
MMseqs2 User Guide

Martin Steinegger, Milot Mirdita, Eli Levy Karin, Lars von
den Driesch, Clovis Galiez, Johannes Söding



Contents

Summary	5
System Requirements	5
Installation	6
Install MMseqs2 for Linux	7
Install MMseqs2 for macOS	8
Install MMseqs2 for Windows (preview)	9
Use the Docker image	9
Set up the BASH command completion	10
Getting Started	10
Usage of MMseqs2 modules	10
Easy workflows	11
Searching	11
Clustering	13
Linclust	14
Updating a clustered database	14
Overview of Folders in MMseqs2	15
Overview of important MMseqs2 Modules	15
Description of Workflows	16
Batch Sequence Searching using <code>mmseqs search</code>	16
Translated Sequence Searching	18
Mapping Very Similar Sequences using <code>mmseqs map</code>	18
Clustering Databases using <code>mmseqs cluster</code> or <code>mmseqs linclust</code>	19
Linear time clustering using <code>mmseqs linclust</code>	23
Updating a Database Clustering using <code>mmseqs clusterupdate</code>	27
Taxonomy assignment using <code>mmseqs taxonomy</code>	27
Reciprocal Best Hit using <code>mmseqs rbh</code>	33
Description of Core Modules	34
Computation of Prefiltering Scores using <code>mmseqs prefilter</code>	34
Local alignment of prefiltering sequences using <code>mmseqs align</code>	37
Clustering sequence database using <code>mmseqs clust</code>	37

File Formats	38
MMseqs2 database format	38
Manipulating databases	39
Sequence database format	40
Prefiltering format	41
Alignment format	42
Clustering format	44
Profile format	46
Identifier parsing	49
Optimizing Sensitivity and Consumption of Resources	49
Prefiltering module	49
Alignment Module	51
Clustering Module	52
Workflows	53
How to run MMseqs2 on multiple servers using MPI	53
How to run MMseqs2 on multiple servers using batch systems	54
Frequently Asked Questions	55
How to set the right alignment coverage to cluster	55
How does MMseqs2 compute the sequence identity	57
How to restart a search or clustering workflow	58
How to find the best hit the fastest way	58
How does MMseqs2 handle low complexity	58
How to redundancy filter sequences with identical length and 100% length overlap.	59
How to add sequence identities and other alignment information to a clustering result.	59
How to run external tools for each database entry	59
How to compute a multiple alignment for each cluster	60
How to manually cascade cluster	60
How to cluster using profiles	61
How to create a HHblits database	61
How to create a target profile database (from PFAM)	62
How to cluster a graph given as tsv or m8 file	63
How to search small query sets fast	64
What is the difference between the map and search workflow	64
How to build your own MMseqs2 compatible substitution matrices	65
Workflow Control Parameters	65
Search Workflow	65

Clustering Workflow	65
Updating Workflow	66
External Libraries used in MMseqs2	66
License Terms	67

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, Mac OS and Windows. 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 400 times 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 allows 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.

System Requirements

MMseqs2 runs on modern UNIX operating systems and is tested on Linux and macOS. Additionally, we are providing a preview version for Windows.

The alignment and prefiltering modules are using with SSE4.1 (or optionally AVX2) and OpenMP, i.e. MMseqs2 can take advantage of multicore computers.

When searching large databases, MMseqs2 may need 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 disk space (~500 GB). In section [Optimizing Sensitivity and Consumption of Resources](#), we will discuss the runtime, memory and disk space consumption of MMseqs2 and how to reduce resource requirements for large databases.

To check if MMseqs2 supports your system execute the following commands, depending on your operating system:

Check system requirements under Linux

```
[ $(uname -m) = "x86_64" ] && echo "64bit: Yes" || echo "64bit: No"
grep -q sse4_1 /proc/cpuinfo && echo "SSE4.1: Yes" || echo "SSE4.1: No"
grep -q avx2 /proc/cpuinfo && echo "AVX2: Yes" || echo "AVX2: No"
```

Check system requirements under macOS

```
[ $(uname -m) = "x86_64" ] && echo "64bit: Yes" || echo "64bit: No"
sysctl machdep.cpu.features | grep -q SSE4.1 && echo "SSE4.1: Yes" || echo "SSE4.1: No"
sysctl machdep.cpu.leaf7_features | grep -q AVX2 && && echo "AVX2: Yes" || echo "AVX2: No"
```

Check system requirements under Windows

The `mmseqs.bat` script will print a message if it is run on an unsupported system. On a supported system, it will execute the correct MMseqs2 version and forward all parameters.

Installation

MMseqs2 can be installed for Linux, macOS, or Windows by

- (1) downloading a statically compiled version (see below),
- (2) compiling the from source (see below),
- (3) using [Homebrew](#) (for macOS) or [Linuxbrew](#)

```
brew install mmseqs2
```

- (4) using bioconda

```
conda install -c bioconda mmseqs2
```

(5) or using [Docker](#).

```
docker pull soedinglab/mmseqs2
```

Install MMseqs2 for Linux

Install with Linuxbrew

You can use [Linuxbrew](#) for installation:

```
brew install mmseqs2
```

Install static Linux version

The following commands will download the latest MMseqs2 version, extract it and sets the PATH variable.

If your computer supports AVX2 use:

```
wget https://mmseqs.com/latest/mmseqs-linux-avx2.tar.gz
tar xvzf mmseqs-linux-avx2.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

If your computer supports SSE4.1 use:

```
wget https://mmseqs.com/latest/mmseqs-linux-sse41.tar.gz
tar xvzf mmseqs-linux-sse41.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

Compile from source under Linux

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

```
git clone https://github.com/soedinglab/MMseqs2.git
cd MMseqs2
```

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. .
make
make install
export PATH=$(pwd)/bin/:$PATH
```

Install MMseqs2 for macOS

Install with Homebrew

You can install the latest stable version of MMseqs2 for macOS with [Homebrew](#) by executing

```
brew install mmseqs2
```

This will also automatically install the bash completion (you might have to execute `brew install bash-completion` first).

Install static macOS version

Alternatively, you can download and install the newest commit using our statically compiled binaries. (If you do not have `wget` installed, install it using Homebrew `brew install wget`).

If your computer supports AVX2, use:

```
wget https://mmseqs.com/latest/mmseqs-osx-avx2.tar.gz
tar xvzf mmseqs-osx-avx2.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

If your computer supports SSE4.1, use:

```
wget https://mmseqs.com/latest/mmseqs-osx-sse41.tar.gz
tar xvzf mmseqs-osx-sse41.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

Compile from source under macOS

:exclamation: Please install the `gcc@8` `zlib` `bzip2` `vim` `cmake` packages from Homebrew. The default MacOS `clang` compiler does not support OpenMP and MMseqs2 will not be able to run multithreaded. Use the following cmake call:

```
CC="gcc-8" CXX="g++-8" cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. .
```


Install MMseqs2 for Windows (preview)

Install static Windows version

The latest version is always available on:

<https://mmseqs.com/latest/mmseqs-win64.zip>

Download and unzip it at a convenient location. Inside you will find the `mmseqs.bat` wrapper script, which should be used to substitute all calls to `mmseqs` in the remainder of this document, and a `bin` folder with all dependencies of the MMseqs2 Windows version. Please keep the `mmseqs.bat` script always one folder above the `bin` folder, or it will not be able to correctly identify its dependencies anymore.

The windows build also contains both the SSE4.1 and the AVX2 version. The `mmseqs.bat` script will automatically choose the correct one.

Compile from source under Windows

The windows build process is more involved due to MMseqs2's dependency on an installed shell. We use the Cygwin environment and Busybox to provide all necessary dependencies and bundle them all together. If you want to compile MMseqs2 on your own, install the following packages from Cygwin:

```
bash xxd cmake make gcc-g++ zlib-devel libbz2-devel busybox-standalone binutils
```

Afterwards, use a workflow similar to the `util/build_windows.sh` script to build MMseqs2 on Windows.

Use the Docker image

You can pull the official docker image by running:

```
docker pull soedinglab/mmseqs2
```

If you want to build the docker image from the git repository, execute:

```
git clone https://github.com/soedinglab/MMseqs2.git
cd MMseqs2
docker build -t mmseqs2 .
```

Set up the BASH command completion

MMseqs2 comes with a bash command and parameter auto completion by pressing tab. The bash completion for subcommands and parameters can be installed by adding the following lines to your `$HOME/.bash_profile`:

```
if [ -f /Path to MMseqs2/util/bash-completion.sh ]; then
    source /Path to MMseqs2/util/bash-completion.sh
fi
```

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.

Usage of MMseqs2 modules

Calls to MMseqs2 modules and workflows have the structure:

```
mmseqs module input_db output_db args [options]
```

module stands for one of many modules or workflows such as **search**, **easy-search**, **createdb**, etc. **args** stands for other obligatory paths/filenames the module needs, and **options** can be optionally given to change parameter settings or modify the behavior of the module in some way. Workflows and modules are the same from the user perspective. Internally, workflows are shell scripts chaining elementary MMseqs2 module and other workflows together. A list of all available modules can be printed with `mmseqs -h`.

You can get information about any MMseqs2 module by simply calling it without arguments and options, e.g.

```
mmseqs createdb
```

It will give a short description, the usage, and a list of the most important options. The usage explains the syntax of the module. For example `mmseqs createdb` will give you its usage as:

```
Usage: <i:fastaFile1[.gz]> ... <i:fastaFileN[.gz]> <o:sequenceDB> [options]
```

<...> denote files, <i:...> and <o:...> denote input and output files, respectively, brackets [] indicate that what is enclosed is optional, whereas everything not enclosed in [] cannot be omitted.

An extended list of parameters can be shown by adding the `-h` flag to the MMseqs2 call:

```
mmseqs createdb -h
```

Easy workflows

We provide **easy** workflows to search and cluster. The **easy-search** searches directly with a FASTA/FASTQ file against a either another FASTA/FASTQ file or an already existing MMseqs2 target database.

```
mmseqs easy-search examples/QUERY.fasta examples/DB.fasta alnResult.m8 tmp
```

For clustering, MMseqs2 **easy-cluster** and **easy-linclust** are available.

easy-cluster by default clusters the entries of a FASTA/FASTQ file using a cascaded clustering algorithm.

```
mmseqs easy-cluster examples/DB.fasta clusterRes tmp
```

easy-linclust clusters the entries of a FASTA/FASTQ file. The runtime scales linearly with input size. This mode is recommended for huge datasets.

```
mmseqs easy-linclust examples/DB.fasta clusterRes tmp
```

These **easy** workflows are a shorthand to deal directly with FASTA/FASTQ files as input and output. MMseqs2 provides many modules to transform, filter, execute external programs and search. However, these modules use the MMseqs2 database formats, instead of the FASTA/FASTQ format. For optimal efficiency, we recommend to use MMseqs2 workflows and modules directly.

Searching

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 generate five 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.

Important:

`createdb` splits long sequences into multiple separate entries automatically. This avoids excessive resource requirements for later steps. The default value is to split sequences after 65535 residues. The identifiers of the new entries are suffixed with `_0` to `_(n-1)` for `N` splits.

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

```
mmseqs createindex targetDB tmp
```

This call will create a `targetDB.idx` file. It is just possible to have one index per database.

Then generate a directory for temporary files. MMseqs2 can produce a high IO on the file system. It is recommended 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 disk space requirements, see the section [Disk Space](#).

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

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

Search as standard does compute the score only. If you need the alignment information add the option `-a`.

Then, convert the result database into a BLAST tab formatted file (option `-m 8` in legacy blast, `-outfmt 6` in blast+):

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

The file is formatted as a tab-separated list with 12 columns: (1,2) identifiers for query and target sequences/profiles, (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.

Read more about searching [here](#).

Clustering

Before clustering, convert your FASTA database into the MMseqs2 database (DB) format:

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

You will need to specify a directory for temporary files. Please ensure that in case of large input databases `tmp` provides enough free space. For the disk space requirements, see the section [Disk space](#).

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` and `--cov-mode`. 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
```

Read more about clustering [here](#).

Linclust

Linclust is a clustering in linear time. It is magnitudes faster but a bit less sensitive than [clustering](#).

Before clustering, convert your FASTA database into the MMseqs2 database (DB) format:

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

To run linclust to generate a clustering of your database DB execute the following command. The result database follows the same format as the [clustering format](#):

```
mmseqs linclust DB DB_clu tmp
```

To extract the representative sequences from the clustering result call:

```
mmseqs result2repseq DB DB_clu DB_clu_rep
mmseqs result2flat DB DB DB_clu_rep DB_clu_rep.fasta --use-fasta-header
```

Updating a clustered database

It is possible to update previous clustered databases without re-clustering everything from the scratch.

Let us create an older version of the DB.fasta (in the example folder) by removing 1000 sequences:

```
awk '/^>/{seqCount++;} {if (seqCount <= 19000) {print $0;}}' DB.fasta > DB_trimmed.fasta
```

Now we create the sequence database of this simulated old sequence and the corresponding clustering:

```
mmseqs createdb DB_trimmed.fasta DB_trimmed
mmseqs cluster DB_trimmed DB_trimmed_clu tmp
```

To update the clustering DB_trimmed_clu with the new version of your database DB_new:

```
mmseqs createdb DB.fasta DB_new
mmseqs clusterupdate DB_trimmed DB_new DB_trimmed_clu DB_new_updated DB_update_clu tmp
```

`DB_update_clu` contains now the freshly updated clustering of `DB_new`. Furthermore, the `clusterupdate` creates a new sequence database `DB_new_updated` that has consistent identifiers with the previous version. Meaning, the same sequences in both sets will have the same numeric identifier. All modules afterwards (for example `convertalis`) expect this sequence database to be passed.

Read more about the cluster updating [here](#).

Overview of Folders in MMseqs2

- `bin`: `mmseqs`
- `data`: BLOSUM matrices and the workflow scripts (`blastp.sh`, `blastpgp.sh`, `cascaded_clustering.sh`, `linclust.sh`, `searchtargetprofile.sh`, `clustering.sh`)
- `examples`: test data `QUERY.fasta` and `DB.fasta`
- `util`: Contains the Bash parameter completion script.

Overview of important MMseqs2 Modules

MMseqs2 contains five workflows that combine the core MMseqs2 modules (`prefilter`, `align`, `kmermatcher`, `rescorediagonal` 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 linclust`: Clusters sequences by similarity in linear time. It clusters magnitudes faster than `mmseqs cluster` but is less sensitive.
- `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).
- `mmseqs taxonomy`: Taxonomy assignment by computing the lowest common ancestor of homologs using 2bLCA.

- **mmseqs map**: Map calls the search workflow with different default parameters and alignment strategy. Map should be used for high identity searches, which is more of a technical task rather than measuring homology.

And the five core modules:

- **mmseqs prefilter**: Computes k-mer similarity scores between all sequences in the query database and all sequences in the target database.
- **mmseqs kmermatcher**: finds exact k-mer matches between all input sequences in linear time.
- **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 rescorediagonal**: Computes 1D optimal score between all sequences in the query database and the sequences of the target database, which passed the prefilter **mmseqs prefilter** or **mmseqs kmermatcher**.
- **mmseqs clust**: Computes a similarity clustering of a sequence database based on Smith Waterman alignment scores of the sequence pairs computed by **mmseqs align**.

MMseqs2 has more than 96 modules in total. We provide modules for clustering, searching, alignments, taxonomy, and data transformation. For a complete list of all available modules, execute **mmseqs** without arguments.

Description of Workflows

MMseqs2 workflows combine modules in shell scripts. The executed script can be found in the respective temporary directory.

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 MMseqs2. 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.

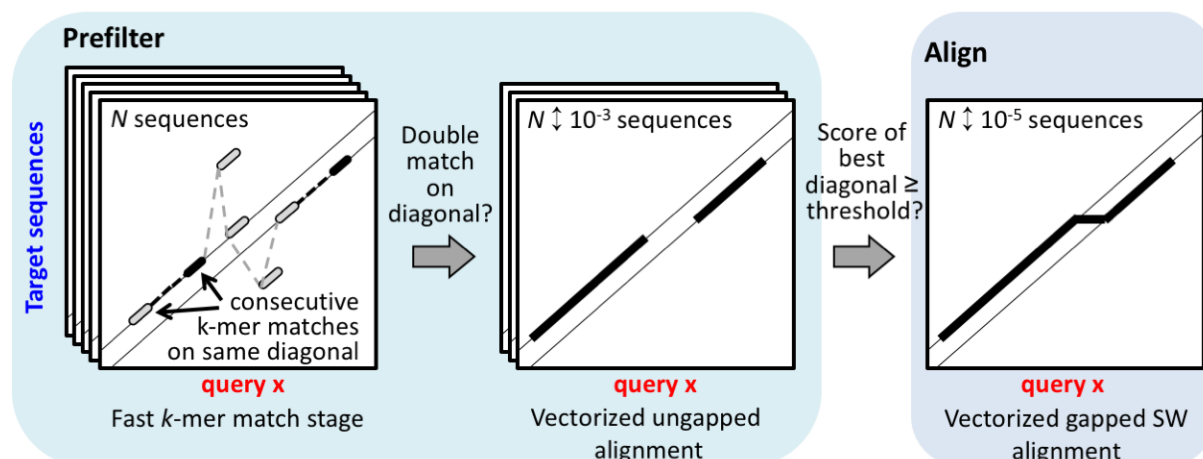


Figure 0.1: Search workflow

MMseqs2 will use the output for the first iteration sequence-sequence search to compute a profile (result2profile). The profile will be used as input in the next search iteration. Accepted hits are aligned only once and removed from later iterations.

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

This workflow combines the prefiltering and alignment modules into a fast and sensitive batch 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. We recommend to use an index for iterative searches or if a target database will be reused several times. However reading the index can be a bottleneck when using a network file systems (NFS). It is recommended to keep the index on a local hard drive. If storing the index file on a local hard drive is not possible and the NFS is a bottleneck then there are two solutions. (1) Load the index into main memory using the module `touchdb` or `vmtouch` (<https://github.com/hoytech/vmtouch>), MMseqs2 can be forced to use the main memory database by using the parameter `--db-load-mode 3`. (2) Do not create an index, MMseqs2 will compute an index on the fly this reduces the IO volume by roughly a factor of seven.

More details to the underlying search algorithm can be found here: [Computation of Prefiltering Scores using mmseqs prefilter](#), and the important parameter list can be found in section [Search Workflow](#).

Translated Sequence Searching

The search workflow can perform translated searches with nucleotide databases on either query or target, or both sides. It will trigger a search similar to BLASTX, TBLASTN and TBLASTX respectively. The search uses the **extractorfs** module to detect all open reading frames (ORFs) on all six frames and translates them into proteins. Per default, MMseqs2 extracts all ORFs per frame starting from any codon until a stop codon occurs (**--orf-start-mode 1**) that are longer than 30 amino acid residues (**--min-length 30**). All ORFs are translated by the **translatenucs** module using the canonical translation table (**--translation-table 1**). After the protein-protein search all alignments will be offset to the original nucleotide coordinates by the **offsetalignment** module.

To perform a translated search create your database using **createdb**. It can automatically detect if the input are amino acids or nucleotides.

```
mmseqs createdb ecoli.fna ecoli_genome
mmseqs createdb ecoli.faa ecoli_proteins
```

A nucleotide/protein (BLASTX) search can be triggered using the nucleotide database on the query database side.

```
mmseqs search ecoli_genome ecoli_proteins alnDB tmp
```

A protein/nucleotide (TBLASTN) search can be triggered using the nucleotide database on the target database side.

```
mmseqs search ecoli_proteins ecoli_genome alnDB tmp
```

A translated nucleotide/nucleotide (TBLASTX) search can be triggered using the flag **--search-type 2**

```
mmseqs search genome_orfs_aa ecoli_genome alnDB tmp --search-type 2
```

Mapping Very Similar Sequences using **mmseqs map**

The **map** workflow of MMseqs2 finds very similar sequence matches in a sequence database. First it calls the **prefilter** module (with a low sensitivity setting) to detect high scoring diagonals and then computes an ungapped alignment with the **rescorediagonal** module. In contrast to the normal search, for maximum speed no gapped alignment is computed, query sequences are not masked for low complexity regions and no compositional bias correction is applied.

```
mmseqs map queryDB targetDB resultDB tmp
```

MMseqs2 will provide a sorted (by E-value) list of best matches in **resultDB**. The best hit can be extracted with:

```
mmseqs filterdb resultDB bestResultDB --extract-lines 1
```

The format of **resultDB** is the same as in **alignment format** of the normal **search** workflow. The mapping workflow can also be used in **iterative-best-hit mode**, where each query that does not find any match is searched with higher sensitivity again.

If either **queryDB** or **targetDB** is a nucleotide sequence database, MMseqs2 will use the **translated sequence search mode** described above.

Clustering Databases using **mmseqs cluster** or **mmseqs linclust**

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

```
mmseqs cluster inDB outDB tmp
```

The sensitivity of the clustering can be adjusted with the **-s** option. MMseqs2 will automatically adjust the sensitivity based on the **--min-seq-id** parameter, if **-s** is not provided.

Linclust can be used by calling **linclust**. The sensitivity can be adjusted by **--kmer-per-seq** (default 20).

```
mmseqs linclust inDB outDB tmp
```

The clustering workflow **cluster** 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* **--single-step-clustering** runs the hashclust and prefiltering, alignment and clustering modules with predefined parameters with a single iteration.
- *Cascaded clustering* (default) clusters the sequence database using the as first step **linclust** and then prefiltering, alignment and clustering modules incrementally in three steps.

Clustering criteria

MMseqs2/Linclust and Linclust has three main criteria, inferred by an local alignment, to link two sequences by an edge:

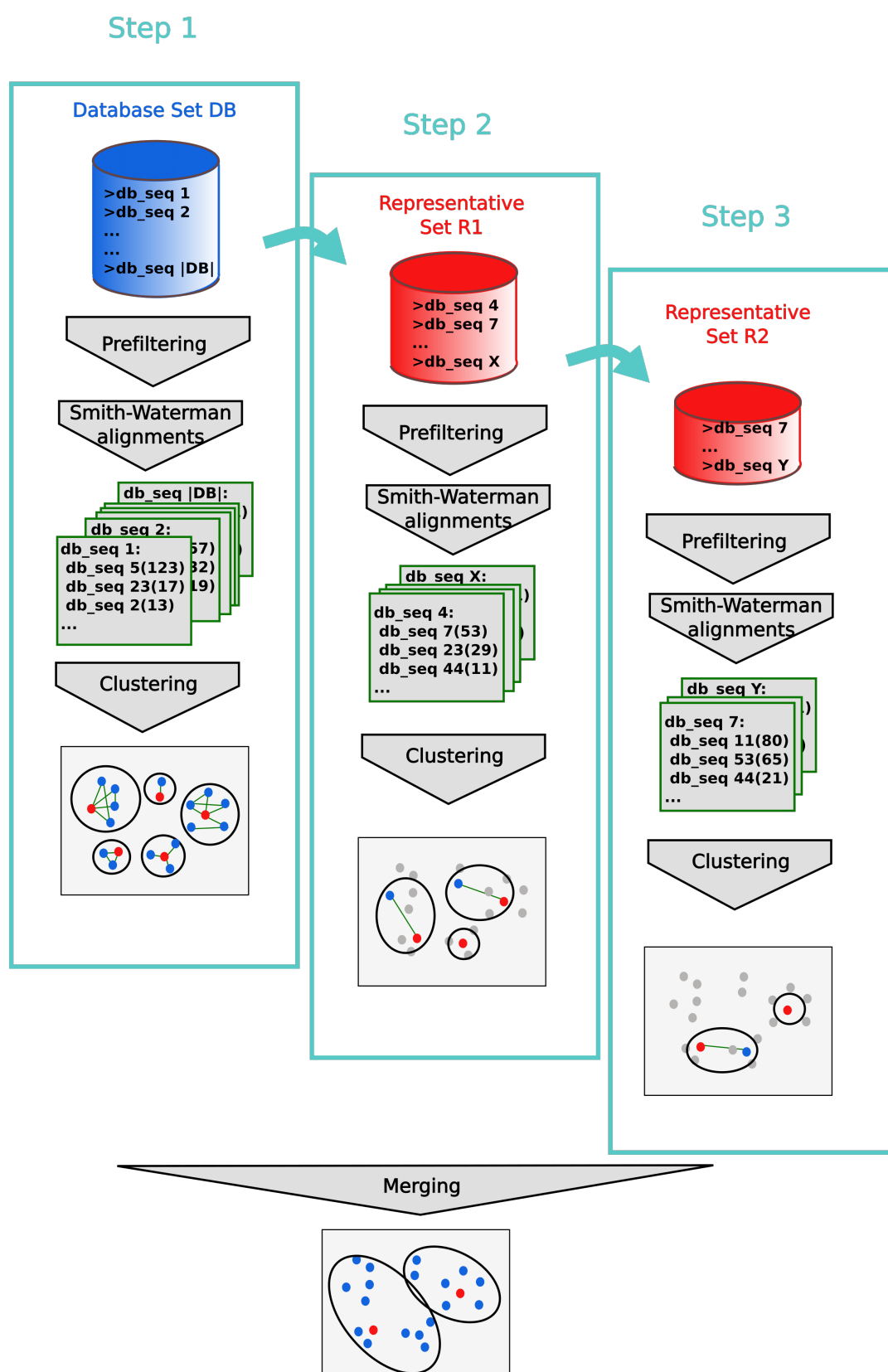
- (1) a maximum E-value threshold (option `-e [0,\infty[`) computed according to the gap-corrected Karlin-Altschul statistics using the ALP library;
- (2) a minimum coverage (option `-c [0,1]`, which is defined by the number of aligned residue pairs divided by either the maximum of the length of query/centre and target/non-centre sequences $\text{alnRes}/\max(\text{qLen}, \text{tLen})$ (default mode, `--cov-mode 0`), or by the length of the target/non-centre sequence $\text{alnRes}/\text{tLen}$ (`--cov-mode 1`), or by the length of the query/centre $\text{alnRes}/\text{qLen}$ (`--cov-mode 2`);
- (3) a minimum sequence identity (`--min-seq-id [0,1]`) with option `--alignment-mode 3` defined as the number of identical aligned residues divided by the number of aligned columns including internal gap columns, or, by default, defined by a highly correlated measure, the equivalent similarity score of the local alignment (including gap penalties) divided by the maximum of the lengths of the two locally aligned sequence segments. The score per residue equivalent to a certain sequence identity is obtained by a linear regression using thousands of local alignments as training set.

Cascaded clustering

The cascaded clustering workflow first runs `linclust`, our linear-time clustering module, that can produce clustering's down to 50% sequence identity in very short time.

To achieve lower sequence identities and/or to further improve the resulting clusters, we continue with three cascaded clustering steps: In the first step of the cascaded clustering the prefiltering runs with a low sensitivity of 1 and a very high result 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 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

**Figure 0.2:** Cascaded clustering

and the alignment. Therefore, we strongly recommend to use cascaded clustering especially to cluster larger databases and to obtain maximum sensitivity.

Clustering modes

All clustering modes transform the alignment results into an undirected graph. In this graph notation, each vertex (i.e. node) represents a sequence, which is connected to other sequences by edges. An edge between a pair of sequences is introduced if the alignment criteria (e.g. `--min-seq-id`, `-c` and `-e`) are fulfilled.

The Greedy Set cover (`--cluster-mode 0`) algorithm is an approximation for the NP-complete optimization problem called set cover.

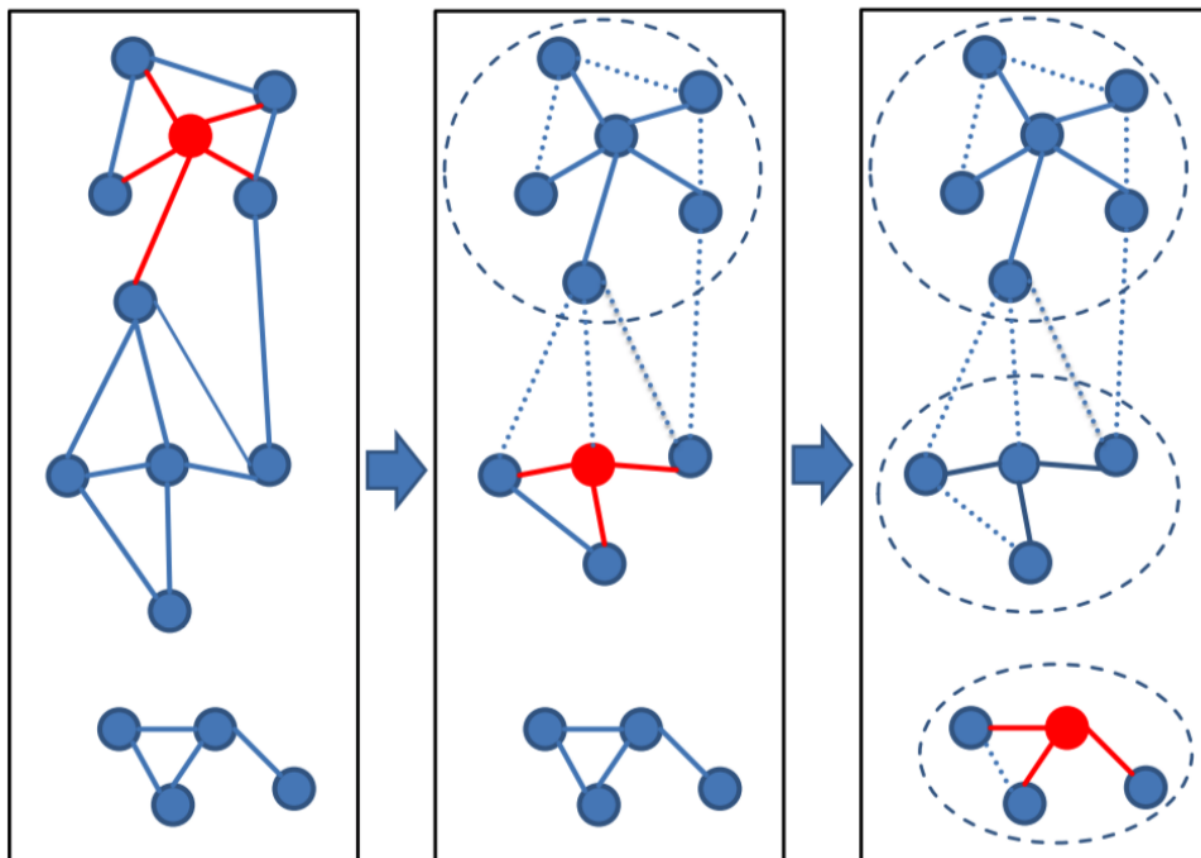


Figure 0.3: Set Cover clustering

Greedy set cover works by iteratively selecting the node with most connections and all its connected nodes to form a cluster and repeating until all nodes are in a cluster. The greedy set cover is followed by a reassignment step. A Cluster member is assigned to another cluster centroid if their alignment score was higher.

Connected component (`--cluster-mode 1`) uses transitive connection to cover more remote homologs.

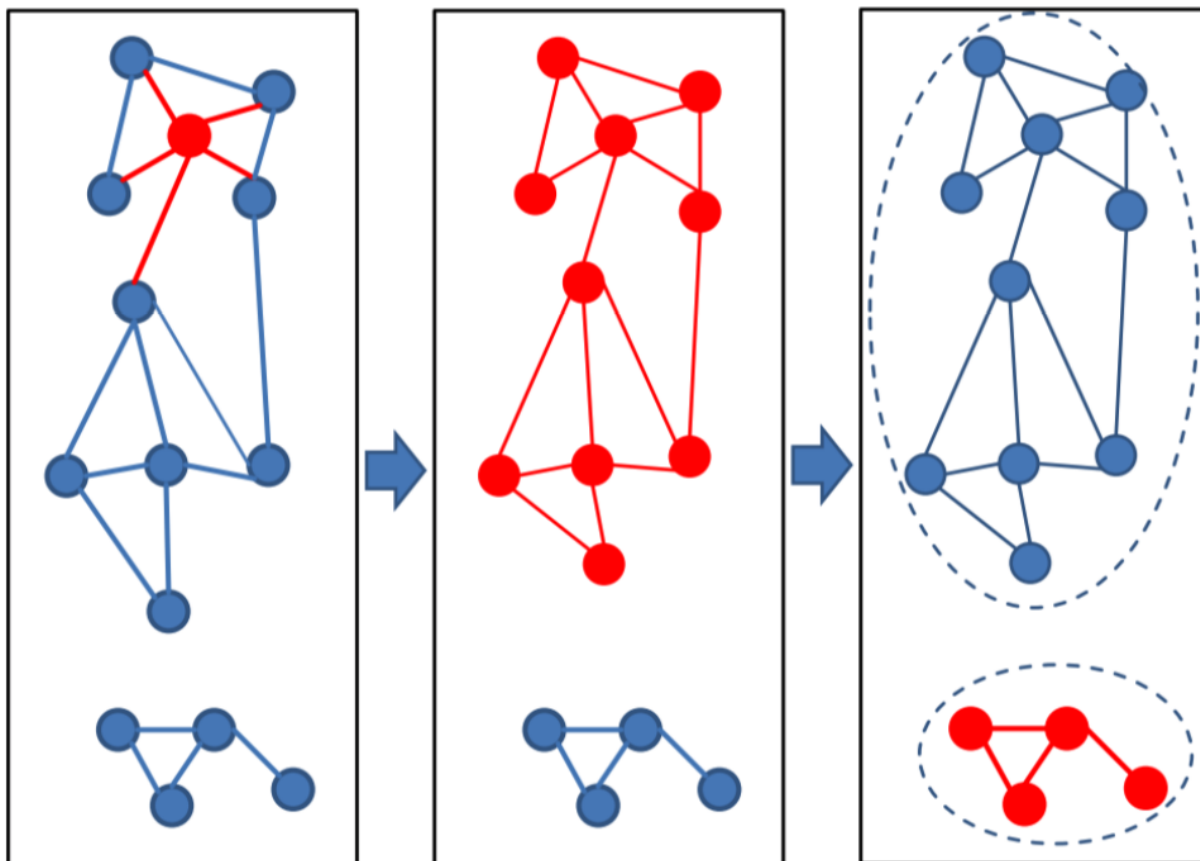


Figure 0.4: Connected component clustering

In connected component clustering starting at the mostly connected vertex, all vertices that are reachable in a breadth-first search are members of the cluster.

Greedy incremental (`--cluster-mode 2`) works analogous to CD-HIT clustering algorithm.

Greedy incremental clustering takes the longest sequence (indicated by the size of the node) and puts all connected sequences in that cluster, then repeatedly the longest sequence of the remaining set forms the next cluster.

Linear time clustering using `mmseqs linclust`

Linclust can cluster sequences down to 50% pairwise sequence similarity and its runtime scales linearly with the input set size.

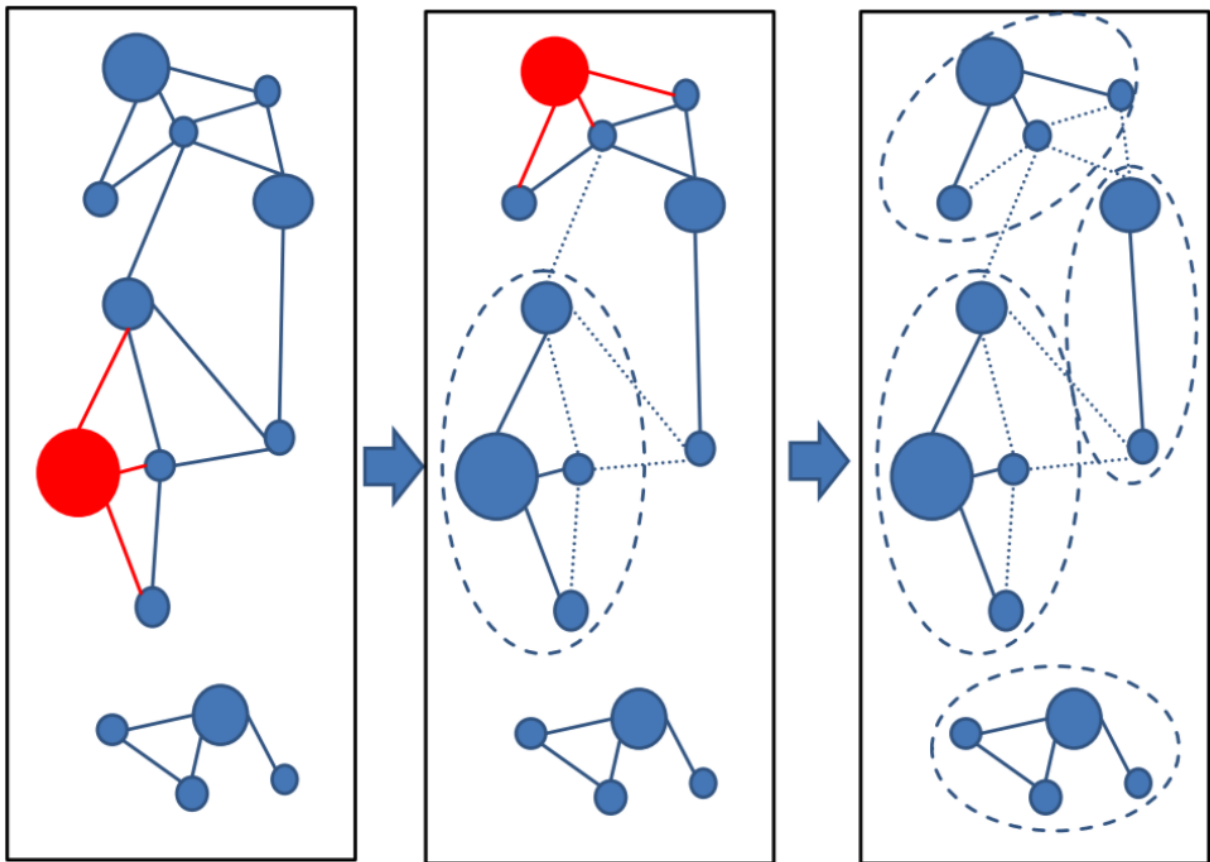


Figure 0.5: Greedy incremental clustering

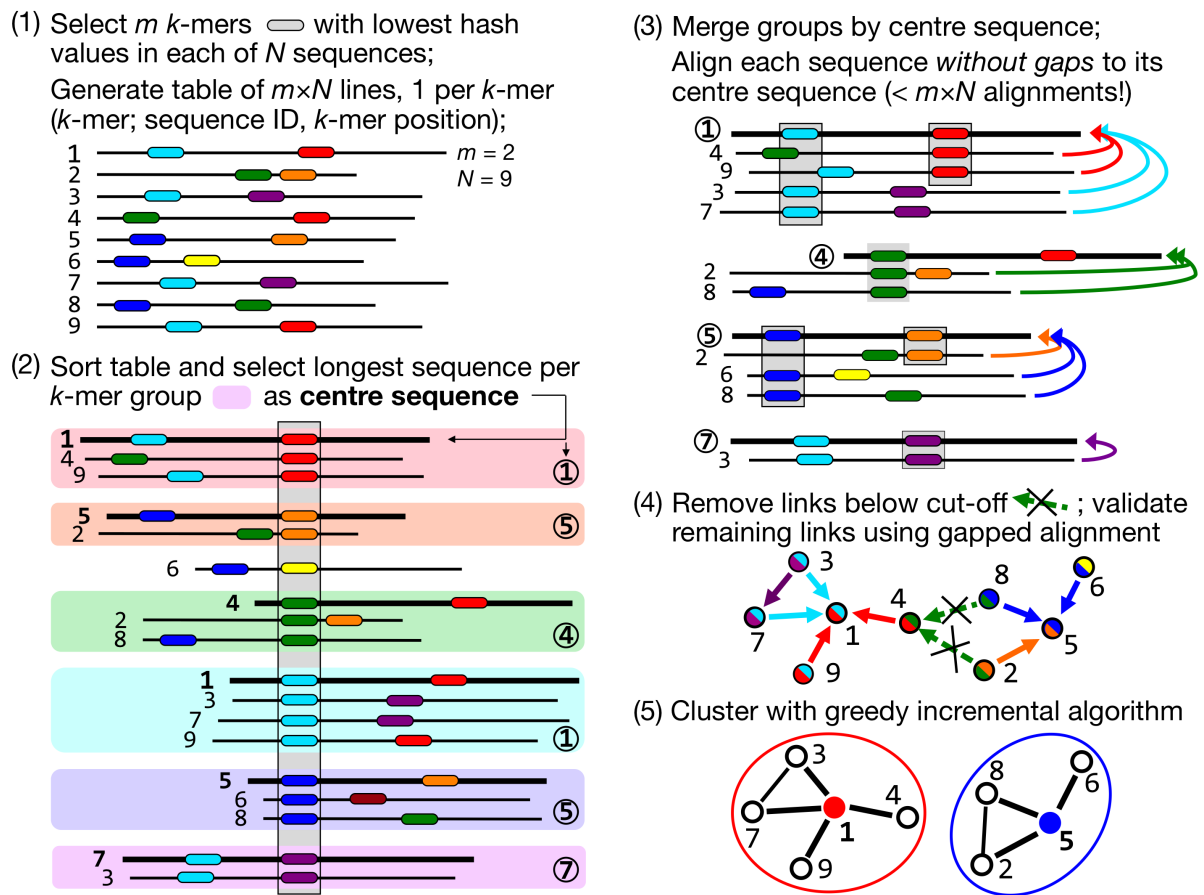


Figure 0.6: Linclust algorithm

Linear-time clustering algorithm. Steps 1 and 2 find exact k -mer matches between the N input sequences that are extended in step 3 and 4.

- (1) Linclust selects in each sequence the m (default: 20) k -mers with the lowest hash function values, as this tends to select the same k -mers across homologous sequences. It uses a reduced alphabet of 13 letters for the k -mers and sets $k=10$ for sequence identity thresholds below 90% and $k=14$ above. It generates a table in which each of the mN lines consists of the k -mer, the sequence identifier, and the position of the k -mer in the sequence.
- (2) Linclust sorts the table by k -mer in quasi-linear time, which identifies groups of sequences sharing the same k -mer (large shaded boxes). For each k -mer group, it selects the longest sequence as centre. It thereby tends to select the same sequences as centre among groups sharing sequences.
- (3) It merges k -mer groups with the same centre sequence together: red + cyan and orange + blue and compares each group member to the centre sequence in two steps: by global Hamming distance and by gapless local alignment extending the k -mer match.
- (4) Sequences above a score cut-off in step 3 are aligned to their centre sequence using gapped local sequence alignment. Sequence pairs that satisfy the clustering criteria (e.g. on the E-value, sequence similarity, and sequence coverage) are linked by an edge.
- (5) The greedy incremental algorithm finds a clustering such that each input sequence has an edge to its cluster's representative sequence. Note that the number of sequence pairs compared in steps 3 and 4 is less than mN , resulting in a linear time complexity.

Run Linclust

Linclust needs a sequence database created by `createdb` and an empty directory for temporary files. Then, you can run the clustering with the following command:

```
mmseqs linclust inDB outDB tmp
```

Increasing the k -mers selected per sequence increases the sensitivity of linclust at a moderate loss of speed. Use the parameter `--kmer-per-seq` to set the number of k -mers selected per sequence. More k -mers per sequences results in a higher sensitivity.

The output format of linclust is the same format as in `mmseqs cluster`. See section [Clustering Format](#).

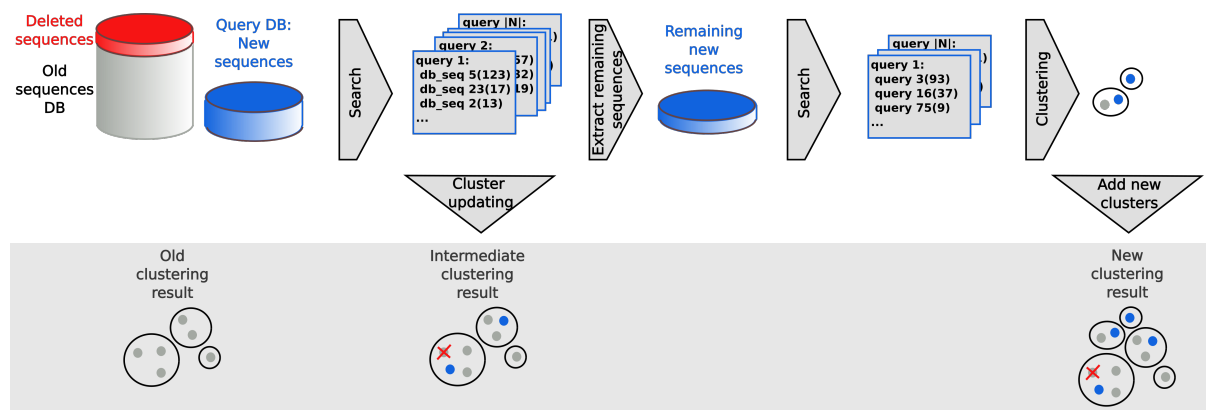


Figure 0.7: Update clustering

Updating a Database Clustering using `mmseqs clusterupdate`

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

```
mmseqs clusterupdate oldDB newDB cluDB_old newDB_updated cluDB_updated 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 corresponding clustering, and the new version of the sequence database. Then it adds the new sequences to the clustering and removes the sequences that were removed in the new database. Sequences which are not similar enough to any existing cluster will be representatives of new clusters.

`clusterupdate` creates a new sequence database `newDB_updated` that has consistent identifiers with the previous sequence databases. Meaning, the same sequences in both sets will have the same numeric identifier. All modules afterwards (for example `convertalis`) expect this sequence database to be passed.

Taxonomy assignment using `mmseqs taxonomy`

By identifying homologs through searches with taxonomy annotated reference databases, MMseqs2 can compute the lowest common ancestor. This lowest common ancestor is a robust taxonomic label for unknown sequences.

MMseqs2 implements the 2bLCA protocol (Hingamp et. al., 2013) with `--lca-mode 2` (default) for choosing a robust LCA.

Figure 0.8: 2bLCA protocol (Hingamp et. al, 2013)

The second search can be disabled with `--lca-mode 1`. The LCA will then be only computed through the usual search workflow parameters (`--max-accept`, `-e`, etc.).

The LCA implementation is based on the Go implementation of [blast2lca](#) software on GitHub. It implements the LCA computation efficiently through *Range Minimum Queries* through an dynamic programming approach.

Prerequisites

MMseqs2 needs the NCBI taxonomy information (merged.dmp, names.dmp, nodes.dmp) and a mapping from taxTargetDB sequences to the taxonomic identifier (targetDB_mapping). The `createtaxdb` module will download the Uniprot taxMappingFile and ncbi-taxdump and map the identifier of the `targetDB` to NCBI taxonomic identifier.

```
mmseqs createdb target.fasta targetDB
mmseqs createtaxdb targetDB tmp
```

By default, `createtaxdb` downloads the Uniprot id mapping file, and thus only support Uniprot identifiers. To create a taxonomy database from an existing NCBI BLAST database (such as `nr` or `nt`), see the section [Create a sequence database with taxonomic information from an existing BLAST database](#). For other database types, the mapping must be created manually, as described in section [Manually annotate a sequence database with taxonomic information](#).

Classification

Once the prerequisites are generated, the taxonomy classification can be executed:

```
mmseqs taxonomy queryDB targetDB taxonomyResult tmp
mmseqs createtsv queryDB taxonomyResult taxonomyResult.tsv
```

Each line of the result file `taxonomyResult.tsv` will contain a tab separated list of 1) query accession, 2) LCA NCBI taxon ID, 3) LCA rank name, and 4) LCA scientific name.

The `--lca-ranks` parameter can be supplied with a colon (:) separated string of taxonomic ranks. For example, `--lca-ranks genus:family:order:superkingdom` will resolve the respective ranks of the LCA and return a colon concatenated string of taxa as the fifth column of the result file.

The taxonomy result can be summarized in a Kraken-style report using the `taxonomyreport` module.

```
mmseqs taxonomyreport targetDB taxonomyResult taxonomyResult_report
```

The report shows a taxon tree with read counts and mapped fractions.

5.6829	362	362	no rank	0	unclassified
94.3171	6008	43	no rank	1	root
87.8493	5596	126	no rank	131567	cellular organisms
42.5903	2713	79	superkingdom	2759	Eukaryota
32.8257	2091	38	no rank	33154	Opisthokonta
24.0502	1532	2	kingdom	33208	Metazoa
23.8776	1521	3	no rank	6072	Eumetazoa
23.2810	1483	49	no rank	33213	Bilateria
14.2857	910	2	no rank	33511	Deuterostomia
13.9560	889	3	phylum	7711	Chordata
13.3124	848	0	subphylum	89593	Craniata

The column are (1) the percent of mapped reads, (2) count of mapped reads, (3) rank, (4) taxonomy identifier, and (5) taxonomical name.

This report can be visualised using the interactive metagenomics data explorer [Pavian](#):

We implemented modules to work with taxonomical databases e.g. `filtertaxdb` can be used to extract taxas, `addtaxonomy` adds the taxonomy information to a result database.

E.g. it is possible to extract all sequences from a specific part of the taxonomic tree. Here is an example that extract all sequence that are part of the viral branch of taxonomical tree

```
mmseqs filtertaxdb targetDB taxonomyResult taxonomyResult.virus --taxon-list 10239
awk '$3 != 1 {print}' taxonomyResult.virus > taxonomyResult.virus.id
mmseqs createsubdb taxonomyResult.virus.id queryDB queryDB.virus
mmseqs createsubdb taxonomyResult.virus.id queryDB_h queryDB.virus_h
```

Create a sequence database with taxonomic information from an existing BLAST database

It is easy to create MMSeqs2 search databases with taxonomic information from pre-existing local BLAST databases, if BLAST+ is installed. The following example creates a MMSeqs2 database from NCBI's nt database, but it also works with any of the other BLAST databases including the nr protein database.

First, manually download the NCBI taxonomy database dump:

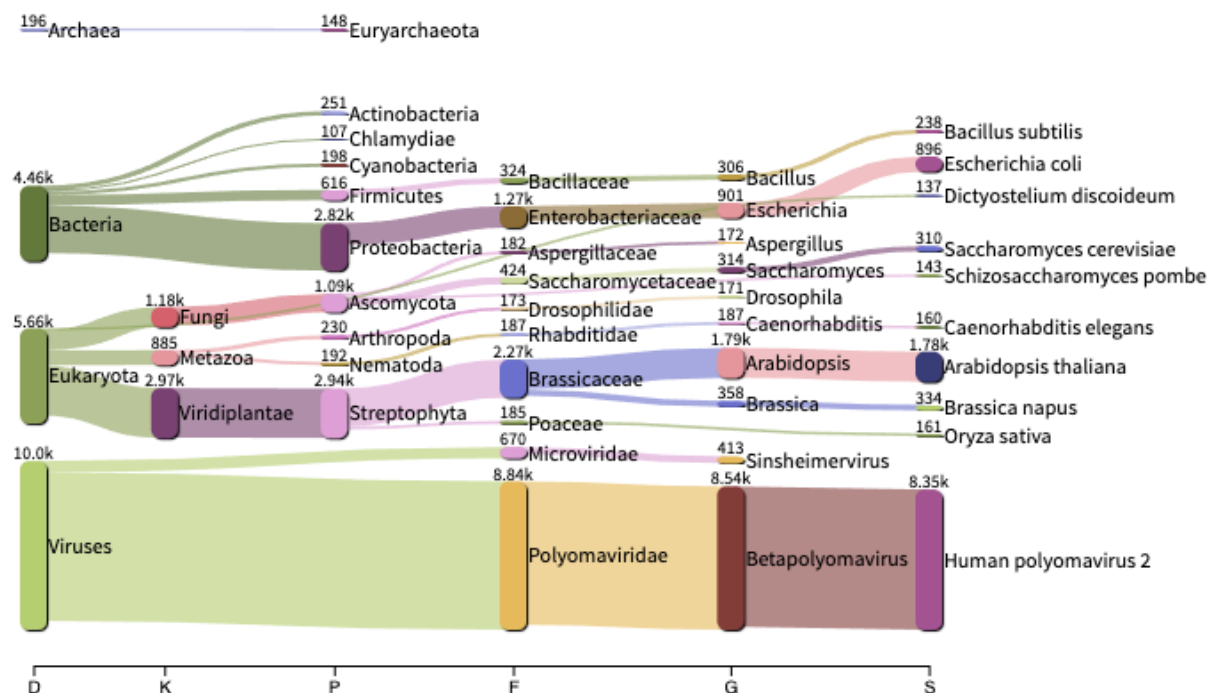


Figure 0.9: Pavian Screenshot

```
wget ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
mkdir taxonomy && tar -xxvf taxdump.tar.gz -C taxonomy
```

BLAST+'s `blastdbcmd` can be used to extract both the FASTA as well as the taxonomy mapping files:

```
blastdbcmd -db nt -entry all > nt.fna
blastdbcmd -db nt -entry all -outfmt "%a %T" > nt.fna.taxidmapping
```

Finally, the `createdb` and `createtaxdb` modules use the information to create a complete MMSeqs2 database:

```
mmseqs createdb nt.fna nt.fnaDB
mmseqs createtaxdb nt.fnaDB tmp --ncbi-tax-dump taxonomy/ --tax-mapping-file nt.fna.taxidmap
```

Manually annotate a sequence database with taxonomic information

Here is an example how to manually annotate a sequence database with taxonomic information. The example uses Uniprot identifiers.

As a first step turn the FAST[A/Q] file into a mmseqs sequence database using `createdb`

```
# Turn the sequences into a MMseqs2 database (this also creates sequenceDB.lookup)
# Skip this step if you already created a database
mmseqs createdb sequence.fasta sequenceDB
```

`createdb` produces a tab-separated `sequenceDB.lookup` file that contains numeric-db-id, Accession (e.g. Uniprot Accession Q6GZX4) and File. IDs are parsed from the header from the input database (see [id parsing from headers](#)).

```
0 Q6GZX4 0
1 Q6GZX3 0
2 Q197F8 0
3 POA031 0
4 Q197F7 0
```

As next step, we create a tab-separated mapping with every target database identifier mapped to a NCBI taxon identifier. The mapping file should be in the format `Accession numeric-ncbi-tax-id`.

```
Q6GZX4    654924
Q6GZX3    654924
Q197F8    345201
Q197F7    345201
```

Here is an example how to transform an Uniprot mapping file into the tab-separated mapping file.

```
# The taxidmapping file should be in the format
# Accession numeric-ncbi-tax-id
# Q6GZX4    654924
# Q6GZX3    654924
# Q197F8    345201
# Q197F7    345201
```

```
# e.g. download the uniprot mapping file and convert it to the taxidmapping mapping format
URL="ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.txt"
wget -nv -O - "$URL" | zcat | awk '$2 == "NCBI_TaxID" {print $1"\t"$3 }' > taxidmapping
```

We need the NCBI taxonomy [taxdump.tar.gz](#). It is available on the NCBI FTP server:


```
mkdir ncbi-taxdump && cd ncbi-taxdump
wget ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz
tar xzvf taxdump.tar.gz
cd -
```

As a final step we can now use `createtaxdb` to annotate our sequence database.

```
# now we can use createtaxdb with our own mapping.
mmseqs createtaxdb sequenceDB tmp --ncbi-tax-dump ncbi-taxdump --tax-mapping-file taxidmapping
```

Extract sequences of a specific taxon from database

It is possible to extract sequences or results based on taxonomy from databases. Here is an example how to extract sequences:

```
# create a taxDB from the taxonomy mapping file
mmseqs tsv2db sequenceDB_mapping sequenceDB_mapping.taxdb --output-dbtype 8
# extract human
mmseqs filtertaxdb sequenceDB sequenceDB_mapping.taxdb sequenceDB_mapping.human.taxdb --
awk '$3 != 1{print}' sequenceDB_mapping.human.taxdb.index > sequenceDB_mapping.human.tax
mmseqs createsubdb sequenceDB_mapping.human.taxdb.ids sequenceDB humanDB
mmseqs createsubdb sequenceDB_mapping.human.taxdb.ids sequenceDB_h humanDB_h
mmseqs convert2fasta humanDB humanDB.fasta
```

Reciprocal Best Hit using `mmseqs rbh`

Given two protein sets “A” and “B”, MMseqs2 can search in two directions and report pairs (“a”, “b”), such that “b” is the highest scoring hit of “a” and “a” is the highest scoring hit of “b”:

```
mmseqs createdb Aproteins.fasta Adb
mmseqs createdb Bproteins.fasta Bdb
mmseqs rbh Adb Bdb ABrbh tmp
```

To obtain the results in m8 format:

```
mmseqs convertalis Adb Bdb ABrbh ABrbh.m8
```

Behind the scenes

The workflow searches with each “a” against the targets in “B”. It retains for each “a” the maximal bitscore it got with any of its hits (denoted “x”). It then searches with each “b” against targets in A and retains its best scoring hit(s). If among these, “a” is found with the score “x”, the pair (“a”, “b”) will be reported as an RBH pair. Importantly, no correction for composition bias nor low complexity masking is performed during the searches (`--comp-bias-corr 0` and `--mask 0`) to assure the bitscores remain the same irrespective of the search direction. Finally, please note that E-values are not used to determine pairing but they are used as a cutoff for the searches (`rbh` uses the default value of `-e` and this can be changed, if needed).

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 [Detailed Parameter List](#).

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

Computation of Prefiltering Scores using `mmseqs prefilter`

The prefiltering module computes an ungapped alignment score for all consecutive k-mer matches between all query sequences and all database sequences and returns the highest score per sequence.

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 sequenceDB sequenceDB resultDB_pref
```

`sequenceDB` is the base name of the MMseqs2 databases produced from the FASTA sequence databases by `mmseqs createdb`, the prefiltering results are stored in the MMseqs2 database files `resultDB_pref` and `prefilterDB.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
```

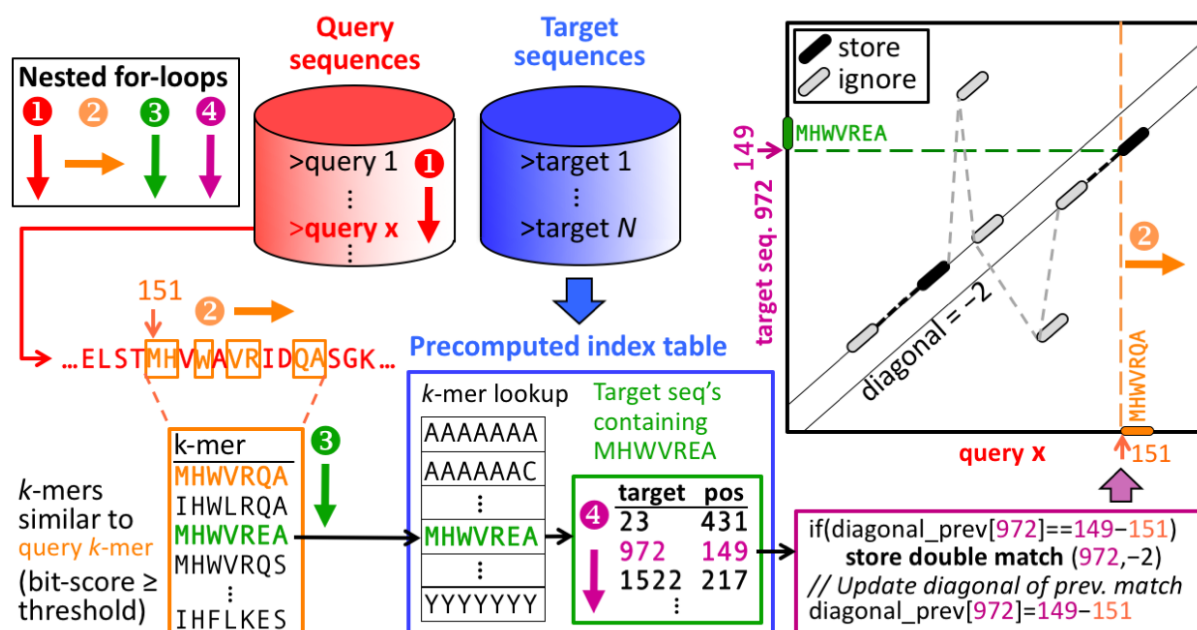


Figure 0.10: Prefilter

MMseqs2 can handle profiles or protein/nucleotide sequences as input for the queryDB.

The prefilter k-mer match stage is key to the high speed and sensitivity. It detects consecutive short words (“k-mer”) match on the same diagonal. The diagonal of a k-mer match is the difference between the positions of the two similar “k”-mer in the query and in the target sequence.

The pre-computed index table for the target database (blue frame) contains for each possible “k”-mer the list of the target sequences and positions where the k-mer occurs (green frame).

Query sequences/profiles are processed one by one (loop 1). For each overlapping, spaced query k-mer (loop 2), a list of all similar k-mer is generated (orange frame). The similarity threshold determines the list length and sets the trade-off between speed and sensitivity. The similar k-mer list length can be controlled with `-s`.

For each similar k-mer (loop 3) we look up the list of sequences and positions where it occurs (green frame). In loop 4 we detect consecutive double matches on the same diagonals (magenta and black frames).

For each consecutive k-mer matches an ungapped alignment is computed. Only the maximal ungapped alignment score for each target is reported.

Set sensitivity `-s` parameter

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.

- *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.

It is possible to speed best hits searches by stepwise increasing `-s`. MMseqs2 includes a workflow for this purpose. [How to find the best hit the fastest way](#)

The following graphic shows the average AUC sensitivity versus speed-up factor relative to BLAST for 637,000 test searches. White numbers in plot symbols give number of search iterations.

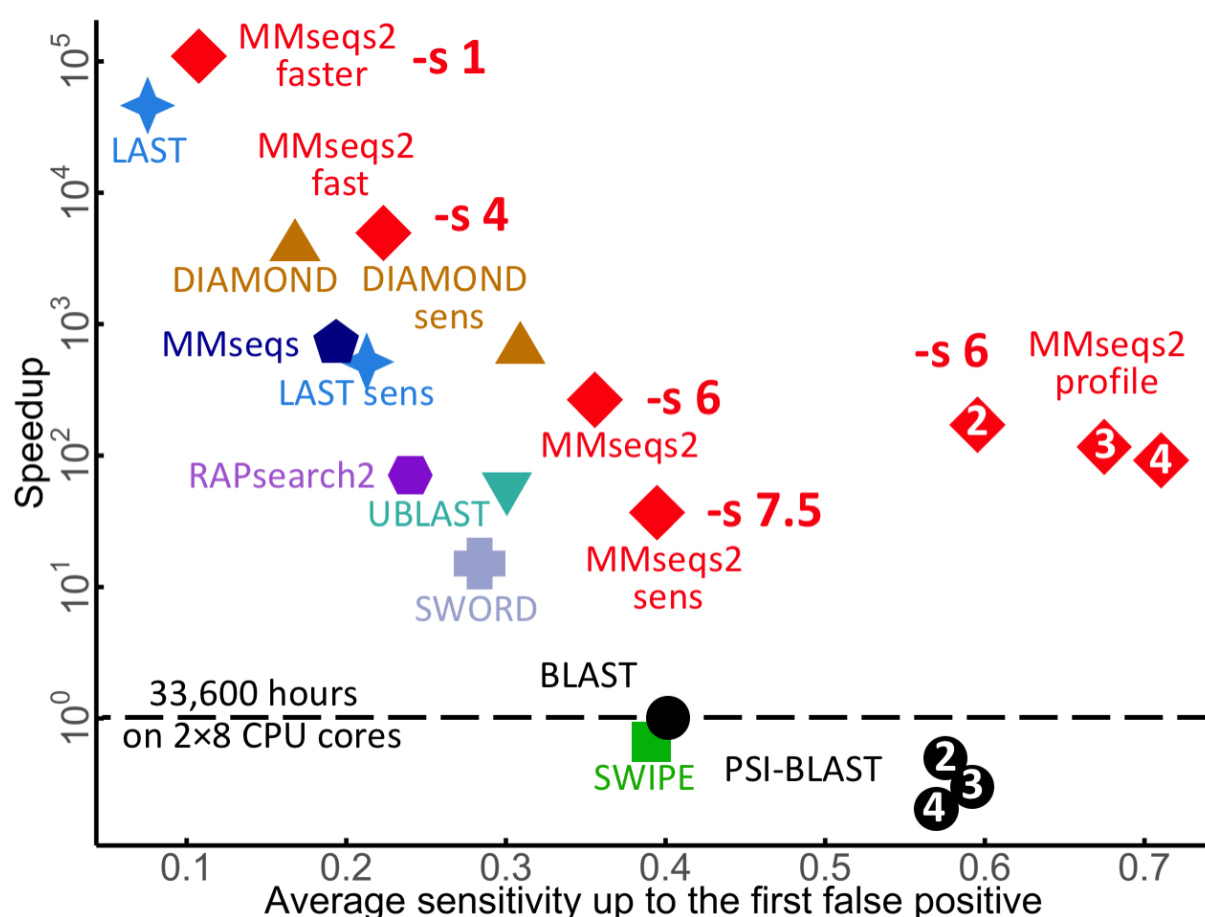


Figure 0.11: Prefilter sensitivity vs. speed

It is furthermore possible to use change the 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. As default MMseqs2 tries to compute the optimal k-mer length based on the target database size.

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, the query and target databases need to be identical:

```
mmseqs align sequenceDB sequenceDB resultDB_pref resultDB_aln
```

Alignment results are stored in the database 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.

Clustering sequence database using `mmseqs clust`

For calling the stand-alone clustering, you need the input sequence database and a result database:

```
mmseqs cluster sequenceDB resultsDB_aln resultsDB_clu
```

Clustering results are stored in the MMseqs 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 sequences to a cluster, that are reachable in a breadth first search starting at the representative with the most connections.

File Formats

MMseqs2 database format

Most MMseqs2 modules consume and produce files in the MMseqs2 database format. The format is inspired by `ffindex` (https://github.com/soedinglab/ffindex_soedinglab), which was developed by Andreas Hauser. It avoids drastically slowing down the file system when millions of files would need to be written or accessed, e.g. one file per query sequence in a many-to-many sequence search. MMseqs2 databases hide these files from the file system by storing them in a single data file. The *data file* `<name>` contains the data records, i.e. the contents of the file, concatenated and separated by `\0` characters. A second, *index file* `<name>.index` contains for each numerical identifies (corresponding to the file name) the position of the corresponding data record in the data file. The `<name>.dbtype` contains the database type e.g. Protein, Nucleotide, ...

Each line of the *index file* contains, separated by tabs, (1) the ID, (2) the offset in bytes of the `data_record` counted from the start of the data file, and (3) the size of the data record. The IDs have to be sorted numerically in ascending order, since for accessing a data record by IDs the matching IDs are found by binary search.

Here is an example for a database containing four sequences:

```
PSSLDIRL
\OGTLKRLSAHYTPAW
\OAEAFIHEG
\OYTHGAGFDNDI
\0
```

The corresponding index file (file extension `.index`) could look like this.

```
10 0 9
11 9 15
12 24 10
13 34 12
```

The index contains four IDs, one for each data record: 10, 11, 12 and 13. The corresponding data records have offset positions 0, 9, 25, 35 and the data record sizes are 9, 15, 10, and 12 respectively.

Databases can contain optional a `dbtype` (file extension `.dbtype`). For sequence database there are three db types amino acid, nucleotide and profile. The `dbtype` just contains a number in binary format. In case the `.dbtype` is missing it is possible to create a `.dbtype` file with the following commands.

```
# Amino acid sequence database
awk 'BEGIN { printf(\"%c%c%c%c\",0,0,0,0); exit; }' > seqDb.dbtype
# Nucleotide sequence database
awk 'BEGIN { printf(\"%c%c%c%c\",1,0,0,0); exit; }' > seqDb.dbtype
# Profile database
awk 'BEGIN { printf(\"%c%c%c%c\",2,0,0,0); exit; }' > seqDb.dbtype
# Generic database e.g for header databases (extension '_h')
awk 'BEGIN { printf(\"%c%c%c%c\",12,0,0,0); exit; }' > seqDb.dbtype
```

The MMseqs2 modules `createdb` converts from FASTA/FASTQ[.gz|.bz] to the MMseqs2 database format. `createdb` generates an MMseqs2 database from a FASTA sequence database. It assigns each sequence in the file a numerical identifier and shuffles the database. Sequences that are longer than `--max-seq-len` (default 65535) letters are split. MMseqs2 sequence database can be converted back to a fasta database by `convert2fasta`.

However, for fast access in very large databases it is advisable to use the MMseqs2 database directly without converting it to FASTA format.

Manipulating databases

The data file of the databases can not be altered easily since any change would break the offset in the `.index` file. MMseqs2 module create new databases rather than changing existing ones.

We have a heap of modules to manipulate MMseqs2 database, such as `createsubdb`, `filterdb`, `concatdbs`, `mergedbs` and `apply`.

Altering the content of the `.index` file is possible. You can, for example, create a subset of the index. This mechanism could be used to create a database with only sequence longer than 100 residues.

```
mmseqs createdb seqDb.fas seqDb
# here we select member that are greater 100 (also count the newline and null bytes)
awk '$3 > 102 {print $1}' seqDb.index > ids.gt100
mmseqs createsubdb ids.gt100 seqDb seqDb.gt100
mmseqs createsubdb ids.gt100 seqDb_h seqDb.gt100_h
```

Sequence database format

The sequence database consists of two databases the sequence data and the header. `createdb` takes an FASTA/FASTQ[.gz|.bz] as input and creates two six files. Each line in the sequence data file is a separate sequence followed by an null byte `\0`. E.g. the following data file has four sequences.

```
MPNGASLCVRFRAHGDAPFFSRD
\0MEVVERHAMFGGEMLTFTTHLSASCGFAM
\0MSSQTVSEVVNEYVGPADGVCAVVGAT
\0MAVALELISQH
\0
```

The corresponding index file (file extension `.index`) looks like this. The first column is the numeric sequence identifier, second the offset and third the length. The length contains the null byte and the new line. So the real sequence sequence is two shorted (`$3 - 2`).

0	0	25
1	25	30
2	55	30
3	85	13

The header database (file ending `_h`) contains the data of `>` entries in fasta and `@` entries in fastq files.


```
tr|Q0KJ32|Q0KJ32_9ACTO Aspartate semialdehyde dehydrogenase OS=Streptomyces albulus GN=asd P
\Otr|FOYHT8|FOYHT8_9STRA Putative uncharacterized protein OS=Aureococcus anophagefferens GN=
\Otr|COXU54|COXU54_9CORY Aspartate-semialdehyde dehydrogenase OS=Corynebacterium lipophilofl
\Otr|D6KVP9|D6KVP9_SCAIO Aspartate-semialdehyde dehydrogenase OS=Scardovia inopinata F0304 C
```

The header index has also four entries. The identifier in the first column correspond to identifier in the index of the data file.

0	0	102
1	102	118
2	220	123
3	343	118

Sequence database can be converted back to fasta only with `convert2fasta`

```
mmseqs convert2fasta seqDb seqDb.fasta
```

Prefiltering format

Each data record consists of the prefilter results for one query sequence. The ID is the database accession code, a numerical identifier (ID) for the query that was assigned by `createdb`.

Each line in a data record reports on one matched database sequence and has the following format (white space between columns contains one tab character):

```
targetID  ungappedScore  diagonal
```

where `targetID` is the database identifier of the matched sequence, the ungapped score of the match, and `diagonal` is the diagonal $i-j$ (i = position in query, j = position in db sequence) on which the match occurs.

Example of a database record for prefiltering:

0	71	0
2	35	0
3	15	8

The first line describes a match with database sequence 2 on diagonal 0 with a $-\log(\text{e-value})$ of 71 (e-value 1.46e-31).

Prefilter database can be converted to TSV by `createtsv`.

```
mmseqs createtsv queryDB targetDB prefRes prefRes.tsv
```

Each line of The TSV represents a single entry in the prefilter result. The first column contains then the query, target identifier, ungapped score and diagonal respectively.

```
Q0KJ32 Q0KJ32 71 0
Q0KJ32 COW539 35 0
Q0KJ32 D6KVP9 15 8
```

Alignment format

Each data record consists of the alignment results for one query sequence. The ID of the queries was assigned by `createdb`.

Each line in a data record reports on match, i.e., one database sequence aligned to the query. It has the following format (white space between columns contains one tab character):

```
targetID alnScore seqIdentity eVal qStart qEnd qLen tStart tEnd tLen [alnCigar]
```

Here, **targetID** is the database identifier of the matched sequence, **alnScore** is the bit score of the alignment in half bits, **seqIdentity** is the sequence identity [0:1], **eVal** is the e-value of the match, **qStart** is the start position of the alignment in the query, **qEnd** is the end position of the alignment in the query, **tStart** and **tEnd** are the start and end positions in the target (i.e. the database sequence), **tLen** is the target sequence length, the optional **alnCigar** string encodes the alignment in compressed format and is only included in the results if the option `-a` was used in MMseqs2 search. The numbers preceding the three letters M, I, and D give the number of match positions in a block aligned without gaps, the number of insertions and of deletions, respectively.

Example data record for alignment results:

```
0 734 1.000 2.084E-243 0 378 379 0 378 379 379M
2 260 0.455 1.305E-79 26 368 379 21 363 369 173M2D41M2D65M6I21M2D37M
3 233 0.434 2.830E-70 25 364 379 30 367 373 162M2I16M3I10M1I5M6D16M2D67M6I25M2D27M
```

The first line with **targetID** 2 is an identity match. The last sequence 3 has a Smith-Waterman alignment score of 347, the sequence identity 0.565 and the e-value 2.722e-99, the query start and end position is 13,367 of the total length 373, the target start and end position is 20,367 of the total length 373, the alignment string is 10M5I53M3I118M1D166M.

The alignment result can be converted into a flat file by `createtsv` or `convertalis`.

```
Q0KJ32 Q0KJ32 783 1.000 7.540E-260 0 418 419 0 418 419 419M
Q0KJ32 COW539 260 0.455 1.305E-79 26 368 379 21 363 369 173M2D41M2D65M6I21M2D37M
Q0KJ32 D6KVP9 233 0.434 2.830E-70 25 364 379 30 367 373 162M2I16M3I10M1I5M6D16M2D67M
```

Custom alignment format with convertalis

An alignment result database can be converted into human readable format with the `convertalis` module.

```
mmseqs convertalis queryDB targetDB alnRes alnRes.tab
```

By default (`--format-mode 0`), `alnRes.tab` will contain alignment result in a BLAST tabular result (comparable to `-m 8 -outfmt 6`) with 12 columns: (1,2) identifiers for query and target sequences/profiles, (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.

The option `--format-output` defines a custom output format. For example, the format string `--format-output "query,target,evalue,qaln,taln"` prints the query and target identifiers, e-value of the alignment and the alignments.

The following fields are supported

- **query** Query sequence identifier
- **target** Target sequence identifier
- **evalue** E-value
- **gapopen** Number of gap opens
- **pident** Percentage of identical matches
- **nident** Number of identical matches
- **qstart** 1-indexed alignment start position in query sequence
- **qend** 1-indexed alignment end position in query sequence
- **qlen** Query sequence length
- **tstart** 1-indexed alignment start position in target sequence
- **tend** 1-indexed alignment end position in target sequence
- **tlen** Target sequence length
- **alnlen** Alignment length (number of aligned columns)
- **raw** Raw alignment score
- **bits** Bit score
- **cigar** Alignment as string. Each position contains either M (match), D (deletion, gap in query), or I (Insertion, gap in target)

- **qseq** Query sequence
- **tseq** Target sequence
- **qaln** Aligned query sequence with gaps
- **taln** Aligned target sequence with gaps
- **qheader** Header of Query sequence
- **theader** Header of Target sequence
- **qframe** Query frame (-3 to +3)
- **tframe** Target frame (-3 to +3)
- **mismatch** Number of mismatches
- **qcov** Fraction of query sequence covered by alignment
- **tcov** Fraction of target sequence covered by alignment
- **empty** Dash column “-”

We support the SAM format using `--format-mode 1`, the AS tag contains the raw score, NM is the miss match count.

Clustering format

Internal cluster format

Each data record consists of the IDs of the members of one cluster. The ID refers to the representative sequence of that cluster, (usually assigned by `createdb`).

Each line in a data record contains one ID of a cluster member. The first line of each data record contains the ID of the representative sequence of that cluster.

Here is an example of two clusters, the first has 3 cluster members (0,2,3) the second 2 cluster member (5,1)

```
0
2
3
\05
1
```

In the first cluster the 0 is the ID of the representative sequence while 2 and 3 are cluster members. In the second cluster 5 is the representative sequence and 1 is a member

Cluster TSV format

The internal format can be converted to a flat tsv file:

```
mmseqs createtsv sequenceDB sequenceDB resultsDB_clu resultsDB_clu.tsv
```

The `resultsDB_clu.tsv` file follows the following format:

```
#cluster-representative    cluster-member
Q0KJ32  Q0KJ32
Q0KJ32  C0W539
Q0KJ32  D6KVP9
E3HQM9  E3HQM9
E3HQM9  F0YHT8
```

All members of the clustering are listed line by line. Each cluster is a consecutive block in the file. The first column always contains the representative sequence, the second contains the cluster member. For the example the cluster with the representative sequence `Q0KJ32` contains four members it self and `C0W539`, `D6KVP9`, `D1Y890`. IDs are parsed from the header from the input database (see [id parsing from headers](#)).

Cluster FASTA-like format

The internal format can be converted to a fasta a like format:

```
mmseqs createseqfiledb DB clu clu_seq
mmseqs result2flat DB DB clu_seq clu_seq.fasta
```

The resulting FASTA-like format file will look like this:

```
>Q0KJ32
>Q0KJ32
MAGA...R
>C0W539
MVGA...R
>D6KVP9
MVGA...R
>D1Y890
MVG...R
>E3HQM9
>E3HQM9
MCAT...Q
>Q223C0
MCAR...Q
```

A new cluster is marked by two identical name lines of the representative sequence, where the first line stands for the cluster and the second is the name line of the first cluster sequence. It is followed by the fasta formatted sequences of all its members.

Extract representative sequence

To extract the representative of a clustering use the following commands:

```
mmseqs result2repseq DB clu clu_rep
mmseqs result2flat DB DB clu_rep clu_rep.fasta --use-fasta-header
```

The resulting fasta will contain all representative sequences:

```
>QOKJ32
MAGA...R
>E3HQM9
MCAT...Q
```

Profile format

The MMseqs2 internal profile format contains 23 values stored per position. The first 20 values are the linear probabilities without pseudo counts in the order ACDEFGHIKLMNPQRSTVWY. We compress the floats using the minifloat implementation with 5 mantissa and 3 exponent bits. Value 21, 22, 23 contains the query residue, consensus residue and the Neff value respectively.

Profiles can be transformed into a Blast like PSSM format with the following command.

```
mmseqs profile2pssm profileDB pssmFile
```

The output format is another database with a human readable representation

Pos	Cns	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
0	M	-2	-1	-1	-2	-2	-2	-2	-1	-2	-2	10	-1	-1	-2	-2	-1	-1	-2	-2	-1
1	A	8	-1	-1	-2	-2	-1	-2	-1	-2	-2	-2	-1	-1	-2	-2	-2	-1	-2	-1	-1
2	A	8	-1	-1	-2	-2	-1	-2	-1	-2	-2	-2	-1	-1	-2	-2	-1	-1	-2	-1	-1
3	M	-2	-1	-1	-2	-2	-1	-2	-1	-2	3	8	-1	-1	-2	-2	-1	-1	-2	-1	-1
4	E	-2	-1	-1	7	-1	-1	-2	-1	4	-2	-2	-1	-1	-2	-2	-2	-1	-2	-1	-1
5	L	-2	-1	-1	-2	-1	-1	-2	-1	4	3	-2	-1	-1	-2	-2	-2	-2	4	-1	-1

6	I	-2	-1	-1	-2	-1	-1	-2	6	-2	1	-2	-1	-1	-2	-2	-2	-2	3	-1	-1
7	E	-2	-1	-1	6	-1	-1	-2	-1	-2	-2	-2	-1	-1	-2	-2	5	-2	-2	-1	-1
8	R	-2	-1	-1	3	-1	-2	-2	-1	-2	-2	-2	-1	-1	5	5	3	-2	-2	-1	-1
9	H	-2	-2	-1	-2	-2	-2	11	-1	-2	-2	-2	-1	-1	-1	-1	-2	-2	-2	-1	-1

By default, `profile2pssm` generates a flat file with a header for each query containing:

Query profile of sequence `#database_key`

`#database_key` corresponds to the database key in the `.index` file of the profile database. `profile2pssm` can also output an indexed database with the `--db-output` parameter. In this case, the header is omitted.

There are three ways to use external PSSM in MMseqs2.

Convert an result database into a profile

All MMseqs2 result database (like clustering, alignment, prefilter results ...) can be transformed into profiles with the `result2profile` module.

```
mmseqs result2profile seqDB seqDB resultDB profileDB
```

Examples how to use `result2profile`. Turn a search result into a profile

```
mmseqs search queryDb targetDb alnDb tmp -a
mmseqs result2profile queryDb targetDb alnDb queryProfileDb
```

Turning a cluster result into a profiles

```
mmseqs cluster sequenceDb clusterDB tmp
mmseqs createsubdb clusterDB sequenceDb sequenceRepDb
mmseqs createsubdb clusterDB sequenceDb_h sequenceRepDb_h
mmseqs result2profile sequenceRepDb sequenceDb clusterDB sequenceRepProfileDb
```

Convert an external MSA into a profile

MMseqs2 can compute profiles from MSAs with the `msa2profile` module. It is possible to use MSAs in FASTA, A3M and CA3M format. In default the first sequence in the MSA is chosen as the query sequence. Gap columns in the query are discarded. But it is also possible to compute a consensus query sequence from the MSA by

```
mmseqs convertmsa stockholm.msa msaDb
mmseqs msa2profile msaDb profileDB
```

The `msa2profile` decide if a column of the multiple sequence alignment will be considered in the profile or not. There are two different modes. The sensitivity of searches with the sequence profiles can depend critically on which columns are represented.

By default, MMseqs2 uses the setting

```
--match-mode 0 (profile column assignment by first sequence in MSA),
```

which means match states are assigned by the first (master) sequence in the MSA: All columns of the first sequence except gaps “-” will be turned into profile columns. This setting can be used for center star MSAs where the first sequence is the center. It is risky for large MSAs in which the first sequence might be not very representative of the entire MSA.

```
--match-mode 1 (profile column assignment by gap fraction)
```

which turns all columns with at least 50% residues (non-gaps) to profile columns and treats all others as insertions. The threshold ratio can be changed with the option

```
--match-ratio 0.5 (change gap fraction threshold for profile column assignment).
```

If you want to use the Pfam database see [How to create a target profile database \(from PFAM\)](#).

`msa2profile` generates also two additional sequences databases: * `profileDB_consensus` contains consensus sequences generated by taking the highest scoring amino acid in each profile position. * `profileDB_seed` contains the representative (= first) sequence of each alignment.

Convert HHsuite HMMs into a profile

It is possible to convert the HH-suite HMM format to MMseqs2 profiles with the `convertprofiledb`. This conversation is only possible if the HMMs do not contain any pseudo counts. The HMMs need to be in a MMseqs2 database format. This can be done using `ffindex_build` (https://github.com/soedinglab/ffindex_soedinglab).

```
hhmake -i 1TIM.a3m -o 1TIM.hmm
hhmake -i 6IGF.a3m -o 6IGF.hmm
ffindex_build hmm hmm.index 1TIM.hmm 6IGF.hmm
mmseqs2 convertprofiledb hmm profileDb
```


Identifier parsing

MMseqs2 parses identifier from the fasta header when transforming a result DB into a flat file by using e.g. `createtsv`, `convertalis`, ...). We support following fasta header types:

```
Uniclust,  
Swiss-Prot,  
Trembl,  
GenBank,  
NCBI Reference Sequence,  
Brookhaven Protein Data Bank,  
GenInfo Backbone Id,  
Local Sequence identifier,  
NBRF PIR,  
Protein Research Foundation,  
General database identifier,  
Patents,  
NCBI GI
```

If none of the header supported could be detected than we extract everything from header start (excluding >) until the first whitespace.

Optimizing Sensitivity and Consumption of Resources

This section discusses how to keep the run time, memory and disk 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.

Prefiltering module

The prefiltering module can use a lot of resources (memory consumption, total runtime and disk 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 $(N * L * 7)$ 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 $a^k * 8$ byte, where a is the size of the amino acid alphabet (default $a=20$, does not include the unknown amino acid X) and k is the k-mer size. The overall memory consumption of the index table is

$$M = (7 * N * L + 8 * a^k) \text{ byte}$$

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

If not enough memory is available to hold the whole database than it will automatically split the target database. This mechanism is triggered when no index `createindex` is used to search. Indexed databases will load the whole index at once.

Runtime

The prefiltering module is the most time consuming step. It can scale from minutes in runtime to days by adjusting the sensitivity setting. Searching with 637000 protein sequences against 30 Mio Uniprot sequences took around 12 minutes on a 16 cores.

Disk Space

The prefiltering results for very large databases can grow to considerable sizes (in the order of TB) of the disk space if very long result lists are allowed and no strict ungapped score threshold is set. As an example, an all-against-all prefiltering run on the 25 Mio sequences with `--max-seqs 300` yielded prefiltering list with an average length of 150 and an output file size of 78 GB. One entry needs roughly 21 byte of space. To compute the worse case hard disk space usage S use the following formula. N is the Database sequence size L is `--max-seqs`.

$$S = (21 * N * L) \text{ byte}$$

Important Options for Tuning the Memory, Runtime and Disk 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. See [Set sensitivity -s parameter](#).

- 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 disk space for databases containing tens of millions of sequences. Changing `--max-seqs` option has no effect on the run time of the prefilter. Decreasing can speed up alignment but might degrade the sensitivity since the order of the prefilter can differ from the alignment.

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

The alignment is based on a striped vectorized algorithm which can process roughly 2 giga cell updates per second (GCUPS). The time to compute the alignment of two average sized proteins (350 residues) takes roughly 2.0625×10^{-4} seconds on one CPU. For example computing 23 Mio. alignments on 8 cores takes 2 minutes.

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-accept` controls the maximum number of alignment results per query sequence.
- The option `--max-rejected` defines the maximum number of rejected sequences for a query until the calculation of alignments stops. The results of the prefilter are aligned from highest ungapped alignment score to lowest.

We increase a rejection counter after each alignment that does not fulfill the alignment criteria such coverage threshold, e-value threshold etc. If the counter is greater than `--max-rejected` then it stops the alignment and does not take a look at any further prefilter hits. The counter is reset whenever an alignment is accepted. Per default, `--max-rejected` is set to `INT_MAX`, i.e. all alignments until `--max-seqs` alignments are calculated.

- The option `--alignment-mode` controls which part of the alignment should be computed. Alignment mode 1: fastest option is to only compute the score and alignment end position. Alignment mode 2: is slightly slower. It computes alignment start,end and score. It is roughly 2 times slower than mode 1. Alignment mode 3: alignment start, end, seq.id score. It is roughly 2.5 times slower than mode 1. The option `-a` is as fast as alignment mode 3.

Disk 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 all hits from the prefilter are accepted by the alignments then the disk space consumption is 1.75 times higher. It needs roughly ~60 byte per record. The option `-a` adds the backtrace and there for increases the size to ~80 byte.

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

$$M = (6 * N * r) \text{ byte}$$

To prevent excessive memory usage for the clustering of large databases, MMseqs2 uses a cascaded clustering by default, which accumulates sequences per cluster incrementally.

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 less than an hour for the clustering of the whole UniProtKB. The core algorithm of cluster mode `--cluster-mode 0, 1, 2` runs single threaded 3 runs multi threaded.

Disk 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.

Workflows

The resource requirements depends on the combination of modules. Search uses the prefilter and alignment module. Clustering uses the prefilter, alignment and clustering module.

The cascaded clustering sets all the options controlling the size of the output, speed and memory consumption, internally adjusting parameters in each cascaded clustering step.

Each workflow uses a different set of default parameters that can be changed.

How to run MMseqs2 on multiple servers using MPI

MMseqs2 can run on multiple cores and servers using OpenMP and message passing interface (MPI). MPI assigns database splits to each servers and each server computes them using multiple cores (OpenMP). 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. The number of chunks is controlled through the `--split` parameter. 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:

$$((7 * N * L) / \text{\#chunks} + 21^k * 8) \text{ byte}$$

Thus, it 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 precompiled 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 -pernode -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 -pernode -np 42" mmseqs cluster DB clu tmp
```

Write temporary files to local disk when running with MPI

As part of its computation, MMseqs2 writes temporary files corresponding to each of the database splits. The number of the database splits is determined by the number of servers and number of cores in each server. By default, temporary files are written to the shared disk.

In case the number of database splits is large, it would mean a high burden of I/O operations on the same disk. To avoid slowdowns due to this issue, an additional parameter can be passed to `search` in MPI mode:

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

Passing this parameter will write the temporary files of each server (created by its cores) on the indicated path (local disk) and reduce the number of temporary files handled on the shared disk.

How to run MMseqs2 on multiple servers using batch systems

MMseqs2 comes with tools to split database to distribute jobs in batch systems (like qsub, slurm, moab, lsf, ...). However, the MPI solution is preferred if available since it optimizes the distribution of computing load. Splitting the query database can be used to distribute the load, MMseqs2 has a module called `splitdb` that splits the database in `--split N` chunks. Each can be submitted separately to the grid system.

```
# script to splits the database in 3 parts and submit them to the grid
# split query db
SPLITS=3
QUERYFASTA=queryDB.fasta
QUERYDB=queryDB
mmseqs createdb "${QUERYFASTA}" "${QUERYDB}"
mmseqs splitdb "${QUERYDB}" "${QUERYDB}_split" --split $SPLITS

# create header database to support full mmseqs functionality
# this step can be used if queryDB is used in downstream steps
for file in $(ls "${QUERYDB}_split"_*_$SPLITS); do
    mmseqs createsubdb "${file}.index" "${QUERYDB}_h" "${file}_h"
done

# submit job
split=0
for file in $(ls "${QUERYDB}_split"_*_$SPLITS); do
    bsub mmseqs search "${file}" targetDB aln_${split} tmp
    ((split++))
done
```

Frequently Asked Questions

This section describes common questions.

How to set the right alignment coverage to cluster

MMseqs2 has three modes to control the sequence length overlap “coverage”: (1) bidirectional, (2) target coverage and (3) query coverage. In the context of **cluster** or **linclust**, the query is seen representative sequence and target is a member sequence. The **--cov-mode** flag also automatically sets the **--cluster-mode**.

- (1) With **--cov-mode 0 -c [0.0,1.0]** only sequences are clustered that have a sequence length overlap greater than X% of the longer of the two sequences. This coverage mode should be used to cluster full length protein sequences. The multi domain structure of proteins will be most likely preserved when using a coverage > 80% (**-c 0.8**). Default **--cluster-mode** is the greedy set cover.

For example:

```
q: MAVGTACRPA
t: -AVGTAC---
```

The coverage of this alignment would be 6/10=60%.

```
q: -AVGTAC---
t: MAVGTACRPA
```

The coverage of this alignment would be 6/10=60%.

- (2) With `--cov-mode 1 -c [0.0,1.0]` (target-cov mode) only sequences are clustered that have a sequence length overlap greater than X% of the target sequence. The target cov mode can be used to cluster protein fragments. To suppress fragments from becoming representative sequences, it is recommended to use `--cluster-mode 2` in conjunction with `--cov-mode 1`. Default `--cluster-mode` is the greedy incremental clustering (by length).

For example:

```
q: MAVGTACRPA
t: -AVGTAC---
```

The target coverage would be 6/6=100%.

```
q: -AVGTAC---
t: MAVGTACRPA
```

The target coverage would be 6/10=60%.

- (3) With `--cov-mode 2 -c [0.0,1.0]` (query-cov mode) only sequences are clustered that have a sequence length overlap greater than X% of the query sequence. The query coverage mode can be used while searching e.g. to assure a certain level of coverage.

For example:

```
q: MAVGTACRPA
t: -AVGTAC---
```

The query coverage would be 6/10=60%.

```
q: -AVGTAC---
t: MAVGTACRPA
```

The query coverage would be 6/6=100%.

How does MMseqs2 compute the sequence identity

MMseqs2 computes the sequence identity in two different ways:

- (1) When using `--alignment-mode 3` mmseqs2 will compute the number of identical aligned residues divided by the number of aligned columns including columns containing a gap in either sequence.
- (2) By default, the sequence identity is estimated from the score per column, i.e., the local alignment bit score divided by the maximum length of the two aligned sequence segments. The estimate uses the linear regression function (shown in red below) between the sequence identity computed as in (1) and the score per column in the scatter plot:

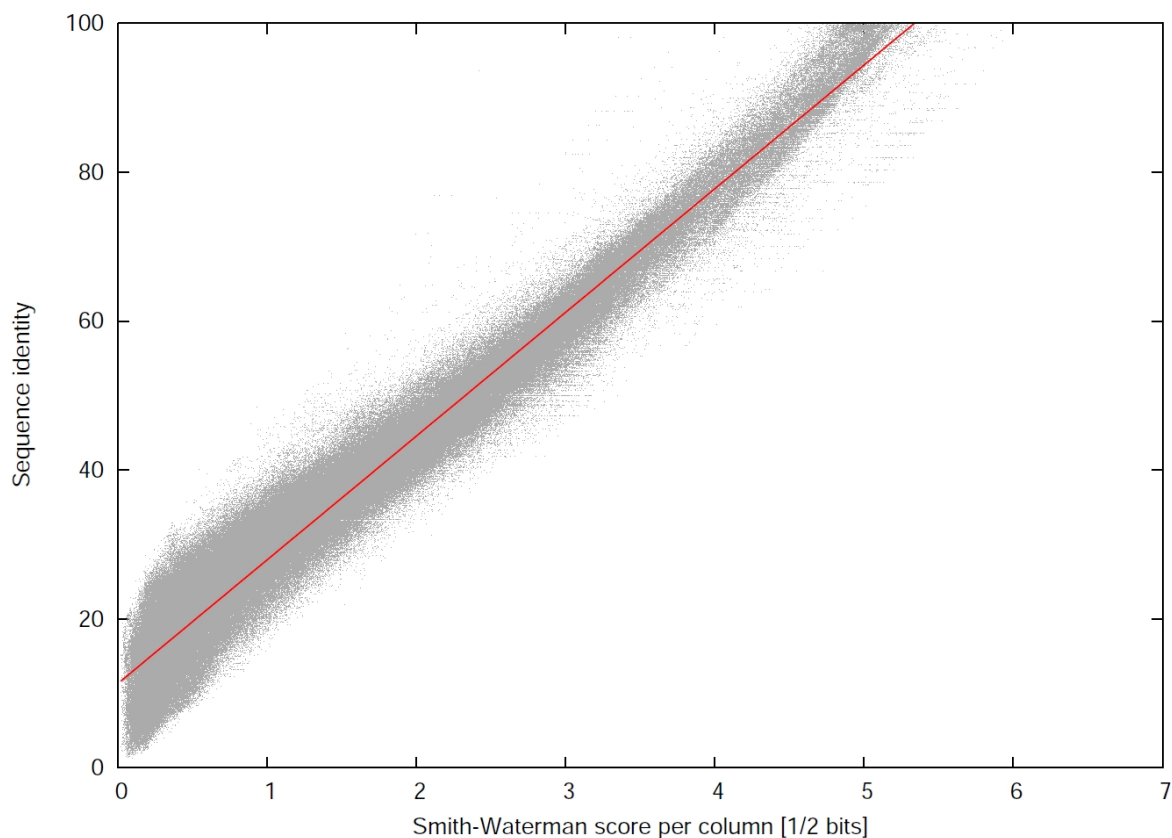


Figure 0.12: Relationship between score per column and sequence identity

The score per column is a better measure of the degree of similarity than the actual sequence identity, because it also takes the degree of similarity between aligned amino acids and the number and length of gaps into account.

How to restart a search or clustering workflow

MMseqs2 checks if files are already computed in the `tmpDir` and skips already computed results. To restart delete temporary result files from the crashing step that were created by MMseqs2 and restart the workflow with the same program call again. You can recognize the temporary files that should be deleted by their file ending `. [0-9]+`.

If the job crashed while merging files they can be merged manually using `ffindex_build` (https://github.com/soedinglab/ffindex_soedinglab). For example, if the merge step of the alignment fails while using 56 threads then the result could be recovered by using the following command.

```
for i in $(seq 0 55); do ffindex_build -a aln{,.tmp.index} -d aln.$i -i aln.index.$i ; done
LC_ALL=C sort --parallel 28 -n -k 1,1 aln.tmp.index > aln.index
```

How to find the best hit the fastest way

MMseqs2 can apply an iterative approach to speed up best-hit-searches. It will start searching with the lowest sensitivity defined with `--start-sens` and search until the target sensitivity `-s` is reached. The amount of steps to reach `-s` can be defined with `--sens-steps`.

Queries are only used again in the next iteration, if no match could be found that fulfilled the acceptance criteria in the previous iteration.

For example, the following search performs three search steps with sensitivity `-s 1, 4 and 7`.

```
mmseqs search qDB tDB rDB tmp --start-sens 1 --sens-steps 3 -s 7
```

Using this iterative approach can speed up best-hit-searches 4-10 times.

There is a chance that the best hit is not found but the chances are low. Prefilter hits found at a lower sensitivity threshold, have more highly conserved k-mers in common. This effect can be reduced if a higher start sensitivity is used (`-start-sens 4`).

If any hit is good enough add the `--max-accept 1` option to gain a further speedup.

How does MMseqs2 handle low complexity

MMseqs2 uses tantan to reduces low complexity effects on the query and target database.

Query sequences are handled by an amino acid local compositional bias correction. In prefilter and alignment stages we apply a correction to substitution matrix scores assigning lower scores

to the matches of amino acids that are overrepresented in the local sequence neighborhood. To switch the compositional bias correction on and off use `--comp-bias-corr`.

Target sequences low-complexity regions are masked during the prefilter stage. We use TANTAN with a threshold of 90% probability for low complexity. Masking can be controlled with `--mask`.

How to redundancy filter sequences with identical length and 100% length overlap.

To redundancy filter sequences of identical length and 100% overlap `mmseqs clusthash` can be used. It reduces each sequence to a five-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.

Example: cluster sequences at 90% sequence identity

```
mmseqs clusthash sequenceDB resultDB --min-seq-id 0.9
mmseqs clust sequenceDB resultDB clusterDB
```

How to add sequence identities and other alignment information to a clustering result.

We can add sequence identities and other alignment information to the clustering result `outDB` by running an additional align step:

```
mmseqs cluster sequenceDB resultDb tmp
mmseqs align sequenceDB sequenceDB resultDb alignDB -a
mmseqs convertalis sequenceDB sequenceDB alignDB align.m8
```

The `-a` parameter computes the whole backtrace. `--alignment-mode 3` could be used instead if the backtrace is not needed. This would save disk space. The backtrace is however computed anyway (for the calculation of the sequence identities) and then discarded.

How to run external tools for each database entry

The `apply` module can be used to call an external tools on each entry of a MMseqs2 database. It works like the map step from the map/reduce pattern. It calls for every index entry the specified process with the passed parameters. The process reads the entry data from `stdin` and its `stdout` is written to a new entry in the result database (with the same key). The tool supports OpenMP and MPI parallelization for spreading out the job over several compute nodes.

Example: An awk script which takes an alignment result entry from stdin and prints out all lines with an e-value <0.001 to stdout (Hint: the `filterdb` module can also solve this problem, but with less overhead):

```
mmseqs apply resultDB filteredResultDB -- awk '$4 < 0.001 { print; }'
```

The `apply` module exports the `MMSEQS_ENTRY_NAME` environment variable into the called processes. It contains the current database key.

How to compute a multiple alignment for each cluster

There are two ways to produce MSAs from a clustering

- (1) MMseqs2 `mmseqs result2msa` can produce a MSA using a centre star alignment without insertions in the query.

```
mmseqs cluster DB DB_clu tmp
mmseqs result2msa DB DB DB_clu DB_clu_msa
```

- (2) The `mmseqs apply` module can be used to call an external multiple aligner. The multiple aligner need the capability to read stdin and write the result to stdout.

```
mmseqs cluster DB DB_clu tmp
mmseqs createseqfiledb DB DB_clu DB_clu_seq
mmseqs apply DB_clu_seq DB_clu_seq_msa -- clustalo -i - --threads=1
```

How to manually cascade cluster

It is possible to cluster the representative sequences of an clustering run and merge the cluDB results with the following workflow.

```
# first clustering run
mmseqs linclust sequenceDB clu1 tmp1
# create a subset of the sequenceDB only with representative sequences
mmseqs createsubdb clu1 sequenceDB cluSequenceDB
# cluster representative sequences
mmseqs cluster cluSequenceDB clu2 tmp2
# merge two clusterings in to one results
mmseqs mergecluster sequenceDB final_clu clu1 clu2
```

How to cluster using profiles

The following workflow is a profile consensus clustering.

1.) Enrich the sequences:

```
# enrich your database to cluster (seqDB1) by searching it against a database seqDb2
mmseqs search seqDB1 seqDB2 resultDB1 tmp
# turn seqDB1 into profiles
mmseqs result2profile seqDB1 seqDB2 resultDB1 profileDB1
```

2.) Cluster profiles by searching the profiles against its consensus sequences

```
# search with profiles against consensus sequences of seqDB1
mmseqs search profileDB1 profileDB1_consensus resultDB2 tmp --add-self-matches # Add your c
# cluster the results
mmseqs clust profileDB1 resultDB2 profileDB1_clu
```

How to create a HHblits database

One can turn the output of a search (or clustering) into a HHblits database. You need to have HH-suite properly installed with MPI support. The following procedure creates an HHblits-compatible database “searchMsa” resulting from the enrichment of sequences of “DBquery” with the sequences of “DBtarget”:

```
mmseqs search DBquery DBtarget searchOut tmp -a
mmseqs result2msa DBquery DBtarget searchOut searchMsa --compress
# MPI version
mpirun -np 2 cstranslate_mpi -i searchMsa -o searchMsa_cs219 -x 0.3 -c 4 -I ca3m -b
# no MPI version
mpirun -np 2 cstranslate -i searchMsa -o searchMsa_cs219 -f -x 0.3 -c 4 -I ca3m -b
```

The files /path/to/cs219.lib and /path/to/context_data.lib are provided in the “data” subfolder of your HH-suite installation. The parameters -x 0.3 -c 4 have been empirically found to perform well.

For creating an HHblits database from a clustering, the procedure is almost the same, except that you have to create symlinks to the findex _header and _sequence files needed by HHblits:

```
mmseqs cluster DB clu tmp
mmseqs result2msa DB DB clu cluMsa --compress
ln -s DB_h cluMsa_header.ffdata
ln -s DB_h.index cluMsa_header.ffindex
ln -s DB cluMsa_sequence.ffdata
ln -s DB.index cluMsa_sequence.ffindex
mpirun -np 2 cstranslate_mpi -i cluMsa -o cluMsa_cs219 -A /path/to/cs219.lib -D /path/to/con
```

In the “search” case, those files are generated by MMseqs2, since it needs to merge the query and the target sequence databases. No merging is done for clustering, since both the query and target sequence database are the same.

How to create a target profile database (from PFAM)

Download the latest version of the PFAM in stockholm format:

```
wget http://ftp.ebi.ac.uk/pub/databases/Pfam/current_release/Pfam-A.full.gz
```

Convert stockholm MSAs to a FASTA formatted MSA database.

```
mmseqs convertmsa Pfam-A.full.gz pfam_msa_db
```

Create a profile database in MMseqs2 format.

To turn an MSA into a sequence profile, mmseqs2 needs to decide for each column whether it will be represented by a column in the sequence profile or not. The sensitivity of searches with the sequence profiles can depend critically on which columns are represented. By default, mmseqs2 uses the setting

```
--match-mode 0 (profile column assignment by first sequence in MSA),
```

which means match states are assigned by the first (master) sequence in the MSA: All columns where this master sequence has a residue will be turned into profile columns, all others will be ignored and the residues in them will be modeled as insertions relative to the sequence profile.

This is risky for large Pfam MSAs in which the first sequence might be not very representative of the entire family. A better choice for Pfam is therefore

```
--match-mode 1 (profile column assignment by gap fraction)
```

which turns all columns with at least 50% residues (non-gaps) to profile columns and treats all others as insertions. The threshold ratio can be changed with the option

`--match-ratio 0.5` (change gap fraction threshold for profile column assignment) .

We compute sequence profiles from the FASTA MSAs using

```
mmseqs msa2profile pfam_msa_db pfam_profile --match-mode 1
```

Precompute mmseqs index table (not required for a single search run). Use the `--no-preload` flag later in the search, if the query database is small to medium sized. Without that the precomputed index table will be first read completely into memory (unnecessary overhead).

```
mmseqs createindex pfam_profile tmp -k 5 -s 7
```

Search now against the created profile database:

```
mmseqs search query_db pfam_profile result tmp -k 5 -s 7
```

If your machine has a lot of main memory, use 6-mers for the search (and index building) by specifying `-k 6` instead of `-k 5`. 6-mers provide slightly better sensitivity, however need about 5x more memory (about 25GB instead of 5GB for PFAM, not including the per-thread memory).

How to cluster a graph given as tsv or m8 file

MMseqs2 needs two things to cluster an external graph (1) a sequence database and an (2) result file.

As a first step create your sequence database by calling `createdb` on your input fasta file.

```
mmseqs createdb sequence.fasta sequence
```

It is possible to transform an external TSV in m8 format (BLAST tab) into an result file database using `tsv2db`. The m8 or tsv file must contains a self hit “ID1 ID1 ...” for each entry in the sequence.fasta. Also we need to overwrite the identifier (first and second column) with numerical identifier for the sequence database before calling `tsv2db`.

```
awk 'NR == FNR { f[$2] = $1; next} { line = f[$1]"\t"f[$2]; for(i = 3; i <= NF; i++){ line  
mmseqs tsv2db result.m8.newid result
```

Now we should be able to use the internal clustering (greedy incremental, connected component, set cover) algorithm of MMseqs2

```
mmseqs clust sequence result clu
```

How to search small query sets fast

MMseqs2 can perform single queries fast if the whole index fits into memory by using memory-mapped file I/O (mmap). If the target database is not in memory, MMseqs2 needs to load GBs from the hard disk for each query, which will take minutes. The following workflow creates an index and fetches the database into memory and efficiently searches through it.

First we need to setup the database by creating a database with **createdb**, index it **createindex** and then load the index into memory **touchdb** or **vmtouch** (<https://github.com/hoytech/vmtouch>).

```
mmseqs createdb targetDB.fasta targetDB
mmseqs createindex targetDB tmp
mmseqs touchdb targetDB
# alternative using vmtouch
vmtouch -l -d -t targetDB.idx
```

Once the database is in memory it is possible to run instant searches against it by using the **--db-load-mode 2**

```
mmseqs search queryDB targetDB aln tmp --db-load-mode 2
mmseqs convertalis queryDB targetDB aln aln.m8 --db-load-mode 2
```

The **touchdb** module fetches the precomputed index database into memory and **--db-load-mode 2** tells MMseqs2 to **mmap** the database instead of copying the whole precomputed index into memory. This saves, for a large database, minutes of copying from the storage system into RAM. However, this is less efficient for large query sets.

The reason for this is that copying the index into memory will use transparent huge pages (on operating systems that support THP), which in turn results in less TLB cache misses during the prefiltering stage. For a large query set the use of THP results in 15-20% speed improvement of the prefiltering stage.

What is the difference between the map and search workflow

The **map** and **search** workflow both call the **search** workflow. The only difference are the default parameters and alignment strategy. **map** should be used for high identities which is more of a technical task rather than measuring homology. Therefore we turned off compositional bias correction, we do not mask the database, search with low sensitivity **-s 2** and we align only ungapped.

How to build your own MMseqs2 compatible substitution matrices

You can use the (still Work in Progress) R script in `util/format_substitution_matrix.R` to prepare a substitution matrix for use in MMseqs2. It takes a matrix from `stdin` and returns the reformatted matrix to `stdout`.

```
util/format_substitution_matrix.R < data/MATRIX.mat > data/MATRIX.out
mmseqs search query db res tmp --sub-mat data/MATRIX.out
```

Workflow Control Parameters

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 [1:8.5] (default=4).

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. 1.0 fastest - 8.5 sensitive. The sensitivity between 8 to 8.5 should be as sensitive as BLAST. For detailed explanation see section [Computation of Prefiltering Scores using mmseqs prefilter](#).

Clustering Workflow

Calculates the clustering of the sequences in the input database.

Usage:

```
mmseqs cluster <sequenceDB> <outDB> <tmpDir> [opts]
```

Options:

`--single-step-clustering` Starts the single-step instead of the cascaded clustering workflow.

The database can be clustered in a single step instead of with a cascaded workflow. This increases runtime and memory requirements substantially and decreases sensitivity, but guarantees, that all cluster members strictly fulfill the selection criteria, such as sequence identity or coverage. After merging of clusters in the cascaded clustering, the e.g. sequence identity of the representative

with the members of the to be merged cluster, might fall under the given sequence identity threshold.

`-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 [Computation of Prefiltering Scores using mmseqs prefilter](#).

`--min-seq-id [float]` list matches above this sequence identity [0.0:1.0] (default=0.0).

Read more about how MMseqs2 computes sequence identity in section [How does MMseqs2 compute the sequence identity](#).

`--cov-mode [int]` "0: coverage of query and target, 1: coverage of target [0:1] (default=0). `-c [float]` "list matches above this fraction of covered residues (see cov-mode) [0.0:1.0] (default=0.8). Read more about coverage is computed at section [How to set the right alignment coverage to cluster](#)

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.

External Libraries used in MMseqs2

We would also like to thank the developers of the open source librarys used in MMseqs2:

- [Striped Smith-Waterman Library](#)
- [ALP Library](#)
- [TANTAN](#)
- [Open MP Template Library](#)
- [ksw](#)
- [kseq](#)
- [iota](#)

- [blast2lca](#)
- [sse2neon](#)
- [zstd](#)
- [tinyexpr](#)

License Terms

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