of your sequence database in ffindex format, the clustering of the old database version and a directory for the temporary files:

```
$ mmseqs_update oldDB newDB oldDB_clustering outDB tmp
```

This workflow efficiently updates the clustering of a database by adding new and removing outdated sequences. It takes as input the older sequence database, the results obtained by this older database clustering, and the newer version of the sequence database. Then it adds the new sequences to the clustering and removes the sequences that were removed from the newer database. Sequences which are not similar enough to any existing cluster will be founders of new clusters.

9 Description of core modules

For advanced users, it is possible to run core modules for maximum flexibility. Especially for the sequence search it can be useful to adjust the prefiltering and alignment parameters according to the needs of the user. The detailed parameter lists for the modules is provided in section 12.

MMseqs contains three core modules: prefiltering, alignment and clustering.

9.1 Computation of prefiltering scores using `mmseqs_pref`

The prefiltering module calculates the sum of scores of similar k-mers between all query sequences and all database sequences and outputs the most similar sequence pairs.

If you want to *cluster* a database, or do an all-against-all search, you only have one input database. In this case, the prefiltering does an all-against-all search with the following program call:

```
$ mmseqs_pref inputDB inputDB resultDB_pref
```

`inputDB` is the base name of the ffindex databases you produced from your FASTA sequence databases, the prefiltering results are stored in the ffindex database files `resultDB_pref`, `resultDB_pref.index`.

If you want to do a *sequence search*, you have two input databases: query database and target database, that also can be identical. In this case, the prefiltering program call is:

```
$ mmseqs_pref queryDB targetDB resultDB_pref
```

First, the database sequences are indexed in an index table to provide a fast access to the k-mers of the database sequences. The index table has an array with a pointer for each possible k-mer to an index list storing the IDs of the database sequences containing this k-mer. After the index table generation, the algorithm processes each query sequence from left to right and generates a list of similar k-mers for the k-mer at the current query sequence position. For each k-mer in the list, the k-mer similarity score is added to the overall prefiltering score for each database sequence containing this k-mer, retrieved using the index table. After processing the whole query sequence, database sequences with significant prefiltering scores are extracted and written to the prefiltering result database.

Since different queries yield different score distribution in the database, a rigid prefiltering score threshold does not work well. The statistical significance of a prefiltering score for a given query sequence is described by the Z-score. The Z-score for a prefiltering score of a query sequence and a database sequence is calculated based on the score distribution for the query in the database. For each query and database sequence pair, the prefiltering score S_{qt} is normalized by subtracting the background score S_0 expected by chance and dividing by the standard deviation of the score σ_S, resulting in a normalized Z-score Z_{qt}:

$$Z_{qt} = \frac{S_{qt} - S_0}{\sigma_S}$$

Instead of setting a prefiltering score threshold, we set a rigid Z-score (i. e. result significance) threshold. Only results with a sufficient Z-score are written to the output.

The sensitivity of the prefiltering can be set using the `-s` option. Internally, `-s` sets the average length of the lists of similar k-mers per query sequence position and the Z-score threshold.

- *Similar k-mers list length:* Low sensitivity yields short similar k-mer lists. Therefore, the speed of the prefiltering increases, since only short k-mer lists have to be generated and less lookups in the index table are necessary. However, the sensitivity of the search decreases, since only very similar k-mers are generated and therefore, the prefiltering can not identify sequence pairs with low sequence identity.
- *Z-score threshold:* Z-score of a prefiltering result describes its statistical significance.

Lower sensitivity yields a higher Z-score threshold, i. e. only the most significant results are displayed.

Furthermore, there is a possibility to use different lengths of the k-mers used in the prefiltering. Longer k-mers are more sensitive, since they cause less chance matches. On the other hand, for a fixed run time, longer k-mers only pay off for larger databases. The reason is the different relation between the time for the k-mer list generation and database matching for different database sizes. Longer k-mers need more time for the k-mer list generation, but less time for database matching. Therefore, the database matching should take most of the computation time, which is only the case for large databases. For a fixed run time, the default value $k = 6$ is the best for databases containing a few million sequences. For very large databases containing about 100 million sequences, $k = 7$ should be a better choice theoretically, though the real life performance of 7-mers on large databases is not tested yet. For databases containing only hundreds of thousands of sequences, $k = 5$ should be sufficient.

9.2 Local alignment of prefiltering sequences using `mmseqs_align`

In the alignment module, you can also specify either identical or different query and target databases. If you want to do a clustering in the next step, query and target database need to be identical.

```
$ mmseqs_align inputDB inputDB resultDB_pref resultDB_aln
```

Alignment results are stored in the ffindex files `resultDB_aln`, `resultDB_aln.index`.

Program call in case you want to do the sequence search and have different query and target databases:

```
$ mmseqs_align queryDB targetDB resultDB_pref resultDB_aln
```

This module implements an SSE2-accelerated Smith-Waterman-alignment (Farrar, 2007) of all sequences that pass a cut-off for the prefiltering score in the first module. It processes each sequence pair from the prefiltering results and aligns them in parallel, calculating one alignment per core at a single point of time. Additionally, the alignment calculation is vectorized using SIMD (single instruction multiple data) instructions. Eventually, the alignment module calculates alignment statistics such as sequence identity, alignment coverage and e-value of the alignment.

9.3 Clustering sequence database using `mmseqs_clu`

For the clustering, you need the input sequence database and the alignment results for the database:

```
$ mmseqs_clu inputDB resultsDB_aln resultsDB_clu
```

Clustering results are stored in the ffindex database files `resultsDB_clu`, `resultsDB_clu.index`.

The clustering module offers the possibility to run two different clustering algorithms. A greedy set cover algorithm is the default. It tries to cover the database by as few clusters as possible. At each step, it forms a cluster containing the representative sequence with the most alignments above the special or default thresholds with other sequences of the database and these matched sequences. Then, the sequences contained in the cluster are removed and the next representative sequence is chosen.

The other clustering algorithm is a greedy clustering algorithm, as used in CD-HIT. It sorts sequences by length and in each step forms a cluster containing the longest sequence and sequences that it matches. Then, these sequences are removed and the next cluster is chosen from the remaining sequences.

Note that we *always* recommend to use the cascaded clustering workflow instead of the clustering module for larger databases, since the maximum cluster size is limited to a quite low value otherwise (between 50 and 300 for large databases containing millions of sequences, depending on the database size). The reasons are the limited result list length in the prefiltering and alignment modules (the maximum list length determines the maximum cluster size in the simple clustering workflow) and the high memory consumption of the clustering for large databases with many alignment results per query.

10 Output file formats

Results of MMseqs commands are stored in ffindex databases. All records within those ffindex databases are in plain text format.

10.1 Prefiltering

The ffindex accession code is the UniProtKB ID (or other ID depending on the database format) of the query. A line in the prefiltering result database record (= one match) has the following format:

```
targetId Z-score prefilteringScore
```

where `targetId` is the database ID of the matched sequence, `Z-score` is the statistical significance score of the match and `prefilteringScore` is the raw score of the match (the sum of the scores of similar k-mers of the query and target sequence) in half bits. Example of a prefiltering result for the SwissProt sequence Q54G30 (excerpt):

```
Q54G30 1177.95 55735
Q869W0 159.179 5274
Q86IM3 99.2044 1823
Q54E43 85.8743 3224
```

The first match is the identity Q54G30 having a very high prefiltering score of 55735 and the Z-score of 1177.95.

10.2 Alignment

The ffindex accession code is the UniProtKB ID (or other ID depending on the database format) of the query. One line of the alignment results record has the following format:

```
targetId alnScore queryCov targetCov seqId evalue
```

where `targetId` is the database ID of the matched sequence, `alnScore` is the raw score of the alignment in half bits, `queryCov` is the alignment coverage of the query in the range [0 : 1], `targetCov` is the alignment coverage of the target database sequence in the range [0 : 1], `seqId` is the sequence identity and `evalue` is the e-value of the match. Example of an alignment result for the SwissProt sequence A0PUH6 (excerpt):

```
A0PUH6 1305 1.000 1.000 1.000 1.507e-186
Q6NFN4 824 0.956 0.974 0.649 3.682e-114
Q8DD39 256 0.900 0.909 0.335 1.136e-28
P52973 182 0.808 0.822 0.238 1.597e-17
```

The first line is the identity match. The last sequence P52973 has a Smith-Waterman alignment score 182, query sequence coverage 0.808, database sequence coverage 0.822, the alignment has the sequence identity 0.238 and the e-value 1.597e-17.

10.3 Clustering

Every cluster is stored once (i. e. one result database record per cluster). Each database record contains UniProtKB IDs (or other IDs depending on the database format) of the sequences assigned to this cluster, one ID per line. The ffindex accession code is the ID of the representative sequence of the cluster. An example of a cluster record with 4 cluster members:

```
Q9ZZZ1
Q96189
O03850
P03887
```

11 Optimizing sensitivity and consumption of resources

This section discusses how to keep the run time, the memory and disc space consumption of MMseqs at reasonable values, while obtaining results with the highest possible sensitivity. These considerations are relevant if the size of your database exceeds several millions of sequences and they are most relevant if the database size is in the order of tens of millions of sequences.

11.1 Prefiltering module

The prefiltering module can use a lot of resources regarding all the memory consumption, the runtime and the disc space, if the parameters are not set appropriately.

Memory consumption For maximum efficiency of the prefiltering, the entire database should be held in RAM memory. The major part of memory is required for the k-mer index table of the database. For a database containing N sequences with an average length L, the memory consumption of the index lists is $N \times L \times 4$B. Note that the memory consumption grows linearly with the size of the sequence database. In addition, the index table stores the pointer array and two auxiliary arrays with the memory consumption of $a^k \times 16$B, where a is the size of the amino acid alphabet (usually 21 including the unknown amino acid X) and k is the k-mer size. The overall memory consumption of the index table is

$$M = (4\,N\,L + 16\,a^k)\mathrm{B}$$

Therefore, the UniProtKB database version of April 2014 containing 55 million sequences with an average length 350 needs about 71 GB of main memory.

To limit the memory use at the cost of longer runtimes, the option `--max-chunk-size` allows the user to split the database into chunks of the given maximum size.

Runtime The prefiltering module is the most time consuming step. To cluster the 55 million sequences of UniProtKB (04/2014), the MMseqs prefiltering module needs about 6 days when running on 32 cores and about 10 days when running on 16 cores of a modern computer.

Disc space The prefiltering results for very large databases can grow to considerable sizes (in the order of TB) of the disc space if very long result lists are allowed and a low Z-score threshold is set. As an example, an all-against-all prefiltering run on the UniProtKB with `--max-seqs` 300 yielded average prefiltering list length 150 and the output file size 146 GB.

Important options for tuning the memory, runtime and disc space usage

- The option `-s` controls the sensitivity in the MMseqs prefiltering module. The lower the sensitivity, the faster the prefiltering gets at the cost of search sensitivity. Default sensitivity is 4, increasing the sensitivity by one roughly doubles the runtime of the prefiltering. In order to cluster the UniProtKB down to ≈30% sequence identity, you should leave this parameter at the default value of 4. For clustering down to 90%, sensitivity 1 should be sufficient, although there are still no specific tests for the optimum parameters necessary for clustering down to a fixed sequence identity.

- The option `--max-seqs` controls the maximum number of prefiltering results per query sequence. For very large databases (tens of millions of sequences), it is a good advice to keep this number at reasonable values (i. e. the default value 300). For considerably larger values of `--max-seqs`, the size of the output can be in the range of several TB of disc space for databases containing tens of millions of sequences. Changing `--max-seqs` option has no effect on the run time.

- The option `--z-score` describes the minimum significance of the results written to the output. Usually, this option is set automatically depending on the sensitivity. However, especially for the sequence search it can be desired to see also less significant results. Setting `--z-score` at lower values yields more results and therefore increases the size of the output written to disc. In addition, it slows down the program.

11.2 Alignment module

In the alignment module, generally only the runtime is a critical issue.

Memory consumption The major part of the memory is required for the three dynamic programming matrices, once per core. Since most sequences are quite short, the memory requirements of the alignment module for a typical database are in the order of a few GB.

Runtime It takes about 2-3 days to compute Smith-Waterman alignments for the UniProtKB sequence pairs which passed the prefiltering step (at default parameters for deep clustering down to ≈20 - 30% pairwise sequence identity).

If a huge amount of alignments has to be calculated, the run time of the alignment module can become a bottleneck. The run time of the alignment module depends essentially on two parameters:

- The option `--max-seqs` controls the maximum number of sequences aligned with a query sequence. By setting this parameter to a lower value, you accelerate the program, but you may also lose some meaningful results. Since the prefiltering results are always ordered by their significance, the most significant prefiltering results are always aligned first in the alignment module.

- The option `--max-rejected` defines the maximum number of rejected sequences for a query until the calculation of alignments stops. A reject is an alignment whose statistics don't satisfy the search criteria such as coverage threshold, e-value threshold etc. Per default, `--max-rejected` is set to `INT_MAX`, i. e. all alignments until `--max-seqs` alignments are calculated.

Disc space Since the alignment module takes the results of the prefiltering module as input, the size of the prefiltering module output is the point of reference. If alignments are calculated and written for all the prefiltering results, the disc space consumption is 1.75 times higher than the prefiltering output size.

11.3 Clustering module

In the clustering module, only the memory consumption is a critical issue.

Memory consumption The clustering module can need large amounts of memory. The memory consumption for a database containing N sequences and an average of r alignment results per sequence can be estimated as

$$M = 40 \times N \times r\,\mathrm{B}$$

To prevent excessive memory usage for the clustering of large databases, you should use cascaded clustering (`--cascaded` option) which accumulates sequences per cluster incrementally, therefore avoiding excessive memory use.

If you run the clustering module separately, you can tune

- `--max-seqs` parameter which controls the maximum number of alignment results per query considered (i. e. the number of edges per node in the graph). Lower value causes lower memory usage and faster run times.
- Alternatively, `-s` parameter can be set to a higher value in order to cluster the database down to higher sequence identities. Only the alignment results above the sequence identity threshold are imported and it results in lower memory usage.

Runtime Clustering is the fastest step. It needs about 2 hours for the clustering of the whole UniProtKB.

Disc space Since only one record is written per cluster, the memory usage is a small fraction of the memory usage in the prefiltering and alignment modules.

11.4 Workflows

The search and clustering workflows offer the possibility to set the sensitivity option `-s` and the maximum sequences per query option `--max-seqs`. `--max-rejected` option is set to `INT_MAX` per default. Cascaded clustering sets all the options controlling the size of the output, speed and memory consumption, internally adjusting parameters in each cascaded clustering step.

12 Detailed parameter list

12.1 Search workflow

Compares all sequences in the query database with all sequences in the target database.

Usage:

`mmseqs_search <queryDB> <targetDB> <outDB> <tmpDir> [opts]`

Options:

`-s [float] Target sensitivity in the range [1:9] (default=4).`

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. For detailed explanation see section 9.1.

`--z-score [float] Z-score threshold (default:  50.0)`

Prefiltering Z-score cutoff. A lower z-score cutoff yields more results, since also less significant results are written to the output. For detailed explanation see section 9.1.

`--max-seqs Maximum result sequences per query (default=300)`

Maximum number of sequences passing the prefiltering and alignment per query. If the prefiltering result list exceeds the `--max-seqs` value, only the sequences with the best Z-score pass the prefiltering and are aligned in the alignment step.

`--max-seq-len [int] Maximum sequence length (default=50000).`

The length of the longest sequence in the input database.

`--sub-mat [file] Amino acid substitution matrix file (default:  BLOSUM62).`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs `data` folder.

12.2 Clustering workflow

Calculates the clustering of the sequences in the input database.

Usage:

`mmseqs_cluster <sequenceDB> <outDB> <tmpDir> [opts]`

Options:

`--cascaded Start the cascaded instead of simple clustering workflow.`

The database is clustered incrementally in three steps and improves the sensitivity of the clustering greatly compared to the general workflow. For detailed explanation, see the section 8.2.

`-s [float] Target sensitivity in the range [2:9] (default=4).`

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. For detailed explanation see section 9.1.

`--max-seqs Maximum result sequences per query (default=300).`

Maximum number of sequences passing the prefiltering and alignment per query. If the prefiltering result list exceeds the `--max-seqs` value, only the sequences with the best Z-score pass the prefiltering and are aligned in the alignment step.

`--max-seq-len [int] Maximum sequence length (default=50000).`

The length of the longest sequence in the database.

`--sub-mat [file] Amino acid substitution matrix file.`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs `data` folder.

12.3 Updating workflow

Updates the existing clustering of the previous database version with new sequences from the current version of the same database.

Usage:

`mmseqs_update <oldDB> <newDB> <oldDB_clustering> <outDB> <tmpDir> [opts]`

Options:

`--sub-mat [file] Amino acid substitution matrix file.`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs `data` folder.

`--max-seq-len [int] Maximum sequence length (default=50000).`

The length of the longest sequence in the database.

12.4 Prefiltering

Calculates k-mer similarity scores between all sequences in the query database and all sequences in the target database.

Usage:

`mmseqs_pref <queryDB> <targetDB> <outDB> [opts]`

Options:

`-s [float] Sensitivity in the range [1:9] (default=4).`

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. For detailed explanation see section 9.1.

`-k [int] k-mer size in the range [4:7] (default=6).`

The size of k-mers used in the prefiltering. For guidelines for choosing a different k as the default, see section 9.1.

`-cpu [int] Number of cores used for the computation (default=all cores).`

`--alph-size [int] Amino acid alphabet size (default=21).`

Amino acid alphabet size, default = 21 (full amino acid alphabet). For using a reduced amino acid alphabet, choose a lower value. Reduced amino acid alphabets reduce the memory usage, but also the sensitivity.

`--z-score [float] Z-score threshold (default: 50.0).`

Prefiltering Z-score cutoff. A lower z-score cutoff yields more results, since also less significant results are written to the output. For detailed explanation see section 9.1.

`--max-seq-len [int] Maximum sequence length (default=50000).`

The length of the longest sequence in the database.

`--nucl Nucleotide sequences input.`

`--max-seqs [int] Maximum result sequences per query (default=300).`

Maximum number of sequences passing the prefiltering per query. If the prefiltering result list exceeds the `--max-seqs` value, only the sequences with the best Z-score pass the prefiltering.

`--no-comp-bias-corr Switch off local amino acid composition bias correction.`

Compositional bias correction assigns lower scores to amino acid matches of the amino acids that are frequent in their neighborhood in the query sequence.

`--max-chunk-size [int] Splits target databases in chunks when the database size exceeds the given size. (For memory saving only)`

Maximum number of sequences stored in the index table at some point of time, default = `INT_MAX`. Restraining the number of sequences stored reduces the memory usage, but slows down the calculation.

`--skip [int] Number of skipped k-mers during the index table generation.`

Number of k-mers in the database sequences skipped during the index table generation. Per default, each k-mer of the database is indexed. With skip = 2, 2 k-mers are skipped and only each third k-mer is indexed. This speeds up the search and reduces the memory usage at the cost of lower search sensitivity.

`--sub-mat [file] Amino acid substitution matrix file.`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs `data` folder.

`-v [int] Verbosity level: 0=NOTHING, 1=ERROR, 2=WARNING, 3=INFO (default=3).`

Verbosity level in the range $[0:3]$. With verbosity 0, there is no terminal output.

12.5 Alignment

Calculates Smith-Waterman alignment scores between all sequences in the query database and the sequences of the target database which passed the prefiltering.

Usage:

```
mmseqs_pref <queryDB> <targetDB> <prefResultsDB> <outDB> [opts]
```

Options:

`-e [float] Maximum e-value (default=0.01).`

E-value of the local alignment is calculated using Karlin-Altschul statistics.

`-c [float] Minimum alignment coverage (default=0.8).`

Minimum alignment coverage of both query and database sequence, default = 0.8. With the value of 0.0, the alignments are assessed using only the e-value criterion.

`-cpu [int] Number of cores used for the computation (default=all cores).`

`--max-seq-len [int] Maximum sequence length (default=50000).`

The length of the longest sequence in the database.

`--max-seqs [int] Maximum alignment results per query sequence (default=300).`

Maximum number of sequences passing the alignment per query. Sequences are read in the order of the prefiltering lists. The reading for a query is stopped if the number of sequences for a query sequence exceeds the `--max-seqs` value.

`--max-rejected [int] Maximum rejected alignments before alignment calculation for a query is aborted. (default=INT_MAX)`

Maximum number of rejected alignments for a query until the alignment calculation is stopped. A rejected alignment is an alignment that does not satisfy the e-value and alignment coverage thresholds. Default = `INT_MAX` (i. e., all alignments are calculated).

`--nucleotides Nucleotide sequences input.`

`--sub-mat [file] Amino acid substitution matrix file.`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs `data` folder.

`-v [int] Verbosity level: 0=NOTHING, 1=ERROR, 2=WARNING, 3=INFO (default=3).`

Verbosity level in the range [0 : 3]. With verbosity 0, there is no terminal output.

12.6 Clustering

Calculates a clustering of a sequence database based on Smith Waterman alignment scores of the sequence pairs.

Usage:

`mmseqs_clu <sequenceDB> <alnResultsDB> <outDB> [opts]`

Options:

`-g greedy clustering by sequence length (default: set cover clustering algorithm).`

Use a greedy clustering algorithm instead of the set cover algorithm. For the description of the two algorithms, see section 9.3.

`-s [float] Minimum sequence identity of sequences in a cluster (default = 0.0)`

Minimum sequence identity of the cluster members and the representative sequence. Per default, the sequence identity criterion is switched off.

`--max-seqs [int] Maximum result sequences per query (default=100)`

Maximum alignment results read per query. This is at the same time the maximum possible number of sequences in the cluster.

`-v [int] Verbosity level: 0=NOTHING, 1=ERROR, 2=WARNING, 3=INFO (default=3).`

Verbosity level in the range [0 : 3]. With verbosity 0, there is no terminal output.

13 License terms

The software is made available under the terms of the GNU General Public License v3.0. Its contributors assume no responsibility for errors or omissions in the software.