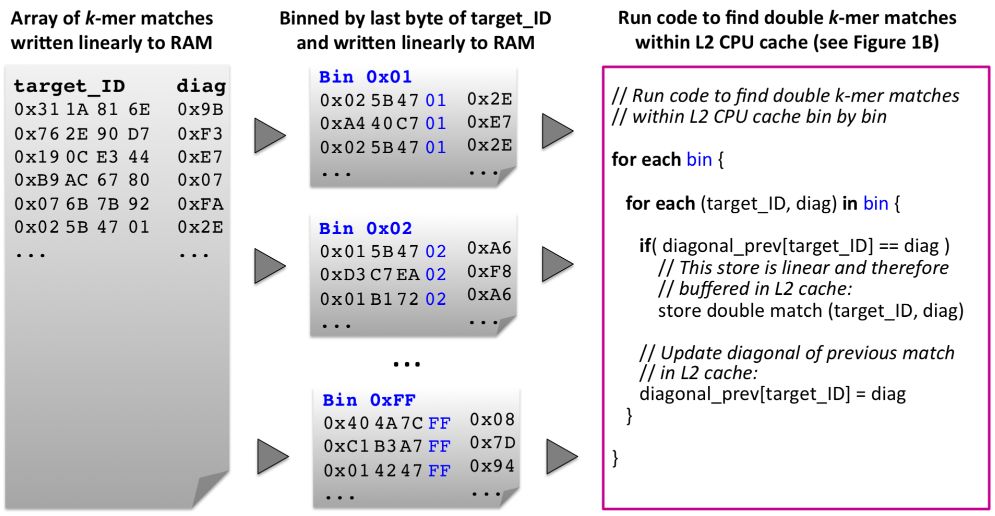
Sensitive protein sequence searching for analysis of massive data sets

Martin Steinegger and Johannes Söding[*]
Quantitative and Computational Biology group, Max-Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077 Göttingen, Germany



```
// Run code to find double k-mer matches
// within L2 CPU cache bin by bin

for each bin {

  for each (target_ID, diag) in bin {

    if( diagonal_prev[target_ID] == diag )
      // This store is linear and therefore
      // buffered in L2 cache:
      store double match (target_ID, diag)

    // Update diagonal of previous match
    // in L2 cache:
    diagonal_prev[target_ID] = diag
  }

}
```

Figure S 1. **Eliminating random memory access at k-mer match stage MMseqs2** Numbers in this figure are represented as hexadecimal (e.g. 0xFF is equal to 255 decimal). After the end of loop 2 (**Fig. 1B**), the `matches` array on the left containing single k-mer matches between the query sequence and various target sequences 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`. (Here, for illustration $B = 8$.) 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 `ceil`$(N/2^B)$ entries, where N is the number of sequences in the target database.

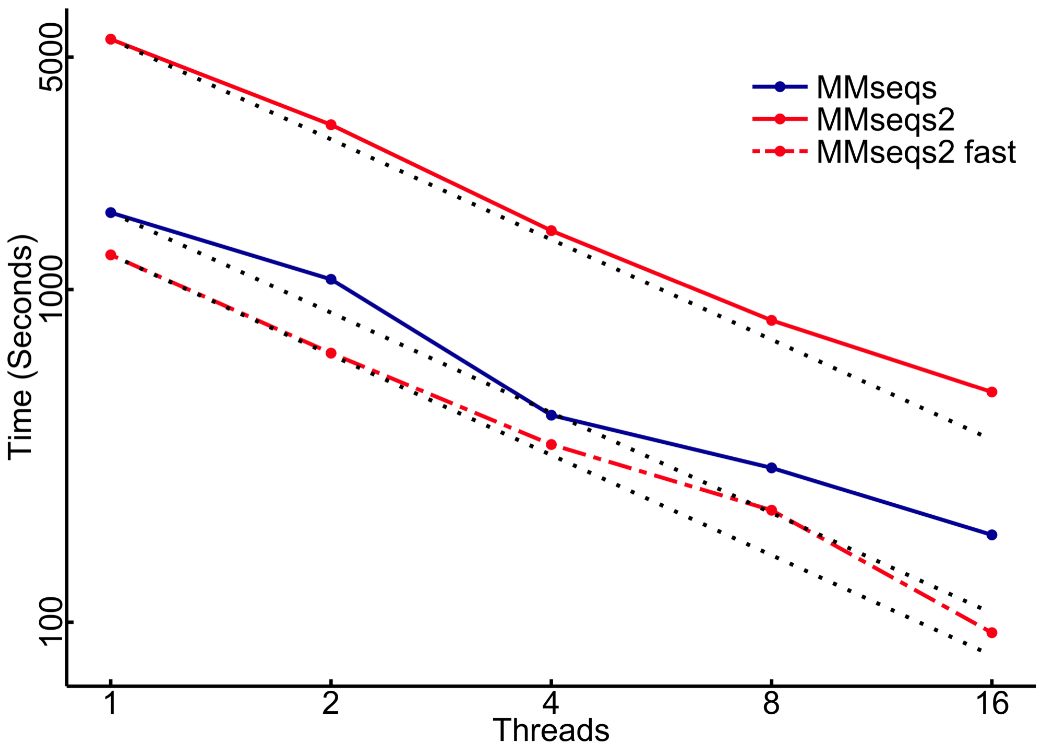


Figure S 2. **Multi core scaling of MMseqs2** Performances of MMseqs and MMseqs2 (fast, normal) using 1,2,4,8 and 16 threads. The optimal scaling is indicated as dashed black line for each method. To measure the multicore scaling performance we searched with 6370 full length protein queries against 30 Mio. Uniprot sequences. When running MMseqs2 and MMseqs with 16 cores it archives a 0.58% and 85% throughput of the theoretical maximum of the interpolated single core performance respectively. This improvement is the result of minimizing random access to the main memory, as explained in **Fig. S1**.

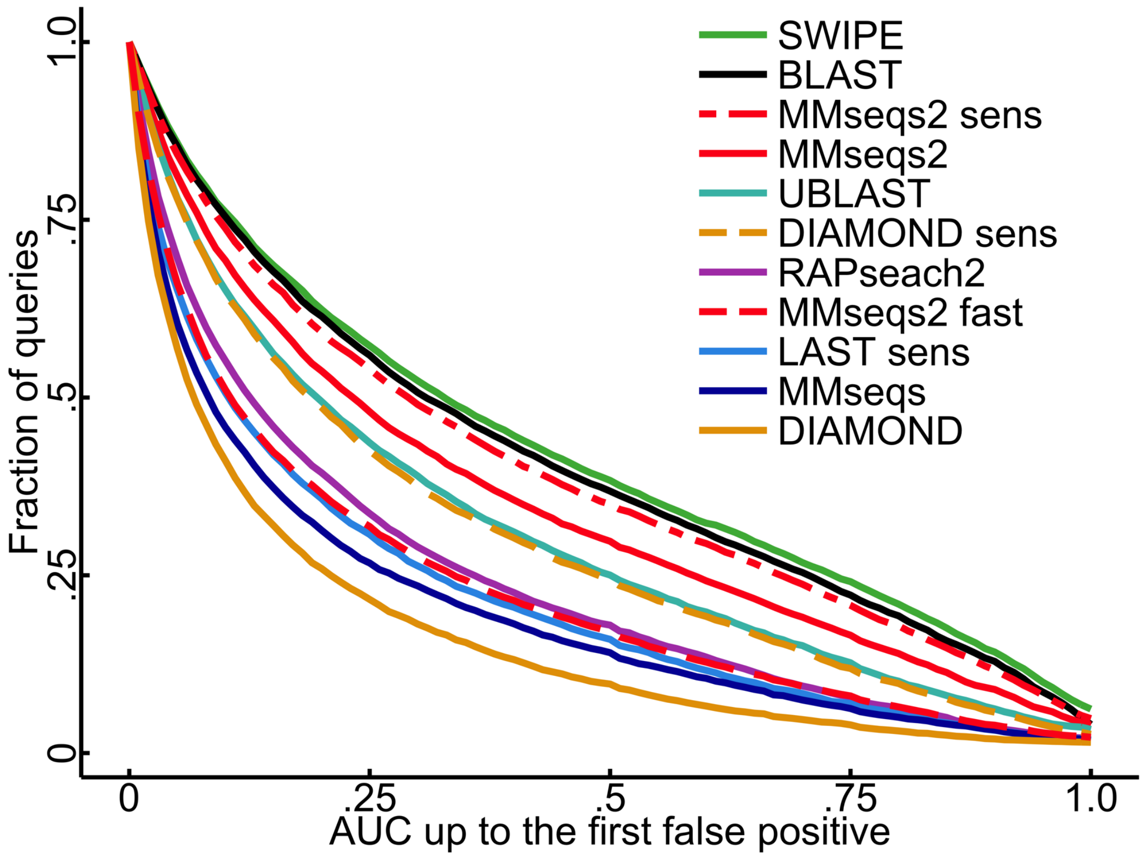


Figure S 3. **MMseqs2 single domain.** Cumulative distribution of AUC sensitivity for all 7616 single domain SCOP sequences. Higher curves signify higher sensitivity. Area under the curve (AUC) up to the first false positive is the fraction of true positive matches found with better E-value than the first false positive match.

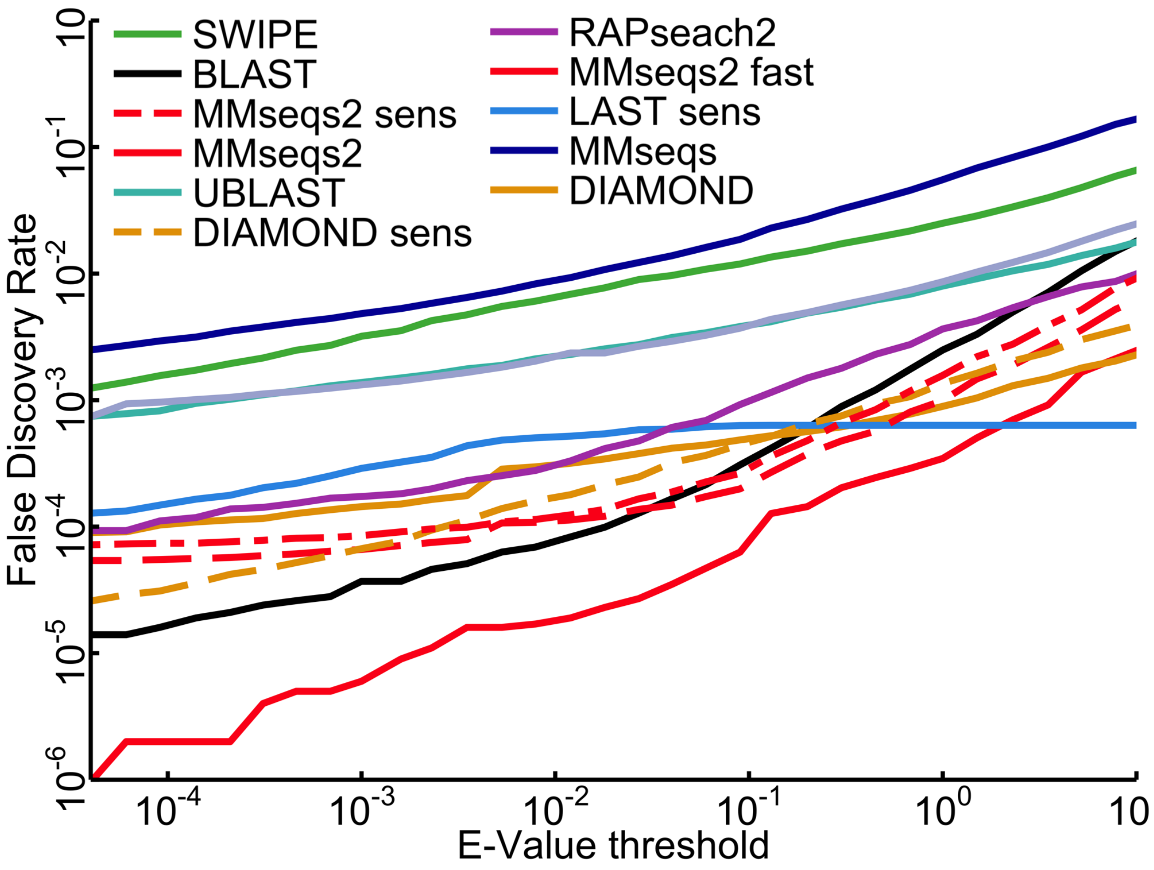


Figure S 4. False discovery rate versus *E*-value threshold. The color code is the same as in **supplementary Fig. S3**.

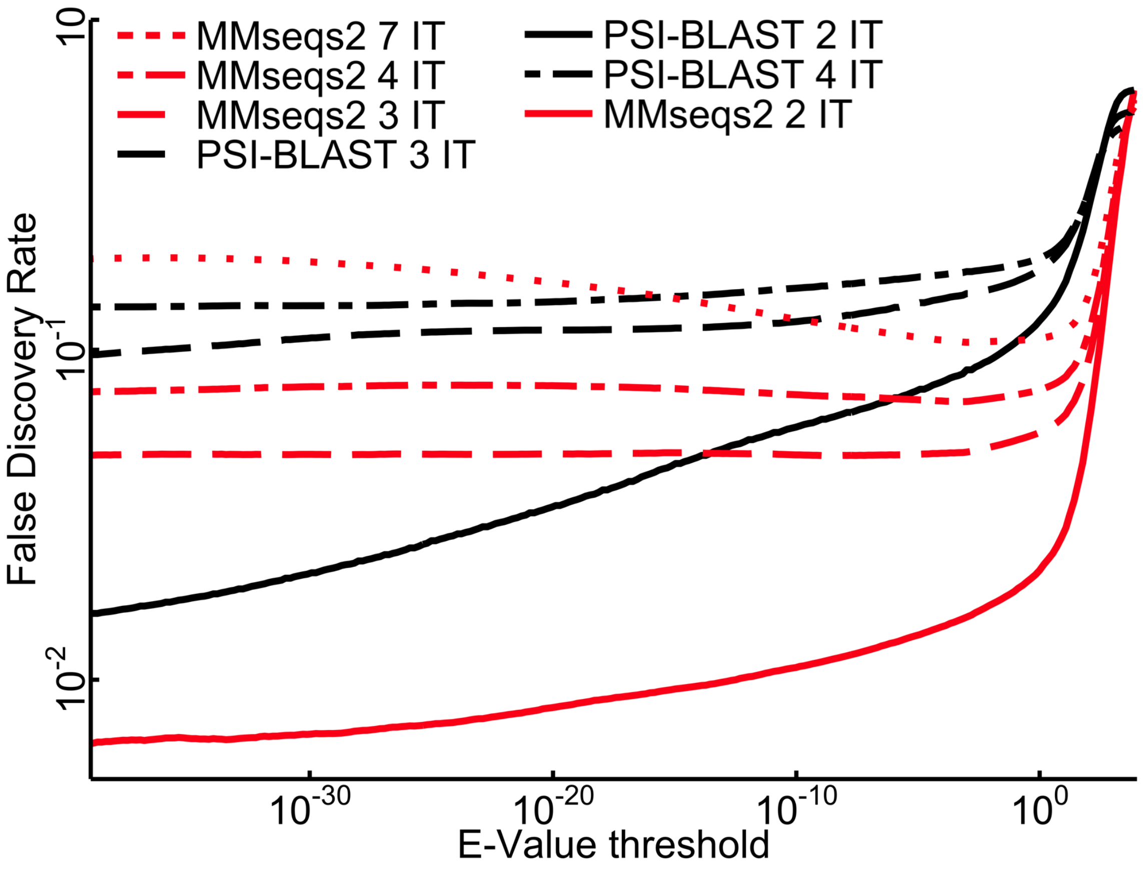


Figure S 5. False discovery rate versus *E*-value threshold for profile search.

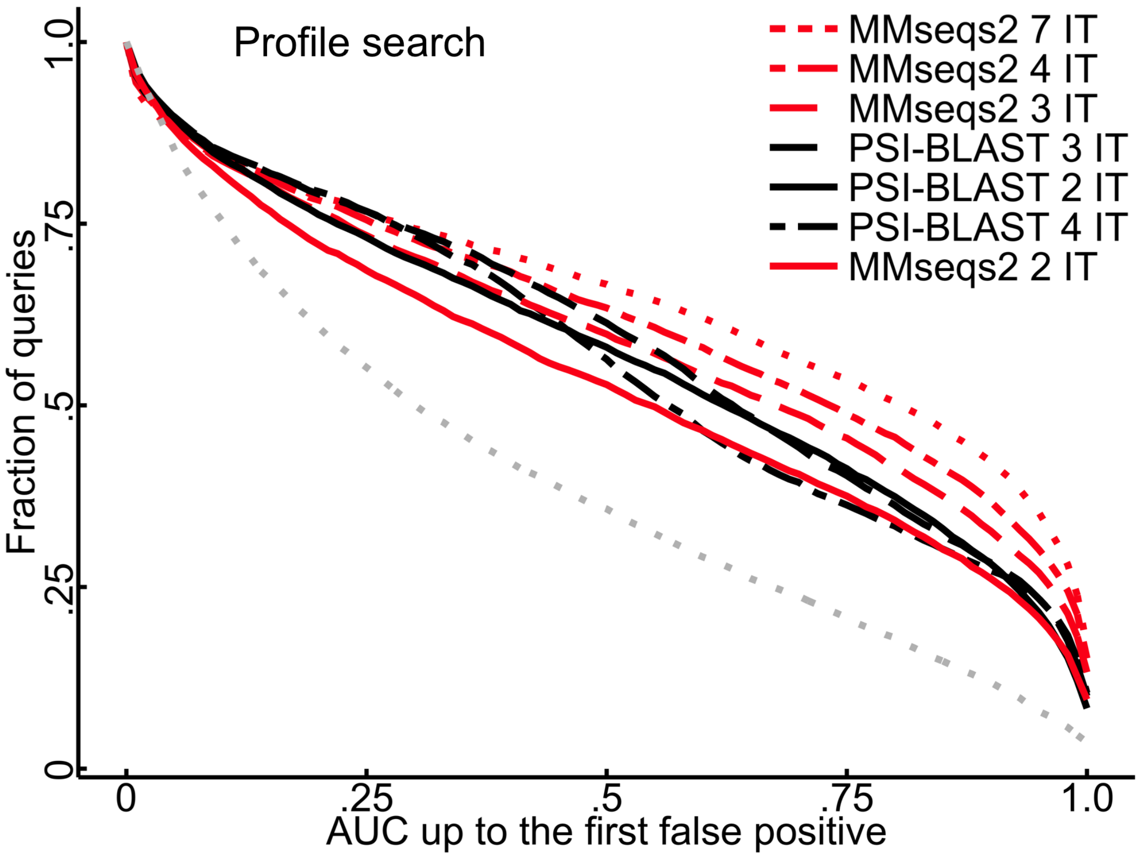


Figure S 6. Cumulative distribution of AUC sensitivity for all 6370 multi domain sequences. The sensitivity of MMseqs2 improves up to the 7th iteration.

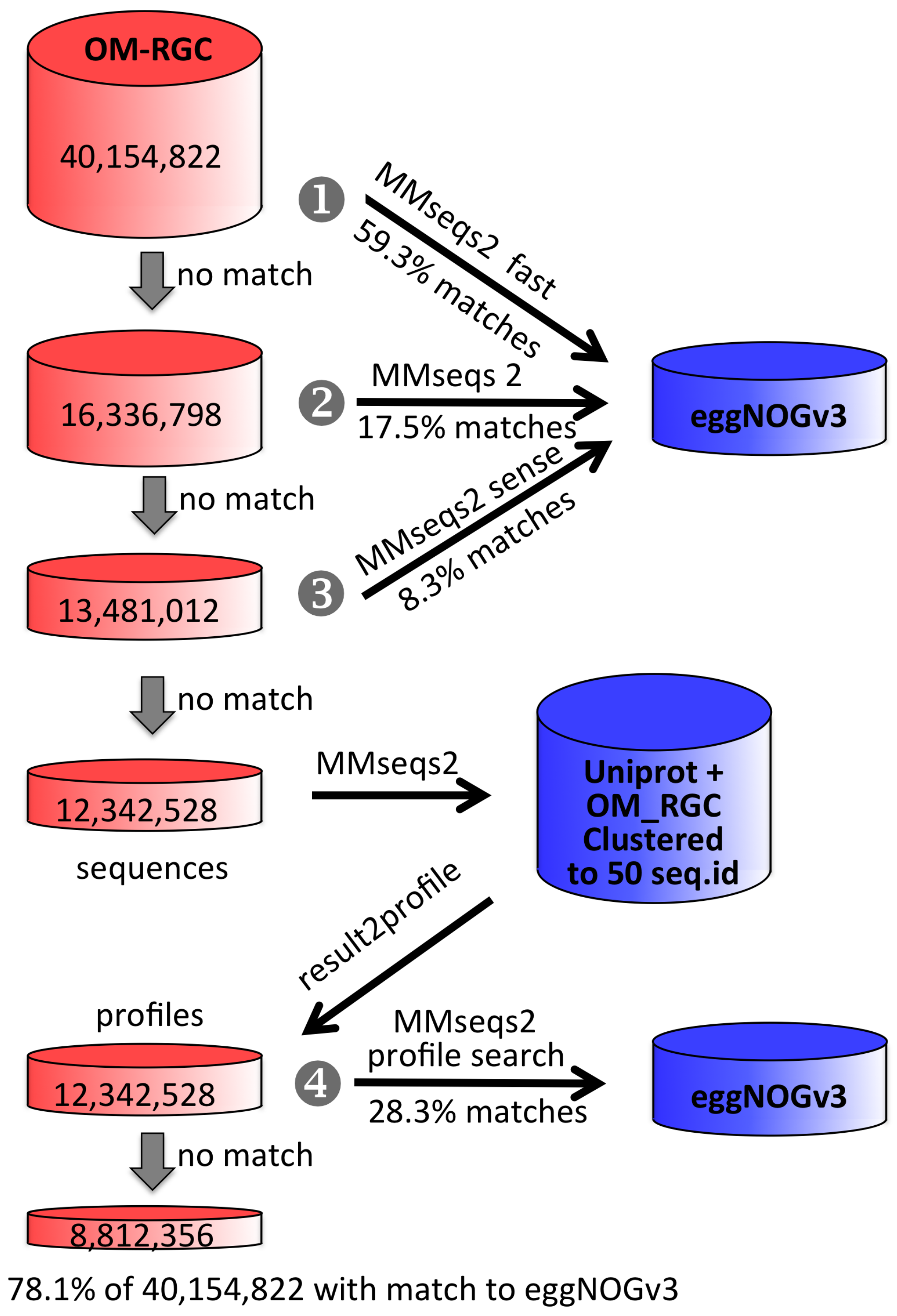


Figure S 7. Workflow for fast and deep annotation of the Ocean Microbiome Reference Gene Catalog (OM-RGC) using MMseqs2.

Feature	**MMseqs**	**MMseqs2**	**Remark**
Iterative profile searches	no	yes	Iterative profile searches increase sensitivity much beyond that of BLAST
k-mer match stage	Sums up similarity scores of similar 6-mers between pairs of sequences	Finds consecutive double 7-mer matches on the same diagonal	MMseqs aggregates scores of spurious matches across all possible $L_{\text{query}} \times L_{\text{target}}$ start positions. OK for global alignment, but suboptimal for local similarities. MMseqs2 consecutive double-diagonal k-mer match criterion suppresses most spurious matches and works well also for local similarities.
Fast gapless alignment stage	no	yes (AVX2 / SSE2)	Increases sensitivity-versus-speed trade-off by allowing MMseqs2 to evaluate more matches from the k-mer matching stage while still reducing the number of Smith-Waterman alignments
Multicore scalability	Speed-up for 16 cores is 9.3-fold	Speed-up for 16 cores is 13.7-fold	MMseqs2 minimizes random memory access by using low-level CPU cache (supplementary Figures S1, S2)
Suppression of false positive matches	Compositional bias score correction on query side in k-mer match stage	Compositional bias score correction on query and target side in all three stages	MMseqs2 eliminates high-scoring false positives much more effectively than MMseqs
Clustering methods	simple greedy strategy	Simple greedy, greedy set-cover, single-linkage with depth cut-off	MMseqs2 has option to reassign of cluster members to best representative
Utility scripts	3	37 (see MMseqs2 userguide.pdf at github)	MMseqs2 has added utility 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
Distribution of jobs on computer cluster	no	yes	MMseqs2 uses Message Passing Interface
Option to split target database across servers	no	yes	Allows MMseqs2 to search or cluster arbitrarily large databases
SIMD parallelization	SSE2	AVX2 (SSE4.1 if no support for AVX2)	AVX2 has two-fold higher parallelism and is therefore faster
Lines of code	10 000	30 000	A high proportion of the MMseqs code has been rewritten from scratch and considerably modified for better performance.

Table S I. **Comparison between MMseqs and MMseqs2.**

Method	Version	Database	Command
MMseqs2 (normal | sense)	2.0	createindex -k 7	search –k-score (95 | 85) -e 10000.0 –max-seqs 4000
MMseqs2 (very fast | fast)	2.0	createindex -k 7	prefilter –k-score (145 | 115) –max-seqs 4000
MMseqs	1.0	fasta2ffindex	–z-score-thr 10.0 -s 4 –max-seqs 4000 -c 0.0 -e 10000.0
SWIPE	2.0.11	makeblastdb -dbtype prot	-e 10000.0 -a 16 -v 4000 -b 4000
RAPsearch2	2.23	makeblastdb -dbtype prot	-v 4000 -z 16 -e 4 -t a -b 0
UBLAST	7.0.1090	-makeudb_ublast	-threads 16 -evalue 10000.0 -ublast
SWORD sens	commit fcb2117		-t 16 -a 4000 –evalue 10000
LAST	last-712	lastdb -cR01 -p -v	-P 16 -u3 -D100
LAST sens	last-712	lastdb -cR01 -p -v	-P 16 -m 4000 -u3 -D100
DIAMOND sens	0.7.9.58	diamond makedb	–max-target-seqs 4000 –evalue 10000.0 -t /dev/shm –threads 16 (–sensitive)
BLAST	2.2.31+	makeblastdb -dbtype prot	-num_descriptions 4000 -num_alignments 4000 -num_threads 16 -evalue 10000.0
PSI-BLAST	2.2.31+	makeblastdb -dbtype prot	-num_descriptions 4000 -num_alignments 4000 -num_threads 16 -num_iterations (2,3,4)
MMseqs2 profile	2.0	createindex -k 7	–num-iterations (2,3,4) -k 7 –k-score 100 -e 10000.0 –max-seqs 4000 –use-index

Table S II. Program versions and command line parameters of tools used in the benchmark.