Sensitive protein sequence searching for the analysis of massive data sets

Martin Steinegger[1,2] **and Johannes Söding**[1,*]

[1]Quantitative and Computational Biology group, Max-Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077 Göttingen, Germany
[2]Department for Bioinformatics and Computational Biology, Technische Universität München, 85748 Garching, Germany

Sequencing costs have dropped much faster than Moore's law in the past decade, and sensitive sequence searching has become the main bottleneck in the analysis of large metagenomic datasets. We developed the parallelized, open-source software MMseqs2 (`mmseqs.org`), which improves on current search tools over the full range of speed-sensitivity trade-off, achieving sensitivities better than PSI-BLAST at > 400 times its speed. MMseqs2 offers great potential to better exploit large-scale (meta)genomic data.

Sequencing costs have decreased 10^4-fold since 2007, outpacing the drop in computing costs by three orders of magnitude. As a result, many large-scale metagenomic projects with applications in medical, biotechnological, microbiological, and agricultural research are being performed, each producing terabytes of sequences[1–4]. A central step in the computational analysis is the annotation of open reading frames by searching for similar sequences in the databases from which to infer their functions. In metagenomics, computational costs now dominate sequencing costs [5–7] and protein searches typically consume $> 90\%$ of computational resources[7], even though the sensitive but slow BLAST [8] has mostly been replaced by much faster search tools[9–12]. But the gains in speed are paid by lowered sensitivity. Because many species found in metagenomics and metatranscriptomics studies are not closely related to any organism with a well-annotated genome, the fraction of unannotatable sequences is often as high as 65% to 90% [2, 13], and the widening gap between sequencing and computational costs quickly aggravates this problem.

To address this challenge, we developed the open-source software suite MMseqs2. Compared to its predecessor MMseqs[14], it is much more sensitive, supports iterative profile-to-sequence and sequence-to-profile searches and offers much enhanced functionality (**Supplementary Table S I**).

MMseqs2 searching is composed of three stages (**Fig. 1a**): a short word ("k-mer") match stage, vectorized ungapped alignment, and gapped (Smith-Waterman) alignment. The first stage is crucial for the improved performance. For a given query sequence, it finds all target sequences that have two consecutive inexact k-mer matches on the same diagonal (**Fig. 1b**). Consecutive k-mer matches often lie on the same diagonal for homologous sequences (if no alignment gap occurs between them) but are unlikely to do so by chance. Whereas most fast tools detect only *exact* k-mer matches[9–12], MMseqs2, like MMseqs and BLAST, finds *inexact* k-mer matches between *similar* k-mers. This inexact matching allows MMseqs2 to use a large word size $k = 7$ without loosing sensitivity, by generating a large number of similar k-mers, ~ 600 to 60 000 per query k-mer depending on the similarity setting (**Fig. 1b**, orange frame). Importantly, its innermost loop 4 needs only a few CPU clock cycles per k-mer match using a trick to eliminate random memory access (last line in magenta frame, **Supplementary Fig. S1**).

MMseqs2 is parallelized on three levels: time-critical parts are manually vectorized, queries can be distributed to multiple cores, and the target database can be split into chunks distributed to multiple servers. Because MMseqs2 needs no random memory access in its innermost loop, its runtime scales almost inversely with the number of cores used (**Supplementary Fig. S2**).

MMseqs2 requires 13.4 GB plus 7 B per amino acid to store the database in memory, or 80 GB for 30.3 M sequences of length 342. Large databases can be searched with limited main memory by splitting the database among servers, at very moderate loss of speed (**Supplementary Fig. S3**).

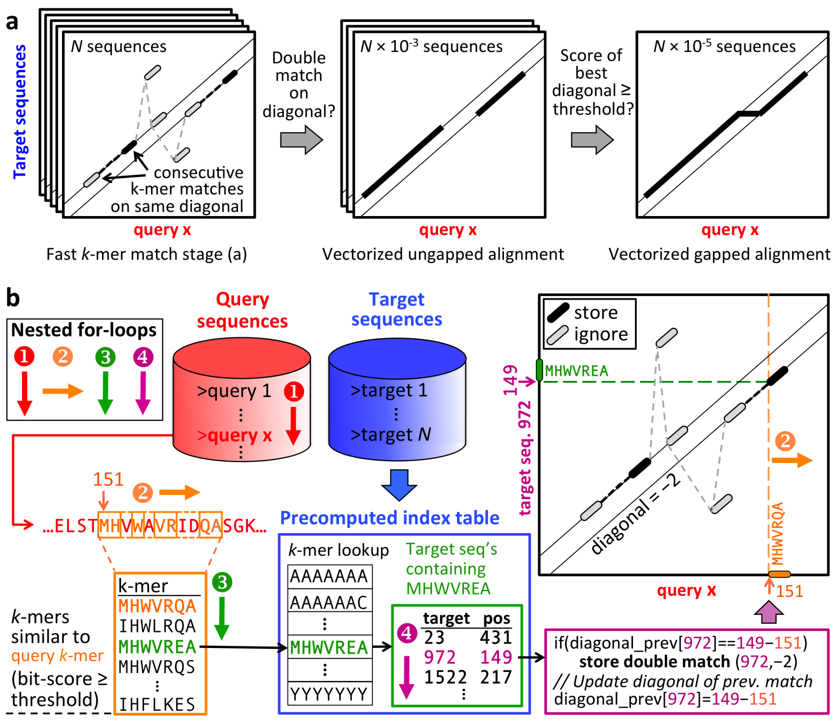
We developed a benchmark with full-length sequences containing disordered, low-complexity and repeat regions, because these regions are known to cause false-positive matches, particularly in iterative profile searches. We annotated UniProt sequences with structural domain annotations from SCOP [15], 6370 of which were designated as query sequences and 3.4 M as database sequences. We also added 27 M reversed UniProt sequences, thereby preserving low complexity and repeat structure [16]. The unmatched parts of query sequences were scrambled in a way that conserved the local amino acid composition. A benchmark using only unscrambled sequences gives similar results (**Supplementary Figs. S4, S5, S6, S7**). We defined true positive matches to have annotated SCOP domains from the same SCOP family, false positives match a reversed sequence or a sequence with a SCOP domain from a different fold. Other cases are ignored.

Figure 2a shows the cumulative distribution of search sensitivities. Sensitivity for a single search is measured by the area under the curve (AUC) before the first false positive match, i.e., the fraction of true positive matches found with better E-value than the first false positive match. MMseqs2-sensitive reaches BLAST's sensitivity while being 36 times faster. Interestingly, MMseqs2 is as sensitive as the exact Smith-Waterman aligner SWIPE[17], compensating some unavoidable loss of sensitivity due to its heuristic prefilters by effectively suppressing false positive matches between locally biased segments (**Fig. 2d, Supplementary Fig. S4**). This is achieved by correcting the scores of regions with biased amino acid composition or repeats, masking such regions in the k-mer index using TANTAN [18], and reducing homologous overextension of alignments with a small negative score offset (**Fig. 2d, Supplementary Fig. S7**). All tools except MMseqs2 and LAST report quite inaccurate (i.e. too optimistic) E-values (**Supplementary Fig. S8**).

In a comparison of AUC sensitivity and speed (**Fig. 2b**), MMseqs2 with four sensitivity settings (red) shows the best combination of speed and sensitivity over the entire range of sensitivities. Similar results were obtained with a benchmark using unscrambled or single-domain query sequences (**Supplementary Figs. S4, S5, S6, S7, S9, S10**).

Searches with sequence profiles are generally much more sensitive than simple sequence searches, because profiles contain detailed, family-specific preferences for each amino acid at each position. We compared MMseqs2 to PSI-BLAST (**Fig. 2b**) us-

Figure 1. **MMseqs2 searching in a nutshell.** (**a**) Three increasingly sensitive search stages find similar sequences in the target database. (**b**) The short word ("k-mer") match stage detects consecutive inexact k-mer matches occurring on the same diagonal. The diagonal of a k-mer match is the difference between the positions of the two similar k-mers 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-mers is generated (orange frame). The similarity threshold determines the list length and sets the trade-off between speed and sensitivity. 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).


a
Target sequences
N sequences
consecutive k-mer matches on same diagonal
query x
Fast k-mer match stage (a)
Double match on diagonal?
N × 10^-3 sequences
Vectorized ungapped alignment
Score of best diagonal ≥ threshold?
N × 10^-5 sequences
Vectorized gapped alignment
b
Nested for-loops
Query sequences
>query 1
>query x
Target sequences
>target 1
>target N
151
...ELSTMHVWAVRIDQASGK...
k-mers similar to query k-mer (bit-score ≥ threshold)
k-mer
MHWVRQA
IHWLRQA
MHWVREA
MHWVRQS
IHFLKES
Precomputed index table
k-mer lookup
AAAAAAA
AAAAAAC
MHWVREA
YYYYYYY
Target seq's containing MHWVREA
target pos
23 431
972 149
1522 217
if(diagonal_prev[972]==149−151)
store double match (972,−2)
// Update diagonal of prev. match
diagonal_prev[972]=149−151
store
ignore
MHWVREA
149
target seq. 972
diagonal = −2
MHWVRQA
query x
151

ing two to four iterations of profile searches through the target database. As expected, MMseqs2 profile searches are much faster and more sensitive than BLAST sequence searches. But MMseqs2 is also considerably more sensitive than PSI-BLAST, despite being 433 times faster at 3 iterations. This is partly owed to its effective suppression of high-scoring false positives and more accurate E-values (**Fig. 2d, Supplementary Fig. S7**).

The MMseqs2 suite offers workflows for various standard use cases of sequence and profile searching and clustering of huge sequence datasets and includes many utility scripts. We illustrate its power with three example applications.

In the first example, we tested MMseqs2 for annotating proteins in the Ocean Microbiome Reference Gene Catalog (OM-RGC) [1]. The speed and quality bottleneck is the search through the eggNOGv3 database [19]. The BLAST search with E-value cutoff 0.01 produced matches for 67% of the 40.2 M OM-RGC genes [1]. We replaced BLAST with three MMseqs2 searches of increasing sensitivity (**Supplementary Fig. S11**). The first MMseqs2 search in fast mode detected matches for 59.3% of genes at $E \leq 0.1$. ($E \leq 0.1$ corresponds to the same false discovery rate as $E \leq 0.01$ in BLAST, **Fig. 2d**). The sequences without matches were searched with default sensitivity, and 17.5% had a significant match. The last search in sensitive search mode found matches for 8.3% of the remaining sequences. In total we obtained at least one match for 69% sequences in OM-RGC, 3% more than BLAST, in 1% of the time (1 520 vs. 162 952 CPU hours; Shini Sunagawa, personal communication).

In the second example, we sought to annotate the remaining 12.3 M unannotated sequences using profile searches. We merged the UniProt database with the OM-RGC sequences and clustered this set with MMseqs2 at 50% sequence identity cut-off. We built a sequence profile for each remaining OM-RGC sequence by searching through this clustered database and accepting all matches with $E \leq 0.001$. With the resulting sequence profiles we searched through eggNOG, and 3.5 M (28.3%) profiles obtained at least one match with $E < 0.1$. This increased the fraction of OM-RGC sequences with significant eggNOG matches to 78% with an additional CPU time of 900 hours. In summary, MMseqs2 matched 78% sequences to eggNOG in only 1.5% of the CPU time that BLAST needed to find matches for 67% of the OM-RGC sequences[1].

In the third example, we annotated a non-redundant set of 1.1 billion hypothetical proteins sequences with Pfam[20] domains. We predicted these sequences of average length 323 in $\sim$ 2200 metagenome/metatranscriptome datasets [21]. Each sequence was searched through 16 479 Pfam31.0 sequence profiles held in 16 GB of memory of a single 2×14-core server using sensitivity setting `-s 5`. **Supplementary Fig. S12** explains the adaptations to the k-mer prefilter and search workflow. The entire search took 8.3 hours, or 0.76 ms per query sequence per core and resulted in 370 M domain annotations with E-values below 0.001. A search of 1100 randomly sampled sequences from the same set with HMMER3[22] through Pfam took 10.6 s per seqeunce per core, almost 14 000 times longer, and resulted in 514 annotations with $E < 0.001$, in comparison to 415 annotations found by MMseq2. A sensitivity setting of `-s 7` brings the number of MMseqs2 annotations to 474 at 4000 times the speed of HMMER3.

In summary, MMseqs2 closes the cost and performance gap between sequencing and computational analysis of protein sequences. Its sizeable gains in speed and sensitivity should open up new possibilities for analysing large data sets or even the entire genomic and metagenomic protein sequence space at once.

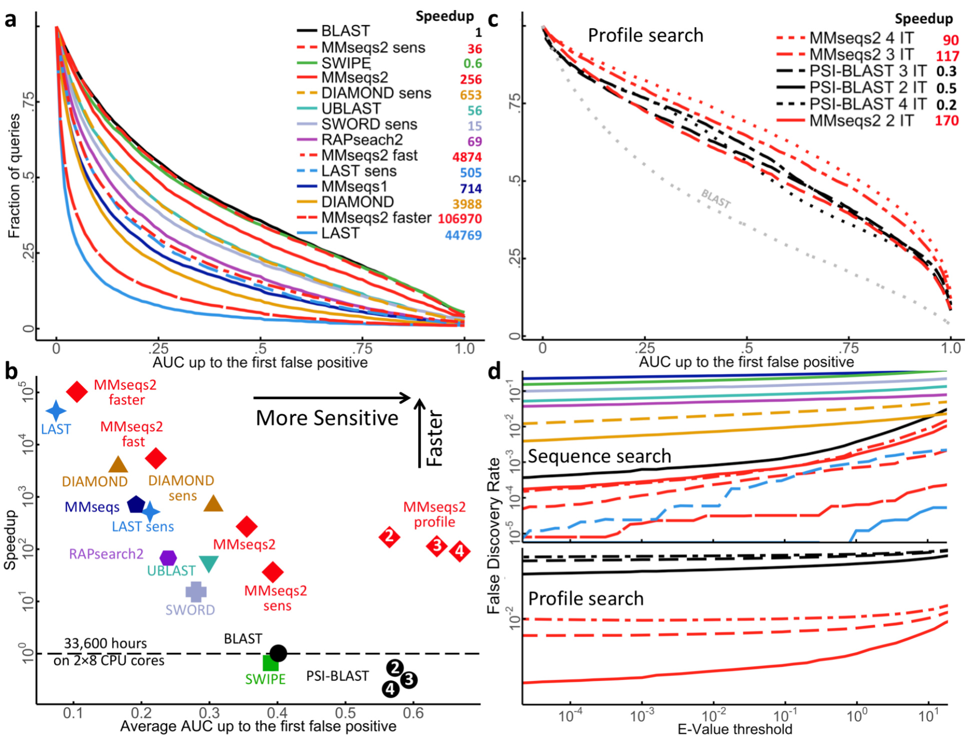


Figure 2. **MMseqs2 pushes the boundaries of sensitivity-speed trade-off.** **a** Cumulative distribution of Area under the curve (AUC) sensitivity for all 6370 searches with UniProt sequences through the database of 30.4 M full-length sequences. Higher curves signify higher sensitivity. Legend: speed-up factors relative to BLAST, measured on a 2×8 core 128 GB RAM server using a 100 times duplicated query set (637 000 sequences). Times to index the database have not been included. MMseqs2 indexing takes 11 minutes for 30.3M sequences of avg. length 342. **b** Average AUC sensitivity versus speed-up factor relative to BLAST. White numbers in plot symbols: number of search iterations. **c** Same analysis as in a, for iterative profile searches. **d** False discovery rates for sequence and profile searches. Colors: as in a (top) and c (bottom).

ACKNOWLEDGMENTS

We thank M. Mirdita, L. van den Driesch and C. Galiez for contributing utilities and workflows, and S. Sunagawa, M. Frith, T. Rattei and our lab for feedback on the manuscript. This work was supported by the European Research Council's Horizon 2020 Framework Programme for Research and Innovation ("Virus-X", project no. 685778) and by the German Federal Ministry for Education and Research (BMBF) (grants e:AtheroSysMed 01ZX1313D, "SysCore" 0316176A).

AUTHOR CONTRIBUTIONS

M.S. developed the software and performed the data analysis. M.S. and J.S. conceived of and designed the algorithms and benchmarks and wrote the manuscript.

COMPETING FINANCIAL INTERESTS

The authors declare no competing financial interests.

[1] Sunagawa, S. *et al.* *Science* **348**, 1261359–1–9 (2015).
[2] Afshinnekoo, E. *et al.* *Cell Systems* **1**, 72–87 (2015).
[3] Howe, A. C. *et al.* *Proc. Natl. Acad. Sci. U.S.A.* **111**, 4904–4909 (2014).
[4] Franzosa, E. A. *et al.* *Nat. Rev. Microbiol.* **13**, 360–372 (2015).
[5] Scholz, M. B. *et al.* *Curr. Opin. Biotechnol.* **23**, 9–15 (2012).
[6] Desai, N. *et al.* *Curr. Opin. Biotechnol.* **23**, 72–76 (2012).
[7] Tang, W. *et al.* Workload characterization for MG-RAST metagenomic data analytics service in the cloud. In *IEEE International Conference on Big Data*, 56–63 (IEEE, 2014).
[8] Altschul, S. F. *et al.* *Nucleic Acids Res.* **25**, 3389–3402 (1997).
[9] Edgar, R. C. *Bioinformatics* **26**, 2460–2461 (2010).
[10] Kiełbasa, S. M. *et al.* *Genome Res.* **21**, 487–493 (2011).
[11] Zhao, Y. *et al.* *Bioinformatics* **28**, 125–126 (2012).
[12] Buchfink, B. *et al.* *Nature Methods* **12**, 59–60 (2015).
[13] Hurwitz, B. L. & Sullivan, M. B. *PLoS One* **8**, e57355 (2013).
[14] Hauser, M. *et al.* *Bioinformatics* **32**, 1323–1330 (2016).
[15] Murzin, A. G. *et al.* *J. Mol. Biol.* **247**, 536 – 540 (1995).
[16] Karplus, K. *et al.* *Bioinformatics* **14**, 846–856 (1998).
[17] Rognes, T. *BMC Bioinformatics* **12**, 221+ (2011).
[18] Frith, M. C. *et al.* *Nucleic Acids Res.* **36**, 5863–5871 (2008).
[19] Jensen, L. J. *et al.* *Nucleic Acids Res.* **36**, D250–D254 (2008).
[20] Finn, R. D. *et al.* *Nucleic Acids Res.* **44**, D279–D285 (2016).
[21] Steinegger, M. & Söding, J. *bioRxiv* 104034 (2017).
[22] Eddy, S. R. *PLoS Comput. Biol.* **7**, e1002195 (2011).

ONLINE METHODS

Overview. MMseqs2 (Many-against-Many sequence searching) is a software suite to search and cluster huge protein sequence sets. MMseqs2 is open source GPL-licensed software implemented in C++ for Linux and Mac OS. At the core of MMseqs2 is its sequence search module. It searches and aligns a set of query sequences against a set of target sequences. Queries are processed in three consecutive stages of increasing sensitivity and decreasing speed (**Fig. 1a**): (1) the fast k-mer match stage filters out 99.9 % of sequences, (2) the ungapped alignment stage filters out a further 99 %, and (3) the accurate, vectorized Smith-Waterman alignment thus only needs to align $\sim 10^{-5}$ of the target sequences.

MMseqs2 builds on our MMseqs software suite[7], designed for fast sequence clustering and searching of globally alignable sequences. The k-mer match stage of MMseqs2, which is crucial for its improved sensitivity-speed trade-off, has been developed from scratch, profile-to-sequence and sequence-to-profile searching capabilities have been developed, and many other powerful features and utilities have been added (see **Supplemental Table S1** for an overview of differences).

The software is designed to run on multiple cores and servers and exhibits nearly linear scalability. It makes extensive use of single instruction multiple data (SIMD) vector units which are part of modern Intel and AMD CPUs. For older CPUs without AVX2 support, MMseqs2 falls back to SSE4.1 instructions throughout with minimal speed loss.

***k*-mer match stage.** Most fast methods follow a seed-and-extend approach: a fast seed stage searches for short-word ("k-mer") matches which are then extended to a full, gapped alignment. Since the k-mer match stage needs to work on all sequences, it needs to be much faster than the subsequent stages. Its sensitivity is therefore crucial for the overall search sensitivity. In contrast to BLAST and SWORD [20], most fast methods index the database k-mers instead of the query sequences, using hashes or suffix arrays, and a few index both to streamline random memory access during the identification of k-mer matches [2, 8, 12]. To increase the seeds' sensitivity, some methods allow for one or two mismatched positions [8, 11], others employ reduced alphabets [2, 8, 19, 22]. Many use spaced k-mer seeds to reduce spatial clustering of chance matches [2, 8].

Whereas most other tools use only single, exact k-mer matches as seeds, the k-mer match stage of MMseqs2 detects double, consecutive, *similar-k*-mer matches occurring on the *same diagonal* $i-j$. i is position of the k-mer in the query and j is the position of the matching k-mer in the target sequence. This criterion very effectively suppresses chance k-mer matches between nonhomologous sequences as these have a probability of only $\sim 1/(L_{\text{query}} + L_{\text{target}})$ to have coinciding diagonals. In contrast, consecutive k-mer matches between homologous sequences lie on the same diagonal if no alignment insertion or deletion occurred between them. A similar criterion is used in the earlier, two-hit 3-mer seed strategy of BLAST [1]. (The new version reverts to a single-hit strategy but uses 6-mers on a reduced size-15 alphabet instead of 3-mers.[16].)

Query sequences are searched one by one against the target set (**Fig. 1b**, loop 1). For each k-mer starting position in the query (loop 2) we generate a list of all similar k-mers (orange frame) with a Blosum62 similarity above a threshold score. Lower threshold scores (option --k-score <int>) result in higher average numbers of similar k-mers and thereby higher sensitivity and lower speed. The similar k-mers are generated with a linear-time branch-and-bound algorithm[6].

For each k-mer in the list of similar k-mers (loop 3), we obtain from the index table (blue frame) the list of target sequence identifiers `target_ID` and the positions j of the k-mer (green frame). In the innermost loop 4 we go through this list to detect double k-mer matches by comparing the current diagonal $i-j$ with the previously matched diagonal for `target_ID`. If the previous and current diagonals agree, we store the diagonal $i-j$ and `target_ID` as a double match. Below, we describe how we carry out this computation within low-level, fast CPU cache without random memory access.

Minimizing random memory access. Due to the increase in the number of cores per CPU and the stagnation in main memory speeds in the last decade, main memory access is now often the chief bottleneck for compute-intensive applications. Since it is shared between cores, it also severely impairs scalability with the number of cores. It is therefore paramount to minimize random memory accesses.

We want to avoid the random main memory access to read and update the value of `diagonal_prev[target_ID]` in the innermost loop. We therefore merely write `target_ID` and the diagonal $i-j$ serially into an array `matches` for later processing. Because we write linearly into memory and not at random locations, these writes are automatically buffered in low-level cache by the CPU and written to main memory in batches with minimal latency. When all k-mers from the current query have been processed in loop 2, the `matches` array is processed in two steps to find double k-mer matches. In the first step, the entries (`target_ID`, $i-j$) of `matches` are sorted into 2^B arrays (bins) according to the lowest B bits of `target_ID`, just as in radix sort. Reading from `matches` is linear in memory, and writing to the 2^B bins is again automatically buffered by the CPU. In the second step, the 2^B bins are processed one by one. For each k-mer match (`target_ID`, $i-j$), we run the code in the magenta frame of Fig. 1b. But now, the `diagonal_prev` array fits into L1/L2 CPU cache, because it only needs $\sim N/2^B$ entries, where N is the number of target database sequences. To minimize the memory footprint, we store only the lowest 8 bits of each diagonal value in `diagonal_prev`, reducing the amount of memory to $\sim N/2^B$ bytes. For example, in the 256 KB L2 cache of Intel Haswell CPUs we can process a database of up to 256K $\times\, 2^B$ sequences. To match L2 cache size to the database size, MMseqs2 sets $B =$ `ceil`(`log2`(N/`L2_size`)).

Index table generation. For the k-mer match stage we preprocess the target database into an index table. It consists of a pointer array (black frame within blue frame in Fig. 1b) and k-mer entries (green frame in Fig. 1b). For each of the 21^k k-mers (the 21st letter X codes for "any amino acid") a pointer points to the list with target sequences and positions (`target_ID`, j) where this k-mer occurs. Prior to index generation, regions of low amino acid compositional complexity are masked out using TANTAN (see Masking low-complexity regions). Building the index table can be done in multithreaded fashion and does not require any additional memory. To that end, we proceed in two steps: counting of k-mers and filling entries. In the first step each thread counts the k-mers, one sequence at a time. All threads add up their counts using the atomic function `__sync_fetch_and_add`. In the second step, we allocate the appropriately sized array for the k-mer entries. We then assign roughly equal sized k-mer-ranges to every thread and initialise their pointers at which they start filling their part of the entry array. Now each thread processes all sequence but only writes the k-mers of the assigned range linearly into the entry array. Building the index table file for 3×10^7 sequences takes about 11 minutes on 2×8 cores.

Memory requirements. The index table needs 4 + 2 bytes for each entry (`target_ID`, j), and one byte per residue is needed to store the target sequences. For a database of NL residues, we therefore require $NL \times 7$ B. The pointer array needs another $21^k \times 8$ B. The target database set can be split into arbitrary chunk sizes to fit them into memory (see Parallelization).

Ungapped alignment stage. A fast, vectorized algorithm computes the scores of optimal ungapped alignments on the diagonals with double k-mer matches. Since it has a linear time complexity, it is much faster than the Smith-Waterman alignment stage with its quadratic time complexity. The algorithm aligns 32 target sequences in parallel, using the AVX2 vector units of the CPU. To only access memory linearly we precompute for each query sequence a matrix with 32 scores per query residue, containing the 20 amino acid substitution scores for the query residue, a score of -1 for the letter X (any residue), and 11 zero scores for padding. We gather bundles of 32 target sequences with matches on the same diagonal and also preprocess them for fast access: We write the amino acids of position j of the 32 sequences consecutively into block j of 32 bytes, the longest sequence defining the number of blocks. The algorithm moves along the diagonals and iteratively computes the 32 scores of the best alignment ending at query position i in AVX2 register S using $S = \texttt{max}(0, S_{\text{match}} + S_{\text{prev}})$. The substitution scores of the 32 sequences at the current query position i in AVX2 register S_{match} are obtained using the AVX2 (V)PSHUFB instruction, which extracts from the query profile at position i the entries specified by the 32 bytes in block j of the target sequences. The maximum scores along the 32 diagonals are updated using $S_{\max} = \texttt{max}(S_{\max}, S)$. We subtract from $S_{\max}$ the $\log_2$ of the length of the target sequence. Alignments above 15 bits are passed on to the next stage.

Vectorized Smith-Waterman alignment stage. We extended the alignment library of Mengyao et al. [21], which is based on Michael Farrar's stripe-vectorized alignment algorithm [3], by adding support for AVX2 instructions and for sequence profiles. To save time when filtering matches, we only need to compute the score and not the full alignment. We therefore implemented versions that compute only the score and the end position of the alignment, or only start and end position and score.

Amino acid local compositional bias correction. Many regions in proteins, in particular those not forming a stable structure, have a biased amino acid composition that differs considerably from the database average. These regions can produce many spurious k-mer matches and high-scoring alignments with non-homologous sequences of similarly biased amino acid distribution. Therefore, in all three search stages we apply a correction to substitution matrix scores developed for MMseqs[7], assigning lower scores to the matches of amino acids that are overrepresented in the local sequence neighborhood. Query sequence profile scores are corrected in a similar way: The score $S(i, \text{aa})$ for amino acid aa at position i is corrected to $S_{\text{corr}}(i, \text{aa}) = S(i, \text{aa}) - \frac{1}{40} \sum_{j=i-20, j \neq i}^{i+20} S(j, \text{aa}) + \frac{1}{L_{\text{query}}} \sum_{j=1}^{L_{\text{query}}} S(j, \text{aa})$.

Masking low-complexity regions. The query-based amino acid local compositional bias correction proved effective, particularly for sequence-to-sequence searches. However, for iterative profile sequence searches a very low level of false discovery rate is required, as false positive sequences can recruit more false positives in subsequent iterations leading to massively corrupted profiles and search results in these instances. We observed that these cases were mainly caused by biased and low-complexity regions in the *target sequences*. We therefore mask out low-complexity regions in the target sequences during the k-mer matching and the ungapped alignment stage. We use TANTAN[4] with a threshold of 0.9% probability for low complexity.

Iterative profile searching The first iteration of the profile search is a straightforward MMseqs2 sequence-to-sequence search. We then realign all matched sequences with E-values below the specified threshold (option `--E-profile <value>`) with a score offset of -0.1 bits per matched residue pair (added to the scores of the substitution matirx) to avoid "homologous overextentions" of the alignments, a serious problem causing many false positives in iterative profile searches [4, 5]. In all further iterations, we remove previously accepted sequences from the prefilter results and align only the newly found sequence matches. From the search results we construct a simple star multiple sequence alignment (MSA) with the query as the master sequence. We filter the multiple sequence alignment with 90% maximum pairwise sequence identity and pick the 100 most diverse sequences using C++ code adapted from our HH-suite software package[14]. As in HH-suite, we compute position-specific sequence weights[1], which ensure that MSAs with many matched segments that stretch only part of the query sequence, as occurs often for multidomain proteins, are treated appropriately. We add pseudocounts to the amino acid counts of each profile column as described for HHsearch [17].

Profile-to-sequence search mode. To enable searching a target profile database, we made four changes to the search workflow (**Supplementary Fig. S11**): (1) We generate a k-mer index table for the target database by looping over all profiles and all k-mer positions and adding all k-mers to the index that obtain a profile similarity score above the threshold. Lower score thresholds lead to more k-mers and higher sensitivity. (2) We only look for the exact query k-mers in the index table. The former loop 3 is omitted. (3) The ungapped alignment for each matched diagonal is computed between the query sequence and the target profile's consensus sequence, which contains at each column the most frequent amino acid. (4) The previous step produced for each query sequence s a list of matched profiles p with score $S_{sp} > 15$bit. However, the gapped alignment stage can only align profiles with sequences and not vice versa. We therefore transpose the scores S_{sp} in memory and obtain for each profile p all matched sequences, $\{s : S_{ps} > 15\text{bit}\}$, which we pass to the gapped alignment stage. Finally, the results are transposed again to obtain for each query sequence a list of matched profiles.

Algorithmic novelty in MMseqs2. MMseqs2 builds upon many powerful previous ideas in the sequence search field, such as inexact k-mer matching [1], finding two k-mers on the same diagonal [1], or spaced k-mers [12]. In addition to carefully engineering every relevant piece of code for maximum speed, we introduce with MMseqs2 several novel ideas that were crucial to the improved performance: (1) the algorithm to find two consecutive, inexact k-mer matches (**Fig. 1b**); (2) the avoidance of random memory accesses in the innermost loop of the k-mer match stage (**Supplementary Fig. S1**); (3) the use of 7-mers, which is only enabled by the fast generation of similar k-mers ($\sim 60\,000$ per k-mer in sensitive mode); (4) iterative profile-sequence search mode including profile-to-sequence vectorized Smith-Waterman alignment; (5) sequence-to-profile search mode; (6) the introduction of a fast, vectorized ungapped-alignment step (**Fig. 1a**); (7) a fast amino acid compositional bias score correction on the query side that suppresses high-scoring false positives.

Parallelization. Due to the stagnation in CPU clock rates and the increase in the number of cores per CPU, vectorization and parallelisation across multiple cores and servers is of growing importance for highly compute-intensive applications. Besides careful vectorization of time-critical loops, MMseqs2 is efficiently parallelized to run on multiple cores and servers using OpenMP and message passing interface (MPI).

OpenMP threads search query sequences independently against the target database and write their result into separate files. After all queries are processed, the master thread merges all results together.

To parallelize the time-consuming k-mer matching and gapless alignment stages 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. The results from each server are automatically merged. Alternatively, the query sequence set is split into equal-sized chunks and each server searches its query chunk against the entire target set. Splitting the target database is less time-efficient due to the slow, IO-limited merging of results. But it reduces the memory required on each server to $NL \times 7\text{B}/\#\text{chunks} + 21^k \times 8\,\text{B}$ and allows users to search through huge databases on servers with moderate memory sizes. If the number of chunks is larger than the number of servers, chunks will be distributed among servers and processed sequentially. By default, MMseqs2 automatically decides which mode to pick based on the available memory on the master server.

MMseqs2 software suite. The MMseqs2 suite consists of four simple-to-use main tools for standard searching and clustering tasks, 37 utility tools, and four core tools ("expert tools"). The core tool `mmseqs prefilter` runs the first two search stages in Fig. 1a, `mmseqs align` runs the Smith-Waterman alignment stage, and `mmseqs clust` offers various clustering algorithms. The utilities comprise format conversion tools, multiple sequence alignment, sequence profile calculation, open reading frame (ORF) extraction, 6-frame translation, set operations on sequence sets and results, regex-based filters, and statistics tools to analyse results. The main tools are implemented as `bash`-scripted workflows that chain together core tools and utilities, to facilitate their modification and extension and the creation of new workflows.

Design of sensitivity benchmark. Some recent new sequence search tools were only benchmarked against short sequences, using BLAST results as the gold standard [2, 8, 9, 22]. Short matches require fairly high sequence identities to become statistically significant, making BLAST matches of length 50 relatively easy to detect. (For a sequence match to achieve an $E-$value < 0.01 in a search through UniProt requires a raw score of ~ 40 bits, which on 50 aligned residues translates to a sequence identity $\gtrsim 40\%$). Because long-read, third-generation sequencing technologies are becoming widespread, short-read technologies are improving read lengths, and ORFs and putative genes in metagenomics are commonly predicted from assembled contigs, we constructed a benchmark set using full-length queries and database sequences, not just sequences of structured domains as usually done. Including disordered regions, membrane helices, and other low-complexity regions is important since they often give rise to false-positive sequence matches, particularly in iterative sequence searches.

Because we cannot use BLAST or SWIPE[15] as gold standard if we want to compare other tools with them, we use evolutionary relationships that have been determined on the basis of structures as gold standard. SCOP [13] is a database of protein domains of known structure organised by evolutionary relationships.

We defined true positive matches to have annotated SCOP domains from the same SCOP family, false positives match a reversed sequence. In the first benchmark version matches to a sequence with a SCOP domain from a different fold except the beta propellers (which are known to be homologous [18]) are also conside--red false positives. Other cases are ignored. -- The false discovery rate (FDR) For a single search is computed as FDR = FP/(FP + TP), where TP and FP are the numbers of true and false positive matches below a cutoff score in that search. To prevent a few searches with many false positives from dominating the FDR, we computed the FDR for all searches as arithmetic mean over the single-search FDRs.

We measure the sensitivity of search tools using a receiver operating characteristic (ROC) analysis [18]. We search with a large set of query sequences through a database set (see next paragraph) and record for each query the fraction of true positive sequences detected up to the first false positive. This sensitivity is also called area under the curve 1 (AUC1). We then plot the cumulative distribution of AUC1 values, that is, the fraction of query sequences with an AUC1 value larger than the value on the x-axis. The more sensitive a search tools is the higher will its cumulative distribution trace lie. We do not analyse only the best match for every search in order to increase the number of matches and to thereby reduce statistical noise.

Benchmark set. The SCOP/ASTRAL (v. 1.75) database was filtered to 25% maximum pairwise sequence identity (7616 sequences), and we searched with each SCOP sequence through the UniRef50 (06/2015) database, using SWIPE [15] and, for maximum sensitivity, also three iterations of HHblits. To construct the query set, we chose for each of the 7616 SCOP sequences the best matching UniRef50 sequence for the query set if its SWIPE E-value was below 10^{-5}, resulting in 6370 query sequences with 7598 SCOP-annotated domains. In the first version of the benchmark (**Fig. 2**), query sequences were shuffled outside of annotated regions within overlapping windows of size 10. This preserves the local amino acid composition while precluding true positive matches in the shuffled regions. In the second version of the benchmark, query sequences were left unchanged (**Supplementary Fig. S3, S4, S5, S6**).

To construct the target database, we selected all UniRef50 sequences with SWIPE or HHblits $E-$value $< 10^{-5}$ and annotated them with the corresponding SCOP family, resulting in 3 374 007 annotations and a median and average number of sequences per SCOP family of 353 and 2150, respectively. Since the speed measurements are only relevant and quantitative on a database of realistic size, we added the 27 056 274 reversed sequences from a 2012 UniProt release. Again, the reversion preserves the local amino acid composition while ruling out true positive matches [10]. We removed the query sequences from the target database and removed queries with no correct matches in the target database from the query set.

Benchmarking. We evaluated results up to the 4000th match per query (ranked by E-value) and, for tools with an upper limit on the number of reported matches, set this limit via command line option to 4000. The maximum E-value was set to 10 000 to detect at least one false positive and to avoid biases due to slightly different E-value calculations. Because the MMseqs2 prefilter is already very sensitive and returns proper E-values, the Smith-Waterman alignment stage is not needed in the "fast" and "faster" versions. Program versions and calls are found in the **Supplemental Table S2**.

All benchmarks were run on a single server with two Intel Xeon E5-2640v3 CPUs (2×8 cores, 2.6 GHz) and 128GB memory. Run times were measured using the Linux `time` command, with the target database (70 GB, 30.4 M sequences) on local solid state drives. Since some search tools are speed-optimized not only for large target databases but also for large query sets, we duplicated the query set 100 times for the runtime measurements, resulting in 637 000 query sequences. For the slowest tools, SWIPE, BLAST and RAPsearch2, we scaled up the runtime for the original query dataset 100-fold.

Data availability. Parameters and scripts for benchmarking are deposited at `https://bitbucket.org/martin_steinegger/mmseqs-benchmark`.

Code availability. The source code and binaries of the MMseqs2 software suite can be download at `https://mmseqs.org`.

[1] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25(17):3389–3402, Sept. 1997.

[2] B. Buchfink, C. Xie, and D. H. Huson. Fast and sensitive protein alignment using DIAMOND. *Nature Methods*, 12(1):59–60, 2015.

[3] M. Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, Jan. 2007.

[4] M. C. Frith, Y. Park, S. L. Sheetlin, and J. L. Spouge. The whole alignment and nothing but the alignment: the problem of spurious alignment flanks. *Nucleic Acids Res.*, 36(18):5863–5871, 2008.

[5] M. W. Gonzalez and W. R. Pearson. Homologous over-extension: a challenge for iterative similarity searches. *Nucleic acids research*, 38(7):2177–2189, 2010.

[6] M. Hauser, C. E. Mayer, and J. Söding. kclust: fast and sensitive clustering of large protein sequence databases. *BMC Bioinformatics*, 14(1):1–12, 2013.

[7] M. Hauser, M. Steinegger, and J. Söding. MMseqs software suite for fast and deep clustering and searching of large protein sequence sets. *Bioinformatics*, 32(9):1323–1330, 2016.

[8] H. Hauswedell, J. Singer, and K. Reinert. Lambda: the local aligner for massive biological data. *Bioinformatics*, 30(17):i349–i355, 2014.

[9] D. H. Huson and C. Xie. A poor man's BLASTX-high-throughput metagenomic protein database search using PAUDA. *Bioinformatics*, 30(1):38–39, 2014.

[10] K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

[11] J. J. Kent. BLAT–the BLAST-like alignment tool. *Genome Res.*, 12(4):656–664, Apr. 2002.

[12] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[13] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247(4):536 – 540, 1995.

[14] M. Remmert, A. Biegert, A. Hauser, and J. Söding. HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nature Methods*, 9(2):173–175, Feb. 2012.

[15] T. Rognes. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics*, 12(1):221+, June 2011.

[16] S. A. Shiryev, J. S. Papadopoulos, A. A. Schäffer, and R. Agarwala. Improved blast searches using longer words for protein seeding. *Bioinformatics*, 23(21):2949–2951, 2007.

[17] J. Söding. Protein homology detection by HMM–HMM comparison. *Bioinformatics*, 21(7):951–960, 2005.

[18] J. Söding and M. Remmert. Protein sequence comparison and fold recognition: progress and good-practice benchmarking. *Curr. Opin. Struct. Biol.*, 21(3):404–411, 2011.

[19] J. Tan, D. Kuchibhatla, F. L. Sirota, W. A. Sherman, T. Gattermayer, C. Y. Kwoh, F. Eisenhaber, G. Schneider, and S. M. Stroh. Tachyon search speeds up retrieval of similar sequences by several orders of magnitude. *Bioinformatics*, 28(12):1645–1646, June 2012.

[20] R. Vaser, D. Pavlović, M. Korpar, and M. Šikić. Sword-a highly efficient protein database search. *Bioinformatics*, 32(17):i680–i684, 2016.

[21] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth. Ssw library: An simd smith-waterman c/c++ library for use in genomic applications. *PLoS One*, 8(12), 12 2013.

[22] Y. Zhao, H. Tang, and Y. Ye. RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics*, 28(1):125–126, Jan. 2012.