*Bioinformatics*, 32(9), 2016, 1323–1330
doi: 10.1093/bioinformatics/btw006
Advance Access Publication Date: 6 January 2016
Original Paper

OXFORD

Sequence analysis

# MMseqs software suite for fast and deep clustering and searching of large protein sequence sets

**Maria Hauser[1,†], Martin Steinegger[1,2,3,*,†] and Johannes Söding[1,2,*]**

[1]Gene Center, Ludwig-Maximilians-Universität München, Munich 81377, Germany, [2]Computational Biology, Max Planck Institute for Biophysical Chemistry, Göttingen 37077, Germany and [3]TUM, Department of Informatics, Bioinformatics & Computational Biology-I12, Garching 85748, Germany

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.
Associate Editor: John Hancock

Received on 24 September 2015; revised on 9 December 2015; accepted on 1 January 2016

## Abstract

**Motivation:** Sequence databases are growing fast, challenging existing analysis pipelines. Reducing the redundancy of sequence databases by similarity clustering improves speed and sensitivity of iterative searches. But existing tools cannot efficiently cluster databases of the size of UniProt to 50% maximum pairwise sequence identity or below. Furthermore, in metagenomics experiments typically large fractions of reads cannot be matched to any known sequence anymore because searching with sensitive but relatively slow tools (e.g. BLAST or HMMER3) through comprehensive databases such as UniProt is becoming too costly.

**Results:** MMseqs (*M*any-against-*M*any *seq*uence *s*earching) is a software suite for fast and deep clustering and searching of large datasets, such as UniProt, or 6-frame translated metagenomics sequencing reads. MMseqs contains three core modules: a fast and sensitive prefiltering module that sums up the scores of similar *k*-mers between query and target sequences, an SSE2- and multi-core-parallelized local alignment module, and a clustering module.

In our homology detection benchmarks, MMseqs is much more sensitive and 4–30 times faster than UBLAST and RAPsearch, respectively, although it does not reach BLAST sensitivity yet. Using its cascaded clustering workflow, MMseqs can cluster large databases down to ∼30% sequence identity at hundreds of times the speed of BLASTclust and much deeper than CD-HIT and USEARCH. MMseqs can also update a database clustering in linear instead of quadratic time. Its much improved sensitivity-speed trade-off should make MMseqs attractive for a wide range of large-scale sequence analysis tasks.

**Availability and implementation:** MMseqs is open-source software available under GPL at https://github.com/soedinglab/MMseqs

**Contact:** martin.steinegger@mpibpc.mpg.de, soeding@mpibpc.mpg.de

**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

## 1 Introduction

During the last 8 years, sequencing costs have come down from 10 000 000$ to less than 1000$ for a human genome at 30 times coverage (http://www.genome.gov/sequencingcosts). As a consequence, protein sequence databases such as the UniProt (Bairoch *et al.*, 2005) database have been growing by a factor of 2 every two

years (Apweiler *et al.*, 2004), leading to longer search times, inflated and redundant results list, large memory requirements and saturating or decreasing sensitivities for informative sequence matches (Chubb *et al.*, 2010).

A solution is to compute a representative subset of sequences by clustering them by their similarity and selecting one representative per cluster. Such clustering schemes achieve a more even sampling, leading to better sensitivities in sequence searches (Li *et al.*, 2002; Park *et al.*, 2000). UniRef provides representative subsets of UniProtKB clustered at 100%, 90% and 50% sequence identity (Suzek *et al.*, 2007). Clustering schemes are also used in metagenomics projects to reduce the size and redundancy of the ever larger amounts of sequence data (Sunagawa *et al.*, 2015).

Several tools for clustering protein sequence databases have been developed. BLASTclust from the NCBI BLAST package is sensitive but slow. It uses greedy single-linkage clustering based on all-versus-all blastp searches (Altschul *et al.*, 1990). The fast tools CD-HIT, (Fu *et al.*, 2012), USEARCH (Edgar, 2010) and kClust (Hauser *et al.*, 2013) share several similarities. First, they employ the same incremental, greedy clustering scheme, in which each database sequence (the 'query') is compared with the representative sequences of already established clusters. If one of the representative sequences is sufficiently similar, the query is added to this cluster or otherwise becomes the representative of a new cluster. Second, all three tools employ a $k$-mer word-based similarity prefilter that drastically reduces the number of slow but accurate Smith-Waterman alignments. The prefilters in CD-HIT and USEARCH count the number of common, identical $k$-mer words between sequences, with $k = 5$ or 6 for USEARCH and $k$ between 2 and 5 for CD-HIT.

To obtain a sufficient number of common $k$-mers between sequences with only 50% residue-wise sequence identity, CD-HIT has to lower $k$ to 2. But this leads to a high probability $\sim 1/20^k$ for chance $k$-mer matches. Therefore, the number of chance matches in an all-against-all comparison of $N$ sequences of average length $L$ is around $(NL)^2/20^k$, which becomes huge for small $k$. Since each chance match costs a constant amount of time to process, short words lead to an enormous slow-down.

kClust employs a $k$-mer-based prefilter that can even detect pairs at 20–30% sequence identity at high speed. To keep the probability for chance matches low and speed high, it uses long words with $k = 6$ or 7. But to increase sensitivity at the same time, it detects *similar* instead of just identical $k$-mers. For each $k$-mer in the query sequence, it computes a list of all $k$-mers with a BLOSUM62 bit score above a certain cut-off and finds identical matches to these similar $k$-mers in the database sequences. The prefilter then scores each database sequence by the sum of similarity scores of similar $k$-mers.

A further challenge arising from the rapid progress in high-throughput sequencing is the need for sensitive but fast protein sequence search methods. A large fraction of metagenomics reads cannot be mapped to any known sequence from a cultivated organism anymore, because it has become too costly to search through the entire UniProt database using a sensitive but slow tool such as BLASTX (Altschul *et al.*, 1990): It would take approximately 2398 CPU years to search with all 6-frame translated sequences from 2 $\times 10^9$ reads of length 150 nucleotides through the current UniProt database using BLASTX. Instead, in most projects, much smaller databases are searched, such as KEGG GENOME (Kanehisa and Goto, 2000), a collection of high-quality genome sequences, or the MetaPhlAn (Segata *et al.*, 2012) database of unique clade-specific marker genes (Human Microbiome Project Consortium, 2012). This carries the risk of missing some of the most interesting matches, which do not conform to prior expectations. To address this challenge, a number of fast protein sequence search tools have been developed: Tachyon (Tan *et al.*, 2012), PAUDA (Huson and Xie, 2014), PSimScan (Kaznadzey *et al.*, 2013), RAPsearch2 (Zhao *et al.*, 2012), Lambda (Hauswedell *et al.*, 2014), UBLAST (Edgar, 2010) and DIAMOND (Buchfink *et al.*, 2015). The latter five, which are the most sensitive in this list, find exact matches of (spaced) $k$-mers and extend the alignment around them.

MMseqs addresses the need for a clustering and search tool that is both fast and sensitive enough to be able to detect sequence matches down to ~30% residue-wise sequence identity. While it uses the same core prefiltering algorithm as kClust, it has various important advantages: (i) Its organization into modules (prefiltering, alignment, clustering) and workflows increases flexibility and facilitates future extensions. (ii) Its search workflow can perform sequence searches. With a speed 1000 times faster than BLAST it finds similarities down to 30% sequence identity and is much more sensitive than similarly fast search tools. (iii) Its cascaded clustering workflow achieves much deeper clustering than kClust in a shorter time. (iv) Its database updating workflow adds sequences to a previously clustered set in linear time, obviating the need for frequent reclustering in quadratic time. (v) MMseqs is implemented highly efficiently, using SIMD (single-instruction-multiple-data) instructions to vectorize time-critical loops. (vi) It is parallelized using OpenMP to run on multi-core CPUs (vii) To save memory, the database can be divided into several parts and processed consecutively. (viii) Its prefilter uses a novel $Z$-score statistic for higher sensitivity and a score correction for compositionally biased sequence regions. (ix) It offers the greedy set cover algorithm for clustering, in addition to the simple, incremental algorithm used by kClust, USEARCH and CD-HIT, enabling deeper clusterings. (x) It performs exact Smith-Waterman alignment based on the striped SIMD algorithm (Farrar, 2007) instead of the approximate $k$-mer dynamic programming algorithm developed for kClust.

## 2 Methods

MMseqs contains three core modules: (i) The *prefilter module* computes a $k$-mer-based similarity score between all sequences from the 'query' set with all sequences from the 'target' set. (ii) The *alignment module* can read prefiltering results and computes Smith-Waterman alignments between query-target pairs that pass a prefilter $Z$-score threshold. (iii) The *clustering module* reads in the results of the alignment module, for which a sequence set must have been compared to itself (query set = target set), and groups sequences into clusters, using user-specified thresholds on sequence similarity, alignment coverage and $E$-value. In addition to the modules, three workflows for sequence searching, clustering and updating a clustering facilitate the most common tasks for the non-expert.

### 2.1 Prefilter module

#### 2.1.1 Prefilter score

The prefilter module is crucial for the speed and sensitivity of MMseqs as it needs to reduce the number of sequences to be aligned with the relatively slow Smith-Waterman algorithm from millions to tens or hundreds per each query sequence while compromising sensitivity as little as possible. For each query sequence it computes a raw prefilter similarity score with each target sequence. This prefilter score $S_{\text{pref}}$ is the sum of similarity scores for all pairs of $k$-mer words $(x, y)$ whose similarity score $S_k(x, y)$—evaluated with the

BLOSUM62 (Henikoff and Henikoff, 1992) substitution matrix—surpasses a minimum score threshold $S_{\min}$:

$$S_{\text{pref}}(\text{query}, \text{target}) = \sum_{x \in \text{query}, y \in \text{target}, S_k(x,y) \geq S_{\min}} S_k(x,y). \quad (1)$$

This score, first introduced with kClust, has a fundamental advantage over the prefiltering score used in CD-HIT and USEARCH, which counts the number of identical $k$-mers: We can increase the sensitivity of the search while still maintaining relatively high specificity of the $k$-mer matches by lowering the minimum similarity threshold $S_{\min}$ while keeping the word length fixed and high (e.g. $k = 6$). In contrast, to maintain a sufficient number of $k$-mer matches in homologous sequence pairs with low similarity, CD-HIT needs to shorten $k$ down to 2, thereby loosing specificity of $k$-mer matches and thus incurring dramatically increased run times.

### 2.1.2 Z-score

The expected 'background score' for a pair of non-homologous sequences of lengths $L_q$ and $L_t$ is proportional to the number of expected chance $k$-mer matches, which is roughly proportional to the number of $k$-mers in the target sequence, $L_t - k + 1$. We subtract the expected background score from the raw prefilter score to improve the discrimination of true and false positives. Since some queries tend to generate more chance $k$-mers than others, we estimate the expected background score by acquiring match statistics on a small, randomly sampled subset of target sequences for each query sequence prior to the actual search. To account for the variance of the background score we divide the background-corrected score by the expected standard deviation, assuming a Poisson distribution for the number of $k$-mer matches. These results in the final prefilter $Z$-score (see Supplemental Material).

### 2.1.3 Local compositional bias correction

Sequence regions with biased amino acid composition, such as transmembrane helices, coiled coils or disordered regions, may cause artificially elevated rates of chance $k$-mer matches between unrelated proteins, leading to high-scoring false positive sequence matches. To reduce this risk, we add to the $k$-mer score $S_k(x,y)$ a correction $\sum_{i=j+1}^{j+k} \Delta S_i(x_i)$ that depends on the amino acids $x_i$ composing the query $k$-mer $x$. $\Delta S_i(x_i)$ is minus the average BLOSUM62 score between $x_i$ and the amino acids within $\pm 20$ residues from $x_i$ (except $x_i$ itself) plus $\sum_{a=1}^{20} f(a) S(x_i, a)$, the expected BLOSUM62 score of $x_i$ with a random amino acid $a$ assuming background frequencies $f(a)$. Thus, amino acids that are similar to amino acids enriched in the local sequence neighbourhood receive lower scores (see Supplemental Material).

### 2.1.4 Core algorithm

For each query sequence (for-loop 1 in Fig. 1) and each overlapping $k$-mer $x$ in the query (for-loop 2), a list of similar $k$-mers and their scores, $\mathbb{L}_{\text{sim}}(x) = \{(y, S_k(x,y)) : S_k(x,y) > S_{\min}\}$, is generated (orange box in Fig. 1). For each similar $k$-mer $y$ (loop 3), we look up in a precomputed *index table* (blue box) the list $\mathbb{L}_{\text{IDs}}(y)$ of target sequence IDs that contain the $k$-mer $y$ (green box). In the most time-critical, innermost loop 4, we add $S_k(x,y)$ to the score of each of the target sequences $t \in \mathbb{L}_{\text{IDs}}(y)$: $S_{\text{pref}}(t) += S_k(x,y)$.

The index table is computed by the prefilter module prior to the actual search. It consists of the $21^k$ lists $\mathbb{L}_{\text{IDs}}(y)$ of target sequence IDs, one list for each of $21^k$ $k$-mers $y$ (green box) and an array of

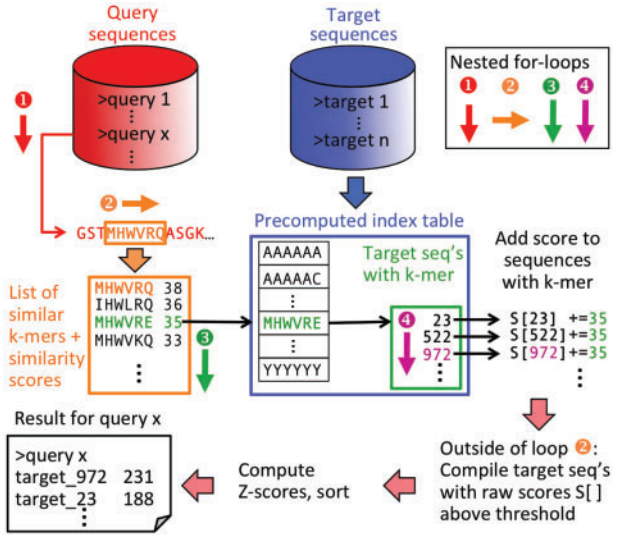


**Fig. 1.** Core prefiltering algorithm. The algorithm computes for all query-target sequence pairs the score in Eq. (1). See Section 2.1 for details

$21^k$ pointers to the lists (black box). The 21st letter X represents unknown amino acids.

After having processed all $k$-mers of a query, we extract the small fraction of target sequences with non-zero scores by checking in parallel using SSE2 instructions if any of the 8 short integer scores differs from 0. This manual vectorization speeds up extraction by around 8-fold. IDs of sequences with non-zero scores are written into a list and final Z-scores are computed.

### 2.1.5 Time complexity

Let us define $N_q$, $N_t$, $L_q$ and $L_t$ as the numbers of sequences in the query and target sets and their average lengths, respectively. Let $l_{\text{sim}} = E_x[|\mathbb{L}_{\text{sim}}(x)|]$ denote the average length of the lists of similar $k$-mers, and let $l_{\text{IDs}} = E_y[|\mathbb{L}_{\text{IDs}}(y)|]$ be the average length of the lists of target IDs. It can be estimated as $l_{\text{IDs}} = N_t L_t / 20^k$, since the target set contains approximately $N_t L_t$ $k$-mers distributed over $20^k$ $k$-mers. The total time for the prefilter is then approximately

$$T_{\text{pref}} \approx N_t L_t T_{\text{index}} + N_q L_q l_{\text{sim}} (T_{\text{sim}} + l_{\text{IDs}} T_{\text{match}}) + N_q N_t T_{\text{extract}}. \quad (2)$$

The first term is the time to build the index table, $N_q L_q l_{\text{sim}} T_{\text{sim}}$ is the time for generating the lists $\mathbb{L}_{\text{sim}}(x)$. The term $N_q L_q l_{\text{sim}} l_{\text{IDs}} T_{\text{match}} = N_q L_q N_t L_t l_{\text{sim}} / 20^k$ describes the contribution of adding the $k$-mer similarity scores to the target sequence scores in the innermost loop, which dominates the runtime when searching large databases with intermediate or high sensitivity. $N_t T_{\text{extract}}$ is the time to extract for each query the target sequences with the most significant scores.

It is desirable to choose $k$ and $S_{\min}$ such that the average list length $l_{\text{IDs}} \approx N_t L_t / 20^k$ does not drop below 1, because otherwise much time is lost in generating lists of similar $k$-mers that do not lead to a $k$-mer match. We should therefore choose $k = 7$ for $N_t L_t \geq 5 \times 20^7 \approx 6 \times 10^9$ and $k = 6$ below that limit. Here, we have used $k = 6$ throughout, and, when setting $k = 7$, further improvements in sensitivity-speed trade-off are possible for large databases such as UniProt (for which $N_t L_t \approx 2 \times 10^{10}$).

The main parameter to control the sensitivity-speed trade-off in MMseqs is the minimum $k$-mer score $S_{\min}$, which determines the average length $l_{\text{sim}}$. This trade-off can be set by the MMseqs sensitivity parameter $S$ using option '-s ⟨$S$⟩'.

The prefilter is the time-limiting module for searching and clustering large databases. It can run on multiple cores (Supplementary Fig. S1) thanks to OpenMP parallization of for-loop 1 in Figure 1 (Supplementary Fig. S2).

#### 2.1.6 Memory requirements

The index table requires $21^k \times 16$B (bytes) to store the array of pointers and the list lengths, which amounts to 654 MB for $k = 6$. The dominating contribution however is to store the lists of sequence IDs in the index table. Each ID is encoded with 4 B, which sums up to a total of $N_t L_t \times 4$B. For the UniProtKB containing $N_t = 54$M sequences of average length $L_t \approx 350$ we need around 70 GB of main memory.

To run MMseqs with a much smaller main memory, the prefilter module can split the target set into equal-sized chunks, only one of which needs to be held in main memory at a time. After all chunks have been processed the results are combined to yield the same results as if done in a single search. This can come at a cost in run time if the average list length $l_{\mathrm{IDs}}$ drops below $\sim 5$ (see Section 11.1 of user guide for details).

### 2.2 Alignment module

This module computes exact, unbounded Smith-Waterman alignments with affine gap penalties for all query-target sequence pairs that pass a user-specified prefilter $Z$-score. We extended the striped SIMD algorithm (Farrar, 2007) by adding a back-tracing procedure to extract the optimal aligments. We keep the three dynamic programming matrices in memory (using 2 B per cell) and trace back from the cell of maximum score by choosing the previous cell whose score is equal to the score in the current cell minus its match score minus gap penalties if applicable. The trace-back stops when it arrives at a cell with score 0. The alignments are multi-core parallelized using OpenMP over the loop of query-target pairs to be aligned.

### 2.3 Clustering module

This module clusters the sequences based on user-specified criteria: sequence identity, $E$-value (to ensure homology) and alignment coverage. A high coverage threshold (0.8 is default in MMseqs) is critical to ensure that all proteins within one cluster have a very similar domain structure. Otherwise, two unrelated families of single-domain proteins composed of domain $A$ or $B$, respectively, could get clustered together with proteins each containing both domains $A$ and $B$, leading to a corrupted sequence cluster.

The greedy set-cover algorithm chooses at each step the sequence with the most remaining neighbours. Neighbours are sequences that satisfy the user-specified clustering criteria. The 'representative' sequence and its neighbours are added to a new cluster, removed from the remaining sequences and the next best representative sequence is picked until all sequences belong to one cluster. MMseqs also offers the greedy incremental clustering algorithm implemented in CD-HIT, USEARCH and kClust.

The time and memory requirements of clustering are typically much lower than for the other two modules. The time and memory complexity are both $O(N_t K)$ where $K$ is the average number of neighbours per sequence.

A stand alone version of our set-cover implementation is available at https://github.com/martin-steinegger/setcover.

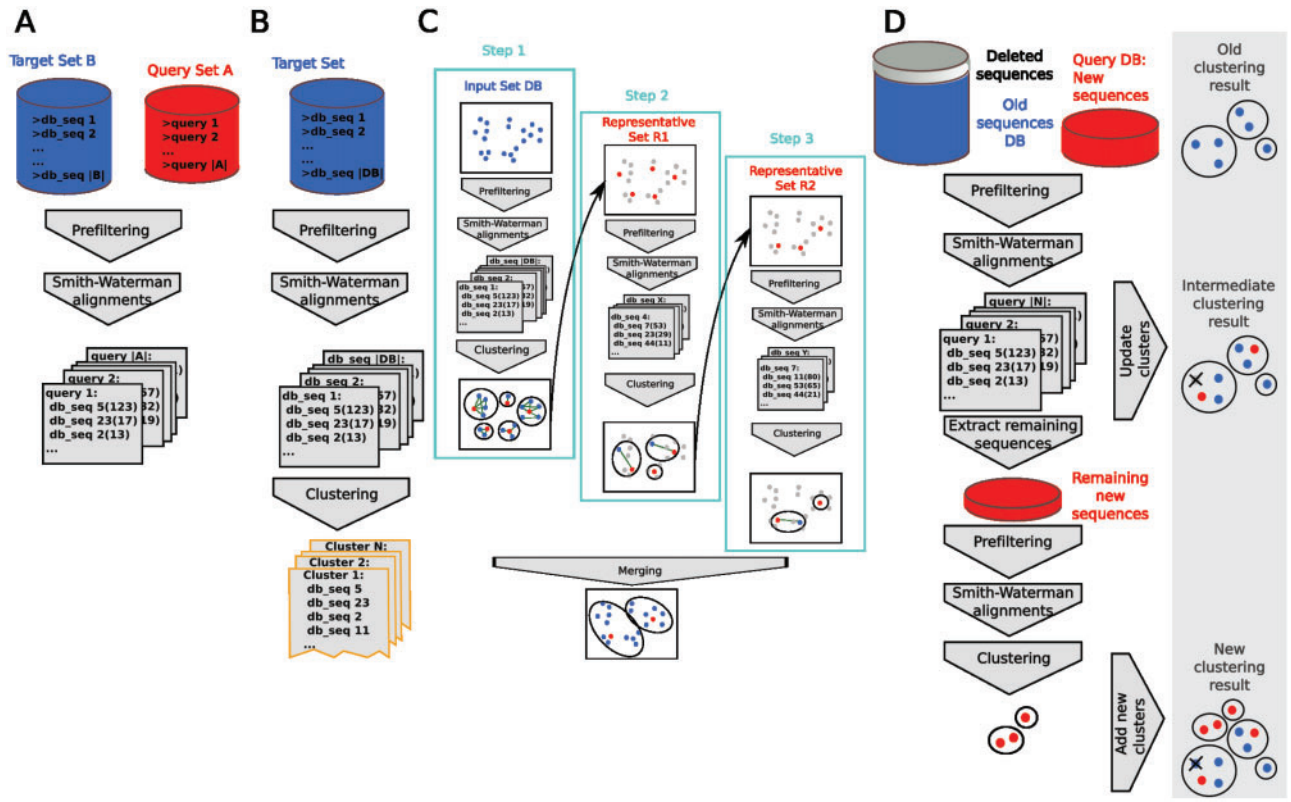


**Fig. 2.** MMseqs workflows are designed to facilitate the most frequent use cases: (**A**) batch sequence searching with one query set through a target set, (**B**) clustering a set of sequences by similarity and other criteria starting from the raw sequence file, (**C**) an efficient *cascaded clustering* procedure for deep clustering and (**D**) updating a clustering of a sequence set by adding new sequences and deleting deprecated sequences

## 2.4 Workflows

### 2.4.1 Sequence search workflow

This workflow searches with each sequence in the query set through all sequences in the target set by running prefilter and alignment modules (Fig. 2A). It outputs an ffindex database with one file of search results per query sequence. (See Supplementary Material for an explanation of the ffindex file format.)

### 2.4.2 Clustering workflow

The clustering workflow clusters sequences starting from a raw sequence file of input sequences (Fig. 2B). It first performs an all-versus-all sequence comparison using the prefilter and alignment modules (with query = target set) and then runs the clustering module on the results.

The clustering workflow also implements a powerful three-step *cascaded clustering* (Fu *et al.*, 2012; Suzek *et al.*, 2007; Fig. 2C). In the first step, the sequences (blue) are clustered using the basic clustering workflow with a fast but low-sensitivity setting (option '–s 1'). The representative sequences from this clustering (R1) are then used as input set (red dots) for a second clustering step with intermediate speed and sensitivity. The resulting representative sequences (R2) are clustered with high sensitivity in the third step. Finally, all sequences in the input set are assigned to one of the clusters of the third clustering step through the intermediate cluster assignments, yielding results formatted as if the clustering had been done in a single step.

Cascaded clustering achieves better sensitivity at comparable speeds. Also, in contrast to single-step clustering it can generate clusters that are much larger than the maximum number of reported sequence matches. This threshold is set to 300 by default to keep the maximum size of the results files manageable.

### 2.4.3 Updating workflow

It has become impractical to frequently update clustered versions of large sequence databases, such as UniRef (Suzek *et al.*, 2007), due to the quadratic scaling of the runtime with the number of sequences. Our updating workflow can update an existing clustering in linear time and with stable cluster identifiers by adding new sequences and deleting deprecated ones that are no longer contained in the new database version.

The updating workflow takes as input a clustering of a previous version of a sequence set ('query DB') and a new version of the sequence set (Fig. 2D). The workflow deletes sequences from the clusters that do not appear in the new sequence set anymore (grey disk, grey crossed-out dot) and compares the newly added sequences (black) to the sequences in the current clusters (blue) using the prefilter and alignment modules. A new sequence is added to the cluster with the most similar representative sequence that satisfies all specified criteria (e.g. *E*-value, alignment coverage, similarity). Those new sequences that do not get recruited to any existing cluster are then clustered amongst themselves and the resulting clusters are added to the clustering.

# 3 Results

## 3.1 Sensitivity and speed of protein searches

### 3.1.1 Benchmark dataset

We downloaded the sequences of the SCOP/ASTRAL database version 1.75 (Murzin *et al.*, 1995) filtered to 25% maximum pairwise sequence identity (Chandonia *et al.*, 2004) ('SCOP25') and removed sequences from the nonstandard class e, folds b.67–b.70 and those with inserted domains. Taking each of the 7616 sequences in this set as a query, we searched for homologous sequences in the uniprot20 database (version 03/2013) using a single search iteration of HHblits (Remmert *et al.*, 2012) with default parameters (*E*-value threshold 0.001). The resulting multiple sequence alignments of the query with its matched sequences were filtered using HHblits options '–qid 30 –id 80 –diff 50', which ensured a minimum sequence identity of 30% to the query, low redundancy, and a maximum of 50 sequences per query. The sequences passing the filtering were labeled with the same SCOP family as the query sequence and pooled into a set of 283406 sequences ('scop25db set'). We created a query fragment set by sampling one randomly chosen fragment of 50 residues length from each of the 7616 sequences in SCOP25.

### 3.1.2 Benchmarked tools

We compared SWIPE (Rognes, 2011), an exact, vectorized implementation of Smith-Waterman alignment, gapped BLAST (Altschul *et al.*, 1997), MMseqs with high and low sensitivity setting, '–s 7' (MMseqs-sens) and '–s 4' (MMseqs-fast), respectively, DIAMOND and DIAMOND-sens (Buchfink *et al.*, 2015), UBLAST (Edgar, 2010) and RAPsearch2 (Zhao *et al.*, 2012). For the ROC5 analysis, we needed to encourage the tools to report at least 5 false positives. We therefore set the *E*-value threshold to 10 for all tools, and we increased the maximum number of reported matches to 1000 for all tools except UBLAST, since UBLAST does not have such a limit. The detailed command line options are listed in the Supplementary Material.

### 3.1.3 Speed measurements

Since there is a trade-off between speed and sensitivity, measuring the speed is important to get the complete picture. Because the size of our scop25db set is too small to extrapolate to the speeds for searching large sequence sets such as UniProt, we measured the speed by searching with the full-length and fragments query sets through the UniProtKB with 54.79 M sequences. We specified 16 cores for all tools and measured the time to build the index structure of the database and the times to complete the search with the 7616 query sequences (Table 1). The searches were run on all 16 cores of a server with two 2.7 GHz Intel Xeon E5-2680 CPUs and 128 GB RAM.

### 3.1.4 Searching with short peptide reads

First, we wanted to assess the ability of fast sequence search tools to find nontrivial homologous matches for short peptide fragments of

**Table 1.** Times in minutes for building the database index, for searching with 7616 fragments of length 50 through the UniProtKB (55 M sequences) and searching with 7616 full-length SCOP sequences through UniProtKB

| Search times | Build db index | Search with SCOP25 | Search with fragments |
|---|---|---|---|
| SWIPE | 36 min | 3214 min (1.7) | 1487 min (3) |
| BLAST | 36 min | 5712 min (1) | 4815 min (1) |
| MMseqs-sens | 77 min | 11 min (520) | 5 min (960) |
| MMseqs-fast | 77 min | 6 min (950) | 3 min (1600) |
| DIAMOND-sens | 15 min | 59 min (100) | 49 min (100) |
| DIAMOND | 15 min | 19 min (300) | 16 min (300) |
| UBLAST | 67 min | 46 min (120) | 19 min (250) |
| RAPsearch | 131 min | 96 min (60) | 66 min (70) |

Values in parenthesis give the speed-up relative to BLAST.

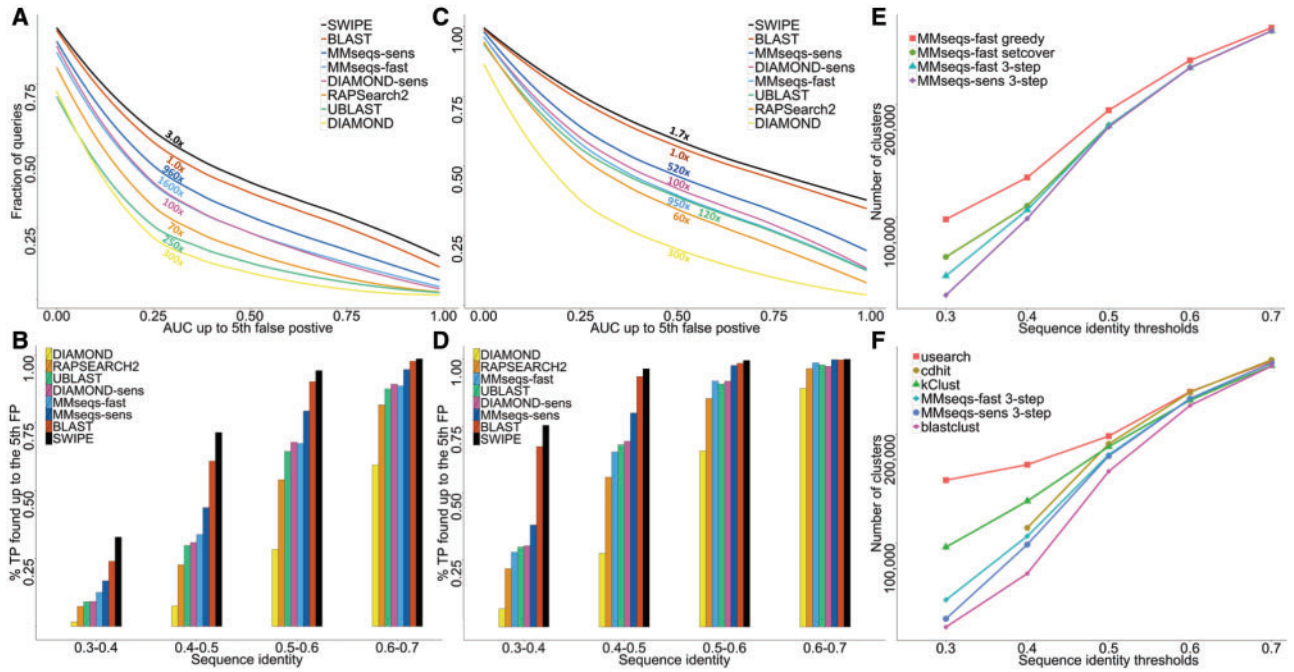


**Fig. 3.** Sensitivity of sequence search tools and clustering performance. (**A**) AUC5 analysis for short peptide queries: Each of 7616 query fragments of length 50 sampled from SCOP25 was searched against the 283 406 sequences of the scop25db set and the area under the ROC curve up to the fifth false positive match (AUC5) was computed. A true positive (TP) match is from the same SCOP family, a false positive (FP) match from a different SCOP fold. The plot shows the cumulative distribution of AUC5 scores for the 7616 queries. The numbers in the legend indicate the search speed relative to BLAST. (**B**) Fraction of true positives found for sequence identity bins [0.3,0.4[, [0.4,0.5[, [0.5,0.6[ and [0.6,0.7[. (**C, D**) Same as A and B, respectively, but using the full length sequences in SCOP25 as queries. Numbers of clusters for various versions of MMseqs (**E**) and other tools (**F**) obtained by clustering the scop25db set with 291 022 sequences

length 50, a length that is typical of hypothetical protein fragments derived from Illumina short read sequences of 150–200 nucleotides.

We created a query fragment set by sampling one random fragment of 50 residues length from each of the scop25 sequences, searched with each query fragment the scop25 set and analyzed the results using a standard ROC5 analysis (Söding and Remmert, 2011): Each matched sequence that came from the same SCOP family as the query fragment was considered a true positive match, each match from a different SCOP fold was considered a false positive, all other matches were ignored. For each query fragment, the area under the curve (AUC) of the receiver operating characteristic (ROC) curve up to the fifth false positive match (AUC5) was calculated (e.g. when all true positives in the scop25db set are found before the first false positive this yields an AUC5 of 1.0). The cumulative distribution of the 7616 AUC5 values in Figure 3A reflects the sensitivity of a tool, e.g. the area under this curve is the average AUC5 over all queries.

SWIPE is more sensitive than BLAST and both are substantially more sensitive than the other tools. MMseqs-sens is by far the most sensitive of the fast tools even though it is 500 time faster than BLAST while the DIAMOND, UBLAST and RAPsearch are only 300, 124 and 59 times faster than BLAST. MMseqs-sens is, somewhat surprisingly, about 12% more sensitive than MMseqs-fast while only being twice slower. The strongest differences are observed for the most difficult cases, as is evident from the fraction of true positive pairs found before the fifth false positive match, plotted for different sequence identity bins (as determined by SWIPE) (Fig. 3B). MMseqs-sens is 8 times faster than UBLAST and 16 times faster than RAPsearch2 but finds 22% more homologs than UBLAST and 15% more than RAPsearch2. MMseqs-sens detects 44% more TP than DIAMOND while being 1.7 times faster.

#### 3.1.5 Searching with full-length SCOP25 sequences

We then repeated the same analysis as before using the full-length sequences in SCOP25 (average length = 166 residues). Figure 3C shows the results of the AUC5 analysis for this query set. All tools achieve better performance, since the longer query sequence contain more information to link them to their homologs. The tools' performance relative to each other is similar as before, although the performance gap between UBLAST and MMseqs-fast has closed. Again, the increased sensitivity of MMseqs-sens over UBLAST and RAPsearch2 is most apparent at low sequence identities (Fig. 3D).

### 3.2 Clustering performance

We used the scop25db set containing 283 406 sequences together with the SCOP25 set of 7616 sequences described above to test the ability of clustering tools to cluster similar sequences together. These single-domain sequences are on average about half as long as full-length sequences. However, since we demand the alignments to cover at least 80% of the longer sequence, the problem of non-transitivity that one faces when clustering multi-domain sequences is largely precluded. We therefore expect this dataset to yield results approximately comparable to those we would obtain when clustering multi-domain sequences.

Figure 3E compares three variants of MMseqs clustering with each other: simple one-step clustering with the greedy algorithm also used by CD-HIT and USEARCH ('MMseqs greedy'), one-step clustering using the greedy set-cover algorithm ('MMseqs set cover'), three-step cascaded clustering using the greedy set-cover algorithm at each step ('MMseqs 3-step') and three-step cascaded clustering with high sensitivity ('–s 7' instead of '–s 4', 'MMseqs-sens 3-step'). We performed clustering runs with five different minimum sequence identity thresholds, 0.3, 0.4, 0.5, 0.6, 0.7, and

compared the performance reflected by the number of clusters found at the given threshold.

Clearly, the greedy set-cover algorithm performs much better than the simple greedy algorithm, even though its speed is comparable. Not surprisingly, the more sensitive sequence comparisons in MMseqs-sens 3-step lead to deeper clustering in comparison with MMseqs 3-step. The 3-step cascaded clustering improves over 1-step clustering by a remarkable margin both in terms of sensitivity and speed. The reason is that reducing the number of sequences from, say, $N_0$ to $N_1$ speeds up the following clustering step by $(N_0/N_1)^2$. In other words, in single-step clustering all true-positive sequence pairs are detected at the maximum level of sensitivity, which is costly, whereas in cascaded clustering most sequence pairs are detected at a lower and faster sensitivity level.

We compared MMseqs 3-step and MMseqs-sens 3-step with popular tools for clustering protein sequence sets: BLASTclust from the BLAST NCBI package (Altschul *et al.*, 1990), CD-HIT, (Fu *et al.*, 2012), kClust (Hauser *et al.*, 2013) and USEARCH (Edgar, 2010; Fig. 3F). Since all tested tools compute either an exact Smith-Waterman local alignment or a banded version, their E-values are either worse or very similar to the E-value for the best Smith-Waterman alignment. For this reason, all clustering tools produce clusterings with a similarly low number of false positive pairs, i.e. non-homologous sequence pairs within the same cluster (see Supplementary Table 1). We can therefore assess the sensitivity of the clustering tools through the number of clusters they produce at a given maximum sequence identity per cluster and by the speed of clustering. A speed comparison on the clustered dataset does not make sense since it is too small for a meaningful speed benchmark. The results on clustering quality therefore have to be viewed in the context of the clustering speeds measured on the full UniProt database (Supplementary Table S1) For all tools a minimum coverage threshold of the longer sequence of 0.8 was used, and all tools except the unparallized kClust and USEARCH were told to use 16 cores. (See Supplementary Material for the command-line options.)

All clustering tools except CD-HIT are faster than BLASTclust by a factor of 1000 or more at all clustering thresholds. For high clustering thresholds, the tools achieve similar sensitivity, as it is simple to find the pairs with high sequence similarities. For a low threshold of 0.3, differences become quite dramatic. Usearch produces 3.5 times more clusters than MMseqs 3-step while running at similar speed, MMseqs-sens 3-step and BLASTclust beat USEARCH in clustering depth by a factor 7. Remarkably, MMseqs-sens 3-step reaches sensitivities similar to BLAST over the entire range of thresholds despite being hundreds of times faster (See Supplementary Table. 2).

To gain a more detailed view of the clustering results of the various tools, Supplementary Figure S3 shows the cumulative size distributions of clusters for threshold of 0.3, corresponding to the leftmost point in Figure 3F. These distributions quite closely reflect the different performances of the underlying sequence similarity searches.

To test the updating workflow in Figure 2D, we randomly divided the scop25db sequence set into 10 equally sized parts, clustered the first part by cascaded clustering and then successively updated it using the second, third etc. up to the tenth part of the sequence set. Supplementary Figure S4 compares the size distribution of the resulting clusters with the size distribution obtained by applying cascaded clustering with default parameters to scop25db. The ten-step updating resulted in slightly fewer and larger clusters.

Finally, we measured the speed and number of clusters obtained when clustering the UniProt database with 54 790 250 sequences down to various sequence identities. We clustered UniProt with MMseqs and USEARCH, the only two tools that are able to cluster such a large database down to the sequence identities of 50% and below. USEARCH requires 11 days and 2 hours for the clustering and produces 9 822 910 clusters, i. e. an average of 5.5 sequences per cluster. MMseqs requires 8 days and 17 hours for the clustering and produces 6 374 156 clusters, i. e. an average of 8.5 sequences per cluster. From an estimate of the runtime of BLASTclust, this is 2000 times faster than BLASTclust (see Supplementary Material Section 2.3 and Table S1 for full results).

## 4 Discussion and outlook

The core of MMseqs is its prefiltering algorithm, to which it owes its favourable combination of high speed and sensitivity. In contrast to most fast search tools, MMseqs does not follow the seed-and-extend paradigm. Instead of depending on a local high similarity, MMseqs' prefilter aggregates evidence for the homology of sequence pairs over their entire length, explaining its success. But this algorithm also makes the prefilter inherently less sensitive to detect short local similarities in relatively long sequences. Since most pairwise alignments above 30% sequence identity will be homologous over most of their length, this is not a severe limitation yet.

In the future, we have plans to improve the sensitivity of the prefilter enough to reach sensitivities of BLAST at several hundred times their speed. This will only be possible with a prefiltering algorithm that scores alignment similarities in a more local way and by eliminating the random memory accesses in the innermost loop. We are also working on extending MMseqs to profile searches and to nucleotide sequence comparisons.

We hope that MMseqs will be able to facilitate and improve the analysis of large sequence sets as produced by massive genome sequencing and metagenomics experiments.

## Acknowledgement

We thank Milot Mirdita for discussions on efficient implementations and for proof reading.

## Funding

We gratefully acknowledge financial support from the Deutsche Telekom-Stiftung to MH, the Bavarian Center for Molecular Biosystems (BioSysNet), the German Research Foundation (DFG grants GRK1721, SFB646) and the German Federal Ministry of Education and Research (BMBF) within the frameworks of e:Med and e:Bio (grants e:AtheroSysMed 01ZX1313D and SysCore 0316176A) to J.S.

*Conflict of Interest:* none declared.

## References

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Altschul,S.F. *et al.* (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

Apweiler,R. *et al.* (2004) UniProt: the Universal Protein knowledgebase. *Nucleic Acids Res.*, **32**, D115–D119.

Bairoch,A. *et al.* (2005) The universal protein resource (uniprot). *Nucleic Acids Res.*, **33**, D154–D159.

Buchfink,B. *et al.* (2015) Fast and sensitive protein alignment using diamond. *Nat. Methods*, **12**, 59–60.

Chandonia,J.M. *et al.* (2004) The astral compendium in 2004. *Nucleic Acids Res.*, **32**, D189–D192.

Chubb,D. *et al.* (2010) Sequencing delivers diminishing returns for homology detection: implications for mapping the protein universe. *Bioinformatics*, **26**, 2664–2671.

Edgar,R.C. (2010) Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, **26**, 2460–2461.

Farrar,M. (2007) Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, **23**, 156–161.

Fu,L. *et al.* (2012) CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, **28**, 3150–3152.

Hauser,M. *et al.* (2013) kClust: fast and sensitive clustering of large protein sequence databases. *BMC Bioinformatics*, **14**, 248+.

Hauswedell,H. *et al.* (2014) Lambda: the local aligner for massive biological data. *Bioinformatics*, **30**, i349–i355.

Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U. S. A.*, **89**, 10915–10919.

Human Microbiome Project Consortium (2012) Structure, function and diversity of the healthy human microbiome. *Nature*, **486**, 207–214.

Huson,D.H. and Xie,C. (2014) A poor man's BLASTX–high-throughput metagenomic protein database search using PAUDA. *Bioinformatics*, **30**, 38–39.

Kanehisa,M. and Goto,S. (2000) Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, **28**, 27–30.

Kaznadzey,A. *et al.* (2013) PSimScan: algorithm and utility for fast protein similarity search. *PLoS One*, **8**, e58505.

Li,W. *et al.* (2002) Sequence clustering strategies improve remote homology recognitions while reducing search times. *Protein Eng.*, **15**, 643–649.

Murzin,A.G. *et al.* (1995) Scop: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, **247**, 536–540.

Park,J. *et al.* (2000) RSDB: representative protein sequence databases have high information content. *Bioinformatics*, **16**, 458–464.

Remmert,M. *et al.* (2012) HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nat. Methods*, **9**, 173–175.

Rognes,T. (2011) Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics*, **12**, 221 +.

Segata,N. *et al.* (2012) Metagenomic microbial community profiling using unique clade-specific marker genes. *Nat. Methods*, **9**, 811–814.

Söding,J. and Remmert,M. (2011) Protein sequence comparison and fold recognition: progress and good-practice benchmarking. *Curr. Opin. Struct. Biol.*, **21**, 404–411.

Sunagawa,S. *et al.* (2015) Structure and function of the global ocean microbiome. *Science*, **348**, 1261359–1–9.

Suzek,B. *et al.* (2007) UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics*, **23**, 1282–1288.

Tan,J. *et al.* (2012) Tachyon search speeds up retrieval of similar sequences by several orders of magnitude. *Bioinformatics*, **28**, 1645–1646.

Zhao,Y. *et al.* (2012) RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics*, **28**, 125–126.