MMseqs2 User Guide

MMseqs2 suite for fast and sensitive batch searching and clustering of huge protein sequence sets

1 Summary

MMseqs2 (Many-against-Many searching) is a software suite to search and cluster huge sequence sets. MMseqs2 is open source GPL-licensed software implemented in C++ for Linux and Mac OS. The software is designed to run on multiple cores and servers and exhibits very good scalability. MMseqs2 reaches the same sensitivity as BLAST magnitude faster and which can also perform profile searches like PSI-BLAST but also 270x faster.

At the core of MMseqs2 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 query database with all sequences a target database based on a very fast and sensitive k-mer matching stage followed by an ungapped alignment. The alignment module implements an vectorized Smith-Waterman-alignment of all sequences that pass a cut-off for the ungapped alignment 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.

MMseqs2 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. MMseqs2 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 are using MMseqs2 to regularly update versions of the UniProtKB database clustered down to 30% sequence similarity threshold. This database is available at uniclust.mmseqs.com.

2 Installation

There are two ways of installing MMseqs: by compiling it or by using a statically pre-compiled binary.

2.1 Static version

The following command will download the latest MMseqs2 release, extract it and set the necessary environment variables.

```
$ wget http://mmseqs.com/latest/mmseqs.tar.gz
```

2.2 Compile

Compiling MMseqs2 from source has the advantage that it will be optimized to the specific system, which might improve its performance. To compile mmseqs `git`, `g++` (4.6 or higher) and `cmake` (3.0 or higher) are needed. Afterwards, the MMseqs2 binary will be located in in `build/bin/`.

```
$ git clone https://github.com/soedinglab/MMseqs2.git
$ cd mmseqs
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. ..
$ make
$ make install
```

MMseqs2 comes with a bash command and parameter auto-completion, which can usually activated by pressing the tab key. The bash completion for subcommands and parameters can be installed by sourcing the `util/bash-completion.sh` file in your `$HOME/.bash_profile`:

```
source path/to/mmseqs/util/bash-completion.sh
```

Homebrew

If you are using Mac OS you can install MMseqs through Homebrew by executing the following:

```
$ brew install https://raw.githubusercontent.com/soedinglab/mmseqs2/master/Formula/mmseqs.rb --HEAD
```

This will also automatically install the bash completion (you might have to do brew install bash-completion first). The formula will also work for Linuxbrew.

3 Getting Started

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

Search

Before searching, you need to convert your FASTA file containing query sequences and target sequences into a sequence DB. You can use the query database `examples/QUERY.fasta` and target database `examples/DB.fasta` to test the search workflow:

```
$ mmseqs createdb examples/QUERY.fasta queryDB
$ mmseqs createdb examples/DB.fasta targetDB
```

These calls should generates five database files each, e.g. `queryDB`, `queryDB_h` and its corresponding index file `queryDB.index`, `queryDB_h.index` and `queryDB.lookup` from the FASTA `QUERY.fasta` input sequences.

The `queryDB` and `queryDB.index` files contain the amino acid sequences, while the `queryDB_h` and `queryDB_h.index` file contain the FASTA headers. The `queryDB.lookup` file contains a list of tab separated fields that map from the internal identifier to the FASTA identifiers.

For the next step, an index file of the the `targetDB` is computed for a fast read in. It is recommend to compute the index if the `targetDB` is reused for several searches.

```
$ mmseqs createindex targetDB
```

This call will create a `targetDB.sk7` file. In this file extension the letter `s` indicates the use of spaced k-mers and the `k7` shows the k-mer size of 7.

Then generate a directory for temporary files. MMseqs2 can produce a high IO on the file system. It is recommend to create this temporary folder on a local drive.

```
$ mkdir tmp
```

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

The alignment consists of two steps the `prefilter` and `alignment`. To run the search, type:

```
$ mmseqs search queryDB targetDB resultDB tmp
```

Search as standart does not compute the score only. If you need the alignment information add the option "-a".

Then, convert the result ffindex database into a BLAST tab formatted file (similar to running blast with the `-m 8` parameter):

```
$ mmseqs convertalis queryDB targetDB resultDB resultDB.m8
```

The file is formatted as a tab-separated lists with 12 columns: (1,2) identifiers for query and target, (3) sequence identity, (4) alignment length, (5) number of mismatches, (6) number of gap openings (7-8, 9-10) domain start and end-position in query and in target, (11) E-value, and (12) bit score.

Clustering

Before clustering, convert your FASTA database into the ffindex format:

```
$ mmseqs createdb examples/DB.fasta DB
```

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 (TODO).

Run the clustering of your database `DB` by executing the following command. MMseqs2 will return the result database files `DB_clu`, `DB_clu.index`:

```
$ mmseqs cluster DB DB_clu tmp
```

To generate a TSV formatted output file from the output file, type:

```
$ mmseqs createtsv DB DB DB_clu DB_clu.tsv
```

You can adjust the sequence identity threshold with `--min-seq-id` and the alignment coverage with `-c`. MMseqs2 will set the sensitivity parameters automatic based on target sequence identity (`--min-seq-id`), if it is not already specified through the `-s` or `--k-score` parameters.

Sequence information can be added by using createseqfiledb and result2flat can produce a result.

```
$ mmseqs createseqfiledb DB DB_clu DB_clu_seq
$ mmseqs result2flat DB DB DB_clu_seq DB_clu_seq.fasta
```

4 System Requirements

MMseqs2 runs on modern UNIX operating systems; it was tested on Linux and OSX. Alignment and prefiltering modules are using with SSE4.1 and OpenMP, i.e. MMseqs2 can take advantage of multicore computers.

MMseqs2 needs uses a lot main memory (see section memory requirements). We offer an option for limiting the memory usage 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 MMseqs2 and how to reduce resource requirements for large databases.

5 Database Format

MMseqs2 works internally with a database format similar to the ffindex databases. The format was developed to avoid drastically slowing down the file system when millions of files need to be written and accessed. ffindex hides the single files from the file system by storing them as unstructured data records in a single huge binary *data file*. In addition to this data file, an ffindex database includes a secondary file: This *index file* stores for each entry as tab separated line with an unique accession code, the start position in bytes of the data record in the ffindex data file, and a record length.

The MMseqs2 modules `createdb` and `createfasta` do the format conversion from fasta to the internal database format. `createdb` generates a ffindex database from a FASTA sequence database. `createfasta` 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 provided several tools (query, build and apply function on each entry) to work with ffindex databases at `http://github.com/soedinglab/ffindex_soedinglab/`. The binary `ffindex_get` can be used to directly access single records stored in an ffindex database.

6 Overview of Folders in MMseqs

- `bin`: `mmseqs`, `blastp.sh`, `blastpgp.sh`, `cascaded_clustering.sh`, `clustering.sh`.
- `data`: BLOSUM matrices and test data.
- `util`: Contains the Bash parameter completion script.

7 Overview of MMseqs2 Commands

MMseqs2 contains three workflows that combine the three core MMseqs2 modules (prefilter, align, and clust) and several other smaller ones.

Workflows:

- `mmseqs search`: Compares all sequences in the query database with all sequences in the target database, using the prefiltering and alignment modules. MMseqs2 search supports sequence/sequence, profile/sequence or sequence/profile searches.
- `mmseqs cluster`: Clusters sequences by similarity. It compares all sequences in the sequence DB with each other using mmseqs search, filters alignments according to user-specified criteria (max. E-value, min. coverage,...), and runs mmseqs clust to group similar sequences together into clusters.
- `mmseqs clusterupdate`: MMseqs2 incrementally updates a clustering, given an existing clustering of a sequence database and a new version of this sequence database (with new sequences being added and others having been deleted).

And the three core modules:

- `mmseqs prefilter`: Computes k-mer similarity scores between all sequences in the query database and all sequences in the target database.
- `mmseqs align`: 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 prefilter` pass a minimum threshold.
- `mmseqs clust`: Computes a similarity clustering of a sequence database based on Smith Waterman alignment scores of the sequence pairs computed by `mmseqs alignment`.

Complete list of all tools

```
Main tools (for non-experts)
createdb        Convert protein sequence set in a FASTA file to MMseqs  sequence DB format
Search        Search with query sequence or profile DB (iteratively) through target sequence
cluster               Compute clustering of a sequence DB (quadratic time)
createindex           Precompute index table of sequence DB for faster searches

Utility tools for format conversions
  createtsv           Create tab-separated flat file from prefilter DB, alignment DB, or
  convertalis         Convert alignment DB to BLAST-tab format, SAM flat file, or to raw
  convertprofiledb    Convert ffindex DB of HMM/HMMER3/PSSM files to MMseqs profile DB
  convert2fasta       Convert sequence DB to FASTA format
  result2flat         Create a FASTA-like flat file from prefilter DB, alignment DB, or cl

Utility tools for clustering
  clusterupdate       Update clustering of old sequence DB to clustering of new sequence D
  createseqfiledb     Create DB of unaligned FASTA files (1 per cluster) from sequence DB
  mergeclusters       Merge multiple cluster DBs into single cluster DB

Core tools (for advanced users)
  prefilter           Search with query sequence / profile DB through target DB (k-mer ma
  align               Compute Smith-Waterman alignments for previous results (e.g. prefilt
  clust               Cluster sequence DB from alignment DB (e.g. created by searching DB
  clustlinear         Cluster sequences of >70% sequence identity *in linear time*
  clusthash           Cluster sequences of same length and >90% sequence identity *in line

Utility tools to manipulate DBs
  extractorfs         Extract open reading frames from all six frames from nucleotide sequ
  translatenucs       Translate nucleotide sequence DB into protein sequence DB
  swapresults         Reformat prefilter/alignment/cluster DB as if target DB had been sea
  mergedbs            Merge multiple DBs into a single DB, based on IDs (names) of entries
  splitdb             Split a mmseqs DB into multiple DBs
  subtractdbs         Generate a DB with entries of first DB not occurring in second DB
  filterdb            Filter a DB by conditioning (regex, numerical, ...) on one of its w
  createsubdb         Create a subset of a DB from a file of IDs of entries
  result2profile      Compute profile and consensus DB from a prefilter, alignment or clus
  result2msa          Generate MSAs for queries by locally aligning their matched targets
  result2stats        Compute statistics for each entry in a sequence, prefilter, alignmen
```

```
Special-purpose utilities
  diffseqdbs              Find IDs of sequences kept, added and removed between two versions o
  concatdbs               Concatenate two DBs, giving new IDs to entries from second input DB
  summarizetabs           Extract annotations from HHblits BAST-tab-formatted results
  gff2db                  Turn a gff3 (generic feature format) file into a gff3 DB
  maskbygff               X out sequence regions in a sequence DB by features in a gff3 file
  prefixid                For each entry in a DB prepend the entry ID to the entry itself
  convertkb               Convert UniProt knowledge flat file into knowledge DB for the select
  summarizeheaders        Return a new summarized header DB from the UniProt headers of a clus
  extractalignedregion      Extract aligned sequence region
  extractdomains          Extract highest scoring alignment region for each sequence from BLAS

Bash completion for tools and parameters can be installed by adding "source␣path/to/mmseqs/u
Include the location of the MMseqs binaries is in your "$PATH" environment variable.
```

8 Description of Workflows

8.1 Batch Sequence Searching using `mmseqs search`

For searching a database, query and target database have to be converted by `createdb` in order to use them in MMseqs. The search can be executed by typing:

```
$ mmseqs search queryDB targetDB outDB tmp
```

MMseqs2 supports iterative searches which are similar to PSI-BLAST. The following program call will run two iterations through the database. In the first iteration sequences are searched against sequence and in the second one profiles are used to search against sequences.

MMseqs2 will use the output for the first iteration sequence-sequence search to computes a profile (result2profile). The profile will be used as input in the next search iteration.

```
$ mmseqs search queryDB targetDB outDB tmp --num-iterations 2
```

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).

MMseqs2 can precompute the prefilter index `createindex` to speed up subsequence prefilter index read-ins. If the parameter `--use-index` is provided to MMseqs2 `search` will try to use this index for the prefilter. We recommend to use an index for iterative searches, if a target database will be reused several times.

The underlying algorithm is explained in more detail in section 9.1, and the full parameter list can be found in section 13.1.

8.2 Clustering Databases using `mmseqs clusteringworkflow`

To cluster a database, MMseqs2 needs a sequence database converted with `createdb` and an empty directory for temporary files. Then, you can run the clustering with:

```
$ mmseqs cluster inDB outDB tmp
```

and cascaded clustering with:

```
$ mmseqs cluster  inDB outDB tmp --cascaded
```

The sensitivity of the clustering can be adjust with the `-s` option. MMseqs2 will automatically adjust the sensitivity based on the `--min-seq-id` parameter, if neither `--cascaded` nor `-s` are provided.

```
$ mmseqs cluster  inDB outDB tmp
```

The clustering workflow combines the prefiltering, alignment and clustering modules into either a simple clustering or a cascaded clustering of a sequence database. There are two ways to execute the clustering:

- The *Simple clustering* runs the prefiltering, alignment and clustering modules with predefined parameters with a single iteration.
- *Cascaded clustering* clusters the sequence database using the prefiltering, alignment and clustering modules incrementally in three steps.

Cascaded Clustering

We introduced an extremely fast redundancy filtering preprocessing step that can cluster sequences of identical length and 100 % overlap. It reduces each sequence to a 5-letter alphabet, computes a 64 bit CRC32 hash value for the full-length sequences, and places sequences with identical hash code that satisfy the sequence identity threshold into the same cluster.

Afterwards we begin with three the cascaded clustering steps: In the first step of the cascaded clustering 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 very 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 executed with a higher sensitivity and a lower result significance threshold for catching sequence pairs with lower sequence identity. In the last step, the whole process is repeated again with the final target sensitivity. At last, 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.

8.3 Updating a Database Clustering using `mmseqs clusterupdate`

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 clusterupdate 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 representatives of new clusters.

9 Description of Core Modules

For advanced users, it is possible to skip the workflows and execute the 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 13.

MMseqs2 contains three core modules: prefiltering, alignment and clustering.

9.1 Computation of Prefiltering Scores using `mmseqs prefilter`

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

If you want to *cluster* a database, or do an all-against-all search, the same database will be used on both the query and target side. the following program call does an all-against-all prefiltering:

```
$ mmseqs prefilter inputDB inputDB resultDB_pref
```

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

For *sequence search* two different input databases are usually used: a query database `queryDB` and a target database `targetDB`, though they can again be identical. In this case, the prefiltering program call is:

```
$ mmseqs prefilter 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.

It is furthermore possible to use different k-mer lengths, which are used in the prefiltering. Longer k-mers are more sensitive, since they cause less chance matches. Though longer k-mers only pay off for larger databases, since more time is needed 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. The default value $k = 6$ is the best for databases containing only a few million sequences. For very large databases containing about 100 million sequences, $k = 7$ should be a better choice.

9.2 Local alignment of prefiltering sequences using `mmseqs alignment`

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, the query and target databases need to be identical:

```
$ mmseqs algin inputDB inputDB resultDB_pref resultDB_aln
```

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

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

```
$ mmseqs align queryDB targetDB resultDB_pref resultDB_aln
```

This module implements a SIMD 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 cluster`

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

```
$ mmseqs cluster inputDB resultsDB_aln resultsDB_clu
```

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

The clustering module offers the possibility to run three different clustering algorithms by altering the `--cluster-mode` parameter. A greedy set cover algorithm is the default (`--cluster-mode 0`). 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 second clustering algorithm is a greedy clustering algorithm (`--cluster-mode 2`), 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.

The third clustering algorithm is the connected component algorithm. This algorithm uses the transitivity of the relations to form larger clusters with more remote homologies. This algorithm adds all proteins to a cluster, that are reachable in a breadth first search starting at the representative with the most connections.

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 MMseqs2 commands are stored in ffindex databases. All records within those ffindex databases are in plain ASCII 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 identifier 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 identifier 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, memory and disc space consumption of MMseqs2 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 are most important 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 (memory consumption, total runtime and 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. 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 $N \times L \times 7$ byte. 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 $a^k \times 8$ byte, 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

$$M \quad = \quad (7\,N\,L + 8\,a^k)\mathrm{B}$$

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 MMseqs2 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 prefiltering list with an average length of 150 and an output file size of 146 GB.

Important Options for Tuning the Memory, Runtime and Disc Space Usage

- The option `-s` controls the sensitivity in the MMseqs2 prefiltering module. The lower the sensitivity, the faster the prefiltering becomes, though at the cost of search sensitivity. The 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 total runtime and disk space are the critical issues.

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 $\approx 20 - 30\%$ pairwise sequence identity).

If a huge amount of alignments have 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

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

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 the following parameters:

- `--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 How to run MMseqs2 on multipe servers using MPI

MMseqs2 can run on multiple cores and servers using OpenMP (OMP) and message passing interface (MPI). MPI assigns database splits to each servers and each server computes them using multiple cores (OMP). Currently `prefilter`, `align`, `result2profile`, `swapresults` can take advantage of MPI. To parallelize the time-consuming k-mer matching and gapless alignment stages prefilter among multiple servers, two different modes are available. In the first, MMseqs2 can split the target sequence set into approximately equal-sized chunks, and each server searches all queries against its chunk. Alternatively, the query sequence set is split into equal-sized chunks and each server searches its query chunk against the entire target set. Splitting the target database is less time-efficient due to the slow, IO-limited merging of results. But it reduces the memory required on each server to $7 \times NL/\#\text{chunks} + 21^k \times 8\,\text{B}$ and allows users to search through huge databases on servers with moderate memory sizes. If the number of chunks is larger than the number of servers, chunks will be distributed among servers and processed sequentially. By default, MMseqs2 automatically decides which mode to pick based on the available memory (assume that all machines have the same amount of memory). Make sure that MMseqs2 was compiled with MPI by using the `HAVE_MPI=1` flag (`cmake -DHAVE_MPI=1 -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`). Our precomplied static version of MMseqs2 can not use MPI. To search with multiple server just call the search and add the RUNNER variable. The TMP folder has to be shared between all nodes (e.g. NFS)

```
RUNNER="mpirun -np 42" mmseqs search queryDB targetDB resultDB tmp
```

For clustering just call the clustering. The TMP folder has to be shared between all nodes (e.g. NFS)

```
RUNNER="mpirun -np 42" mmseqs cluster DB clu tmp
```

13 Detailed Parameter List

13.1 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=32000).`

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 MMseqs2 `data` folder.

13.2 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=32000).`

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 MMseqs2 `data` folder.

13.3 Updating Workflow

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

Usage:

`mmseqs clusterupdate <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 MMseqs2 `data` folder.

`--max-seq-len [int] Maximum sequence length (default=32000).`

The length of the longest sequence in the database.

13.4 Prefiltering

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

Usage:

```
mmseqs prefilter <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 [6:7] (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.

`--k-score [int] Set the K-mer threshold for the K-mer generation.`

`--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.

`--max-seq-len [int] Maximum sequence length (default=32000).`

The length of the longest sequence in the database.

`--profile HMM Profile input.`

`--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 [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.

`--search-mode [int] Search mode. Global: 0 Local: 1 Local fast: 2.`

`--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.

`--fast-mode Fast search is using Z-score instead of logP-Value and extracts hits with a score higher than 6`

`--spaced-kmer-mode Spaced` k`-mer mode (use consecutive pattern). Disable: 0, Enable: 1`

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

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs2 `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.
`--threads [int] Number of cores used for the computation (default=all cores).`

13.5 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 align <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.

`--min-seq-id Minimum sequence identity of sequences`

`--max-seq-len [int] Maximum sequence length (default=32000).`

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).

`--nucl Nucleotide sequences input.`

`--profile HMM Profile input.`

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

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs2 `data` folder.

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

`-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.

13.6 Clustering

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

Usage:

`mmseqs clust <sequenceDB> <alnResultsDB> <outDB> [opts]`

Options:

`--cluster-mode 0 Setcover, 1 connected component, 2 Greedy clustering by sequence length).`

For the description of the three algorithms, see section 9.3.

`--min-seq-id [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.

14 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.