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 (meta)genomic datasets. While previous methods sacrificed sensitivity for speed gains, the parallelized, open-source software MMseqs2 overcomes this trade-off: In three-iteration profile searches it reaches 50% higher sensitivity than BLAST at 83-fold speed and the same sensitivity as PSI-BLAST at 270 times its speed. MMseqs2 therefore offers great potential to increase the fraction of annotatable (meta)genomic sequences.

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 projects each producing terabytes of sequences are being performed, such as the genome 10K project [11] and metagenomic and metatranscriptomic studies with applications in medical, biotechnological, microbiological, and agricultural research [1, 3, 8, 13, 25].

A central step in the computational analysis is the annotation of open reading frames (ORFs) by searching for similar, annotated proteins from which to infer their functions. In metagenomics, computational costs now dominate sequencing costs [5, 22, 27] and protein searches typically consume $> 90\%$ of computational resources [27], even though the gold standard BLAST tool [2] has mostly been replaced by faster and less sensitive search tools such as BLAT[17], UBLAST[6], LAST[18], RAPSearch2[30], and DIAMOND[4].

Like BLAST, 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. In contrast to BLAST and SWORD [28], 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 [4, 12]. To increase the seeds' sensitivity, some methods allow for one or two mismatched positions [12, 17], others employ reduced alphabets [4, 12, 26, 30]. Many use spaced k-mer seeds to avoid strongly overlapping, multiple matches [4, 12].

While being tens to thousands of times faster than BLAST, none of the faster methods even approaches its 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% [1, 14], and the widening gap between sequencing and computational costs quickly aggravates this problem.

To address this challenge, we developed MMseqs2 (many against many sequence searching 2) (https://github.com/soedinglab/mmseqs2). MMseqs2 builds on our software MMseqs [10], designed for fast sequence clustering and searching of globally alignable sequences (see Supplemental Table S1 for a list of differences). MMseqs2 shows a drastically improved sensitivity and speed in sequence searching, enabled by a novel k-mer matching algorithm designed to find *local* sequence similarities, a gapless alignment stage, better suppression of false positive matches, and better scalability across multiple CPU cores and servers. Also, dozens of utilities have been added to the MMseqs2 toolset. Most importantly, MMseqs2 is the first fast method to support iterative sequence profile searches and to reach sensitivities much above BLAST.

MMseqs2 searching proceeds in three stages (**Fig. 1a**): a k-mer match stage, a vectorized ungapped alignment stage, and accurate, stripe-vectorized [7, 29] Smith-Waterman alignment. All stages correct scores for locally biased amino acid composition. Final results are ranked by E-value.

The crucial first stage (**Fig. 1b**) finds all target sequences that have two consecutive k-mer matches on the same diagonal. The diagonal of a k-mer match is the positional offset $i-j$ between the current position in the query i and the position j of the similar k-mer in the target sequence ("pos" in green frame). If the current k-mer match in sequence `target_ID` occurs on the same diagonal $i-j$ as the previous one stored in `diagonal_prev[target_ID]`, we found a double k-mer match. These double matches are passed on to the next stage, which computes on these diagonals the ungapped alignment score.

The efficiency of the first stage is owed to the following: (1) Like BLAST and MMseqs, we find *similar*-k-mer matches with a substitution matrix similarity score above a specifiable threshold. In contrast to BLAST, MMseqs2 uses a large word size $k=7$, resulting in an optimal sensitivity-specificity trade-off. To maintain high sensitivity of the k-mer matches, we generate between 600 and 55 000 similar k-mers (**Fig. 1b**, orange frame) per query k-mer on average, depending on the score threshold for the sensitivity setting (faster, fast, default, sensitive). This consumes less than 2% CPU time. (2) Consecutive k-mer matches on the same diagonal have an even more favourable sensitivity-specificity trade-off because consecutive k-mer matches are unlikely to occur on the same diagonal by chance, whereas for homologous sequences they will 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 [2]. (A new version reverts to a single-hit strategy but uses 6-mers on a reduced size-15 alphabet instead of 3-mers.[23]) (3) Importantly, k-mer matches are very efficiently processed in the innermost loop 4 within only a few CPU clock cycles per k-mer match, by eliminating the random memory access for storing the diagonal of the current match (last line in magenta box of **Fig. 1b**, **Supplementary Fig. S1**). (4) Since all cores share access to main memory, non-linear memory access is usually the bottleneck for efficient multi-core parallelization. By avoiding random access in loop 4, run times scale almost inversely with the number of cores (**Supplementary Fig. S2**).

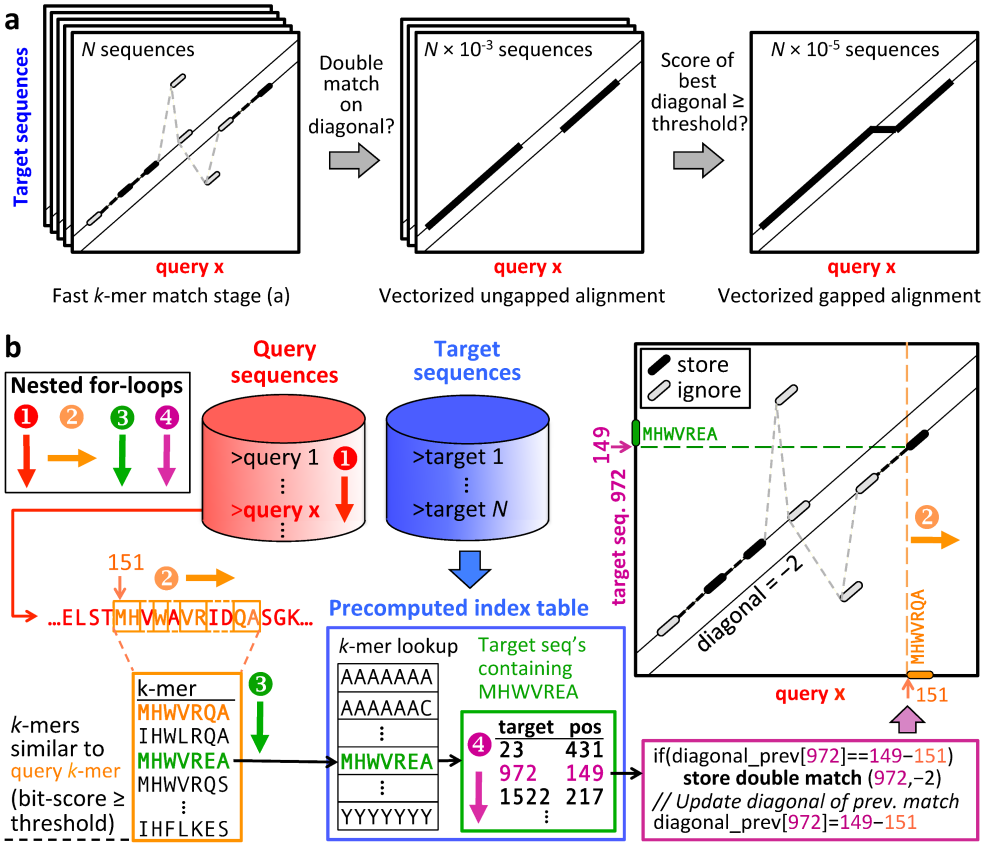

Figure 1. **MMseqs2 searching in a nutshell.** (**a**) Three increasingly sensitive search stages find similar sequences in the target database. (**b**) The k-mer match stage is key to the high speed and sensitivity. It detects consecutive similar-k-mer matches that occur on the same diagonal (positional offset) between query and 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. 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).

MMseqs2 is parallelized on three levels: First, time-critical parts are optimized using AVX2/SSE4.1 vector instructions. Second, queries can be distributed to multiple cores using OpenMP. Third, the target database can be split into chunks and distributed to multiple servers using message-passing interface, and the results are automatically merged. This also permits processing huge datasets with limited main memory.

Fast search tools have commonly been benchmarked with short query sequences translated from sequencing reads [4, 12, 30]. For such short sequences to match with statistical significance requires high similarities. For example significant matches ($E \leq 10^{-3}$) of 50 amino acids length in a search through the UniProt database have an expected sequence identity of ∼45%, making significant matches easy to detect.

Such benchmarks are not suited to compare tools as sensitive as (PSI-)BLAST and MMseqs2. Also, since disordered, low-complexity and repeat regions are known to cause false-positive matches, particularly in iterative profile searches, we devised a benchmark with full-length query sequences. We selected a subset of 7616 sequences of the SCOP database of structural domains [19] with 25% maximal pairwise sequence identity. As query sequences for our benchmark, we chose the 6370 UniRef50 sequences that yielded the best E-values in a search with the Smith-Waterman alignment tool SWIPE [21]. The benchmark database was built from two parts: all UniRef50 sequences that obtained a match to a SCOP sequence with SWIPE or HHblits [20] were annotated by the matched SCOP domain family. The unmatched parts were scrambled in a way that conserved the local amino acid composition. To obtain a large database (30.4 million sequences) with a realistic representation of low complexity and repeat regions, we added reversed UniProt sequences [16]. We used the following definition of true and false positive matches [24]: true positives have annotated SCOP domains from the same SCOP family, false matches have a SCOP domain from a different fold or match a reversed sequence, all other cases are ignored.

Some search tools are speed-optimized not only for large target sets but also for large query sets. For the run time measurements we therefore duplicated the query set 100 times, resulting in 637 000 query sequences. All searches were performed on a server with 2×8 cores and 128 GB main memory.

Figure 2a shows the cumulative distribution of search sensitivities for the 6 370 queries. Sensitivity of 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 in sensitive mode achieves a similar sensitivity as BLAST but is 36 times faster. MMseqs2 in default mode is faster and considerably more sensitive than UBLAST, RAPsearch2, and SWORD. Only DIAMOND achieves a similar speed-sensitivity trade-off, all other tools are clearly less powerful (**Fig. 2b**). MMseqs fared remarkably well in this setting even though it had been designed to detect globally alignable sequences. Similar results were obtained with single-

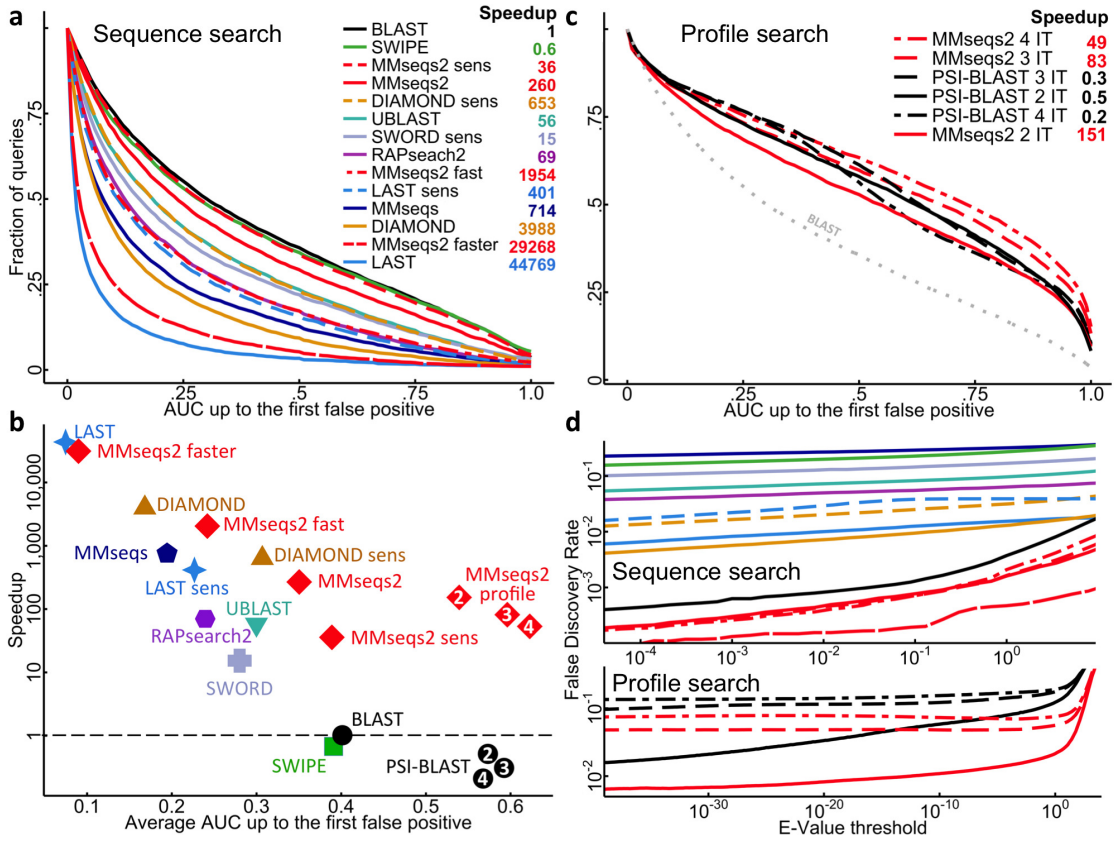

Figure 2. **MMseqs2 pushes out the boundaries of sensitivity-speed trade-off.** **a** Cumulative distribution of Area under the curve (AUC) sensitivity for all 6370 query sequences. Higher curves signify higher sensitivity. Legend: speed-up factors relative to BLAST. **b** Average AUC sensitivity versus speed-up factor relative to BLAST. White numbers: search iterations. **c** Same analysis as A, for iterative profile searches. **d** False discovery rates for sequence and profile searches (top: colors as in a, bottom: colors as in c).

domain queries (**Supplementary Fig. S3**).

Interestingly, despite their heuristic prefilters, MMseqs2 and BLAST are as sensitive as SWIPE, an accurate, vectorized implementation of Smith-Waterman alignment. The reason is that MMseqs2 and BLAST correct the scores for locally biased amino acid composition, which helps suppress matches between locally biased, non-homologous segments (**Fig. 2d, Supplementary Fig. S4**). This also shows that MMseqs2 and BLAST approach the maximum possible sensitivity.

There is no such ceiling for profile searches. We therefore extended MMseqs2 for iterative profile searching and compared it to PSI-BLAST (**Fig. 2b**). We searched with each of the 6 370 query sequences two to four iterations through the target database with default sensitivity. The comparison to BLAST (grey, dotted) demonstrates the considerable sensitivity gained through profiles. But MMseqs2 is even more sensitive than PSI-BLAST despite being 270 times faster. This is due to its efficient correction of locally biased amino acid composition that also leads to lower false discovery rates (**Fig. 2d, Supplementary Fig. S5**) and to a simple, effective measure to reduce homologous overextension of alignments [9] (online methods). As a result, we observe gains in sensitivity even for seven iterations (**Supplementary Figs. S6**).

As an example use case, we tested how much MMseqs2 would help to increase the number of annotated proteins in the Ocean Microbiome Reference Gene Catalog (OM-RGC) [25]. The speed and quality bottleneck is to find for each OM-RGC sequence homologous clusters in the eggNOGv3 [15] and KEGG databases. The BLAST search through eggNOGv3 with E-value cutoff 0.01 produced matches for 26 821 391 (67%) of the 40 154 822 OM-RGC genes [25]. We replaced the BLAST search with three MMseqs2 searches of increasing sensitivity (**Supplementary Fig. S7**). We chose an E-value cutoff of 0.1 that corresponds to the same false discovery rate as $E = 0.01$ in BLAST (**Fig. 2d**). The first MMseqs2 search in fast mode detected 23 818 024(59.3%) matches in eggNOG. The sequences without matches were searched with default sensitivity setting and 2 855 786(17.5%) had a match. The last search in sensitive search mode produced 1 138 484(8.3%) sequences with matches. In total we obtained at least one match for 27 812 294 (69%) sequences in OM-RGC. In only 1 520 CPU hours we could thus find more matches than the 100× slower BLAST searches (162 952 CPU hours) that were performed in the original study [25].

Next we sought to annotate the remaining 12 342 528 sequences using profile searches. We merged the Uniprot

database with the OM-RGC sequences and clustered this set with MMseqs2 using a 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-values below 10^{-3}. The resulting sequence profiles were searched again through eggNOGv3, and 3 530 172(28.3%) profiles obtained at least one significant match with $E < 0.1$. This increased the number of OM-RGC sequences with matches to eggNOGv3 clusters to 31 342 466 (78%) with an additional CPU time of 900 hours. In summary, in the original study BLAST was able to match 67% of the OM-RGC sequences to an eggNOG cluster [25], whereas MMseqs2 matched 78% with only 1.5% of the CPU time.

For protein sequence fragments translated directly from short reads, highly sensitive search tools are unnecessary, as sensitivity is fundamentally limited by the sequence length. However, the trend to longer reads, longer metagenomic contigs, and growing popularity of third generation sequencing technologies acutely intensifies the need for sensitive search tools to raise the fraction of annotatable ORFs in (meta)genomic datasets. The MMseqs2 software suite addresses this need and also offers various workflows to cluster even huge sequence datasets. We are now developing an algorithm for iterative profile-profile searching to further improve sensitivity at high speeds [20].

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.

ACKNOWLEDGMENTS

We thank Milot Mirdita, Lars van den Driesch and Clovis Galiez for contributing utilities and workflows. 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.

[1] E. Afshinnekoo, C. Meydan, S. Chowdhury, D. Jaroudi, C. Boyer, N. Bernstein, J. M. Maritz, D. Reeves, J. Gandara, S. Chhangawala, et al. Geospatial resolution of human and bacterial diversity with city-scale metagenomics. *Cell Systems*, 1(1):72–87, 2015.

[2] 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.

[3] M. Arumugam, J. Raes, E. Pelletier, D. Le Paslier, T. Yamada, D. R. Mende, G. R. Fernandes, J. Tap, T. Bruls, J.-M. Batto, et al. Enterotypes of the human gut microbiome. *Nature*, 473(7346):174–180, 2011.

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

[5] N. Desai, D. Antonopoulos, J. A. Gilbert, E. M. Glass, and F. Meyer. From genomics to metagenomics. *Curr. Opin. Biotechnol.*, 23(1):72–76, 2012.

[6] R. C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, Oct. 2010.

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

[8] E. A. Franzosa, T. Hsu, A. Sirota-Madi, A. Shafquat, G. Abu-Ali, X. C. Morgan, and C. Huttenhower. Sequencing and beyond: integrating molecular'omics' for microbial community profiling. *Nature Reviews Microbiology*, 13(6):360–372, 2015.

[9] 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 research*, 36(18):5863–5871, 2008.

[10] 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.

[11] D. Haussler, S. J. O'Brien, O. A. Ryder, F. K. Barker, M. Clamp, A. J. Crawford, R. Hanner, O. Hanotte, W. E. Johnson, J. A. McGuire, et al. Genome 10k: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

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

[13] A. C. Howe, J. K. Jansson, S. A. Malfatti, S. G. Tringe, J. M. Tiedje, and C. T. Brown. Tackling soil diversity with the assembly of large, complex metagenomes. *Proc. Natl. Acad. Sci. U.S.A.*, 111(13):4904–4909, 2014.

[14] B. L. Hurwitz and M. B. Sullivan. The pacific ocean virome (pov): a marine viral metagenomic dataset and associated protein clusters for quantitative viral ecology. *PLoS One*, 8(2):e57355, 2013.

[15] L. J. Jensen, P. Julien, M. Kuhn, C. von Mering, J. Muller, T. Doerks, and P. Bork. eggnog: automated construction and annotation of orthologous groups of genes. *Nucleic Acids Res.*, 36(suppl 1):D250–D254, 2008.

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

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

[18] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith. Adaptive seeds tame genomic sequence comparison. *Genome Res.*, 21(3):487–493, 2011.

[19] 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.

[20] 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.

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

[22] M. B. Scholz, C.-C. Lo, and P. S. Chain. Next generation sequencing and bioinformatic bottlenecks: the current state of metagenomic data analysis. *Curr. Opin. Biotechnol.*, 23(1):9–15, 2012.

[23] 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.

[24] 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.

[25] S. Sunagawa, L. P. Coelho, S. Chaffron, J. R. Kultima, K. Labadie, G. Salazar, B. Djahanschiri, G. Zeller, D. R. Mende, A. Alberti, et al. Structure and function of the global ocean microbiome. *Science*, 348(6237):1261359–1–9, 2015.

[26] 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.

[27] W. Tang, J. Bischof, N. Desai, K. Mahadik, W. Gerlach, T. Harrison, A. Wilke, and F. Meyer. Workload characterization for mg-rast metagenomic data analytics service in the cloud. In *IEEE International Conference on Big Data*, pages 56–63. IEEE, 2014.

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

[29] 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.

[30] 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.

ONLINE METHODS

Overview MMseqs2 (Many-against-Many sequence searching) is a software suite to search and cluster huge sequence sets. MMseqs2 is open source GPL-licensed software implemented in C++ for Linux and Mac OS. The software is designed run on multiple cores and servers and exhibits very good 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.

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.

k-mer match stage. 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.

The k-mer match stage detects 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.

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. This threshold score (option –k-score <int>‘) determines the average number of similar k-mers and thereby the trade-off between sensitivity and speed. The similar k-mers are generated with a linear-time branch-and-bound algorithm[3] that has been further accelerated in MMseqs2 using AVX2 vector instructions.

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 positions j where this k-mer occurs (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 double match. Below, we describe how this computation can be carried out within low-level, fast CPU cache without random memory access in the innermost loop.

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 has become the main bottleneck for many 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. After the end of 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 $B = \texttt{ceil}(\texttt{log2}(N/\texttt{L2_size}))$.

Index table generation. For the k-mer match stage we preprocess the target database into an index table. An array with 21^k entries contains for each of the 21^k k-mers a pointer to the list with entries ‘(`target_ID`, j) where the k-mer occurs (blue frame in Fig. 1B). Prior to index generation regions of low amino acid compositional complexity are masked out. (see Masking low-complexity regions). Building the index table file for 5×10^7 sequences takes about 45 minutes on a single core. We are working to accelerate this.

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 of the index label 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 diagonal. 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. [10], which is based on Michael Farrar's stripe-vectorized alignment algorithm [2], 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[4], 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 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. Regions satisfying one of the following criteria are masked out: (1) all 6-mers are under a bit score of 8.75 after amino acid local composition bias correction, (2) four consecutive identical residues, (3) four consecutive 2-mers with at most one mismatch between them, (4) four consecutive 3-mers at most two mismatches. Using GPLv2-licensed code from pfilt[7] and default parameters, we also mask (5) coiled coils and (6) all windows of size 12 that contain only three distinct amino acids.

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).

(1) 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.

(2) 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. Alternatively, the query sequence set is split into equal-sized chunks and each server searches its query chunk against the entire target set. Splitting the target database is less time-efficient due to the slow, IO-limited merging of results. But it reduces the memory required on each server to $7 \times NL/\#\text{chunks} + 21^k \times 8\,\text{B}$ and allows users to search through huge databases on servers with moderate memory sizes. If the number of chunks is larger than the number of servers, chunks will be distributed among servers and processed sequentially. By default, MMseqs2 automatically decides which mode to pick based on the available memory 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 tools for format conversion, multiple sequence alignment, sequence profile calculation, 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 creating of new workflows by users.

Design of sensitivity benchmark Some recent new sequence search tools were only benchmarked against short sequences, using BLAST results as the gold standard [1, 5, 11**?**]. Short matches require fairly high sequence identities to become statistically significant, making BLAST matches of length 50 almost trivial to detect even for quite insensitive tools. (For a sequence match to achieve an $E-\text{value} < 0.01$ 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, including their disordered regions, membrane helices, and other low-complexity regions. Including such 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 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 [8] is a database of protein domains of known structure organised by evolutionary relationships. Domains in the same superfamily are homologous and are counted as true positives in our benchmark, whereas domains in different folds except beta propellers can be assumed to be non-homologous and are counted as false positives [9]. All other pairs are ignored.

We measure the sensitivity of search tools using a receiver operating characteristic (ROC) analysis [9]. 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 chose not to analyse only the best match for each search 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 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. Outside of annotated regions, amino acids were shuffled randomly within overlapping windows of size 10. This preserves the local amino acid composition while precluding true positive matches in the shuffled regions.

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. As for query sequences, unannotated regions were shuffled locally. 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 [6].

Benchmarking We evaluated results up to the 4000'th 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. 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 and with a 100-fold duplicated query set (637 000 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://github.com/soedinglab/mmseqs2`.

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

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

[3] 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.

[4] 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.

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

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

[7] L. J. McGuffin, K. Bryson, and D. T. Jones. The psipred protein structure prediction server. *Bioinformatics*, 16(4):404–405, 2000.

[8] 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.

[9] 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.

[10] 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.

[11] 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.