Linclust: clustering billions of protein sequences per day on a single server

Martin Steinegger[1,2] & 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

Metagenomic datasets contain billions of protein sequences that could greatly enhance large-scale functional annotation and structure prediction. But clustering them with current algorithms is impractical because runtimes depend almost quadratically on input set size. Linclust's linear scaling overcomes this limitation, enabling us to cluster and assemble 1.6 billion sequence fragments from ∼2200 metagenomic datasets in $(10+30)$ hours on 28 cores into 711 million sequences. (Open-source software and Metaclust database: https://mmseqs.org/**).**

In metagenomics DNA is sequenced directly from the environment, allowing us to study the vast majority of microbes that cannot be cultivated [1]. During the last decade, costs and throughput of next-generation sequencing have dropped two-fold each year, twice faster than computational costs. This enormous progress has resulted in hundreds of thousands of metagenomes and tens of billions of putative gene and protein sequences [2, 3]. Therefore, computing and storage costs are now dominating metagenomics [4, 5, 6]. Clustering protein sequences predicted from sequencing reads or pre-assembled contigs can considerably reduce the redundancy of sequence sets and costs of downstream analysis and storage.

CD-HIT and UCLUST [7, 8] are by far the most widely used protein clustering tools. In their greedy clustering approach each of the N input sequences is compared with the N_{clus} representative sequences of already established clusters. Runtime therefore scales as $O(N \times N_{\text{clus}})$, almost quadratically with the number of input sequences, which results in impractical runtimes for a billion or more sequences. Main memory further constrains the size of sequence sets: UCLUST requires 10 bytes per sequence residue and CD-HIT requires 1.5 bytes.

We previously developed a clustering workflow as part of the MMseqs software suite [9]. An improved version, contained in our new MMseqs2 package [10], profits from novel, better sequence search modules. The clustering workflow consists of three cascaded steps of clustering with successively higher sensitivity, where each step consists of pairwise comparisons of all input sequences followed by clustering using the standard greedy set-cover algorithm. While MMseqs2 was developed to reach high sensitivity, Linclust was designed for maximum speed. Here, we benchmark both the MMseqs2 clustering workflow and Linclust for the first time. Both achieve linear runtime scaling, but Linclust is 30 to 60 times faster. Whereas MMseqs2 uses 6 bytes per residue, Linclust needs only 320 bytes *per sequence*.

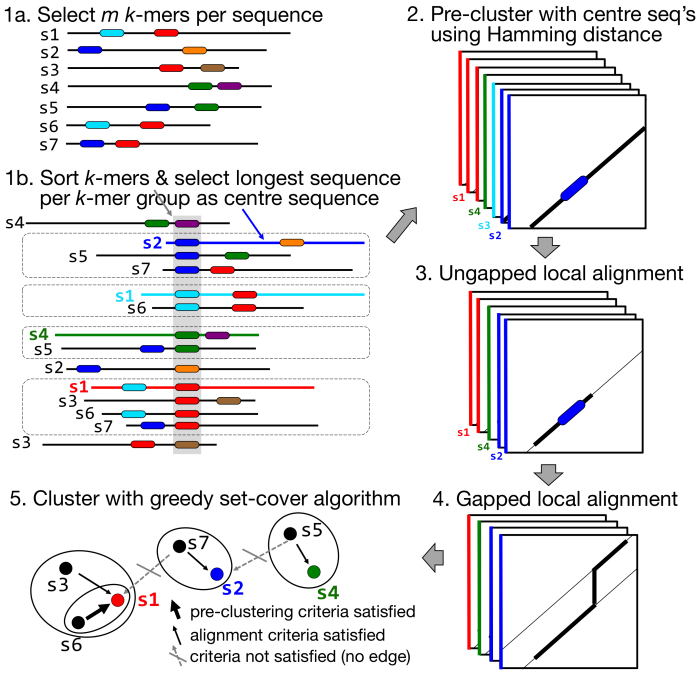

FIG. 1. The five stages of Linclust. (1) The first stage finds exact k-mer matches between the input sequences (using $k=10$ for clustering thresholds below 90% and $k=14$ otherwise). To increase sensitivity we use a reduced alphabet of size 13 here. (1a) For each sequence, Linclust selects the $m=20$ k-mers obtaining the lowest hash values (colored rounded boxes). It thereby tends to select always the same k-mers. (1b) It sorts all selected k-mers alphabetically in quasi-linear time, thereby identifying groups of sequences sharing the same k-mer (dotted boxes). For each k-mer group, the longest sequence is picked as centre. (2) For every sequence in the group the Hamming distance to its centre sequence is computed by gapless extension of the k-mer match to the ends. Sequences that already now fulfill the coverage and sequence identity criteria ("safe matches") are assigned to their centre sequences. (3) For all others the score for the ungapped alignment with the group's center sequence extending the k-mer match is computed and sequences with a sub-threshold score are filtered out. (4) All others are aligned to their centre sequence using a manually vectorized gapped local sequence alignment. Sequence pairs that satisfy coverage and sequence identity criteria are linked by an edge. (5) Finally, the sequences are clustered. Note that the number of sequence pairs processed in steps 2 to 4 is less than mN, resulting in a linear time complexity.

The Linclust algorithm is explained in Figure 1. A crucial insight to achieve linear time complexity is that we need not align every sequence with every other sequence sharing a k-mer. We reach similar sensitivities using a star alignment strategy in which for each group of sequences sharing a k-mer (dotted boxes in 1b) only a single "centre" sequence is compared to all others in the group.

We measured clustering runtimes on seven sets: the 61 522 444 sequences of the UniProt database, randomly sampled subsets with relative sizes 1/16, 1/8, 1/4, 1/2, a set consisting of UniProt plus half of the reversed sequences (92 M), and UniProt plus all reversed sequences (123M). Each tool clustered these sets using a minimum pairwise

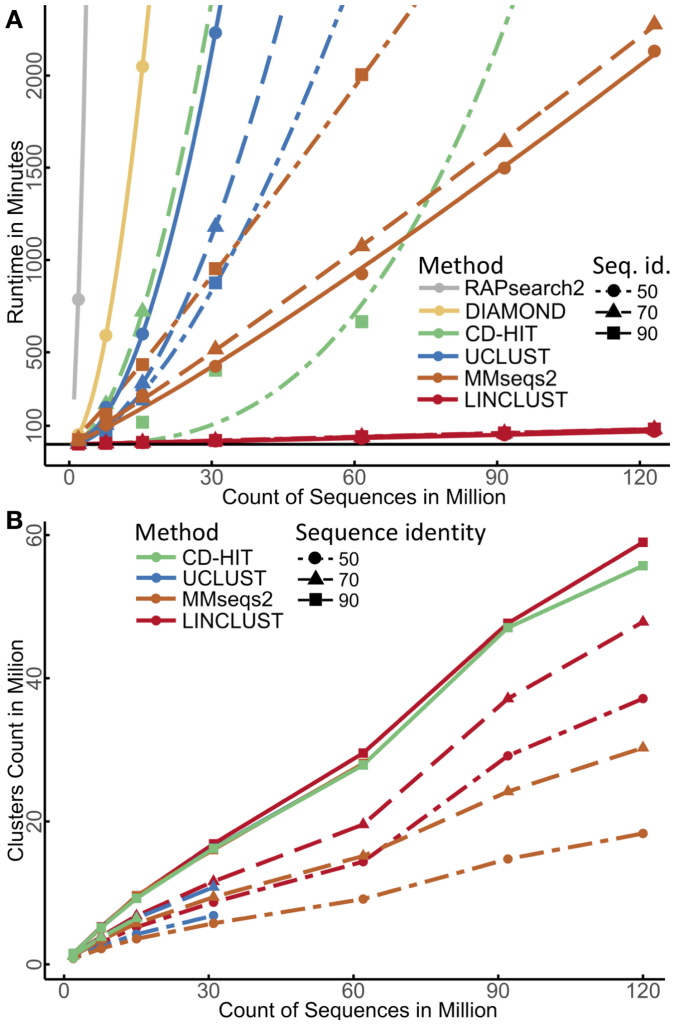

FIG. 2. **Linclust is much faster than other sequence clustering tools, and its runtime scales linearly with sequence set size.** (**A**) The plotting symbols indicate runtimes (measured on a server with two Intel Xeon E5-2640v3 8-core CPUs and 128 GB RAM), the curves are power law function fits. Tools were run with sequence identity thresholds of 90%, 70% and 50%. For comparison, we included the runtimes of all-against-all searches using sequence search tools DIAMOND and RAPsearch2. (**B**) Number of clusters at 90%, 70% and 50% sequence identity. Lower cluster numbers imply higher sensitivities to detect similar sequences. Clustering runs were aborted after 48 h to save time.

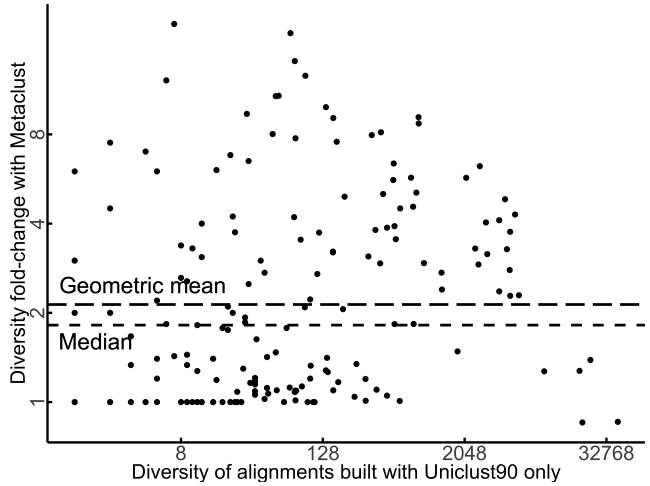

FIG. 3. **Metaclust can boost the diversity of multiple sequence alignments (MSAs).** Fold changes of MSA diversities when searching only the Uniclust90 versus searching the Uniclust90 and the new Metaclust database. MSA diversity is measured as the number of sequences after redundancy filtering. Each dot corresponds to searching with one of 190 CASP11 and CASP12 target sequences.

sequence identity of 90%, 70% and 50%.

At 50% identity, Linclust clusters the 123 million sequences 376 times faster than UCLUST, 30 times faster than MMseqs2 and, by extrapolation, more than three orders of magnitude faster than it would take CD-HIT, DIAMOND [11], and RAPsearch2 [12] (**Figure 2A**). At 90% identity, Linclust still clusters these sequences 100 times faster than UCLUST, 27 times faster than CD-HIT, and 51 times faster than MMseqs2. Whereas runtimes (at 90 % sequence identity threshold) scale with the set size N as $N^{1.62}$ for UCLUST and $N^{2.04}$ for CD-HIT, they grow only linearly for MMseqs2 ($N^{1.09}$) and Linclust ($N^{1.01}$). Linclust's Hamming distance stage decreases the exponent by 0.24, while the ungapped alignment filter mainly contributes a constant speedup factor of 1.35 (**Supplemental Fig. S1**)

To assess the clustering sensitivity we compare the number of clusters: fewer clusters imply higher sensitivity. We also measured the tools' specificities by analyzing the functional homogeneity of their clusters [13] (**Supplemental Fig. S2,S3**). MMseqs2 and Linclust show better specificity than Uclust and CD-HIT when experimentally derived Gene Ontology (GO) annotations are used for the assessment and similar specificity when all annotations are used. The similar specificities are not surprising as all tools use exact Smith-Waterman alignment and very similar acceptance criteria (**Supplemental Fig. S4**, Online Methods).

All three tools produce similar numbers of clusters at 90% and 70% sequence identity (**Fig. 2B**). Importantly, despite Linclust's linear scaling of the runtime with input set size, it manifests no loss of sensitivity for growing dataset sizes. At 50%, Linclust produces 23% more clusters than UCLUST. But we can increase Linclust's sensitivity simply by selecting more k-mers per sequence. By increasing m from 20 to 80 we loose only 34% speed but attain nearly the same sensitivity as UCLUST (**Supplemental Fig. S5**).

As an illustration of Linclust's power and flexibility, we applied it to cluster and assemble 1.59 billion protein sequence fragments predicted by Prodigal [14] in 2200 metagenomic and metatranscriptomic datasets [3, 15, 16] (**Supplemental Fig. S6**). First, we eliminated subfragments which could be aligned to a longer sequence with 99% of their residues and a sequence identity of $\geq 95\%$, producing 849 million filtered sequences in 10 hours on a 2×14-core server. In the assembly step, we ran Linclust's stage 1 followed by stage 4 on these 849 million sequences to find for each centre sequence all sequence matches that (1) extend the centre sequence on either side by at least 5 residues, (2) have an E-value below 10^{-5} and (3) at least 90% sequence

identity in their overlap region. Centre sequences that obtained matches satisfying these criteria were extended by these matches and the matched fragments were discarded (online methods). This assembly step was iterated twice. The resulting Metaclust dataset consists of 711 million sequences, which makes it, as far as we know, by far the largest freely available metagenomics-based protein sequence set.

One application of Metaclust is to build more diverse multiple sequence alignments (MSAs) (**Figure 3**). We ran MMseqs2 searches with the 190 target sequences from the community-wide critical assessments of techniques for protein structure prediction, CASP11 and CASP12 [17], and constructed MSAs from the matched sequences. In the first version, we searched the Uniclust90 database with an acceptance E-value of 0.1, in the second version we merged matches from two searches, through Uniprot90 and through the Metaclust databases, with $E \leq 0.05$. We measured the diversity of the obtained MSAs as number of sequences remaining after redundancy-filtering with 80% pairwise sequence identity and 70% minimum coverage with the query sequence. By including the search through Metaclust, we increase the diversity of MSAs dramatically, by 113% on average (dashed line). The two assembly steps helped a lot, as a search through the set of 849 million unassembled sequences led to a much lower average diversity increase of 60 % (**Supplemental Fig. S7**).

Through this increase in diversity (1) Metaclust will make profile sequence searches more sensitive and raise the fraction of annotatable sequences in genomic and metagenomic datasets [6, 16]. (2) It will also increase the number protein families for which reliable structures can be predicted de novo, as shown by Ovchinnikov et al. [18], who used an unpublished dataset of 2 billion metagenomic sequences. (3) It should allow us to predict more accurately the effects of mutations on proteins [19].

We have integrated Linclust into our software package MMseqs2 (Many-to-many sequence searches) [10]. We hope Linclust and MMseqs2 will prove helpful for exploiting the tremendous value of the many publicly available metagenomic and metatranscriptomic datasets. Linclust should lead to considerable savings in computer resources in current applications. Most importantly, we hope it will make previously infeasible large-scale analyses possible.

Acknowledgements

We are grateful to Cedric Notredame and Chaok Seok for hosting MS in their groups at the CRG in Barcelona and at Seoul National University for 12 and 18 months, respectively. We thank Milot Mirdita and Clovis Galiez for feedback to the manuscript. We thank all who contributed metagenomic datasets used to build Metaclust, in particular the US Department of Energy Joint Genome Institute http://www.jgi.doe.gov/ and their user community. This work was supported by the EU's Horizon 2020 Framework Programme (Virus-X, grant 685778) and by the Bundesministerium für Bildung und Forschung (e:AtheroSysMed 01ZX1313D).

Author contributions

MS performed research and programming, MS and JS jointly designed the research and wrote the manuscript.

[1] Rappe, M. S. & Giovannoni, S. J. *Ann. Rev. Microbiol.* **57**, 369–394 (2003).
[2] Wilke, A. *et al. Nucleic Acids Res.* **44**, D590–D594 (2016).
[3] Markowitz, V. M. *et al. Nucleic Acids Res.* **42**, D568–D573 (2014).
[4] Scholz, M. B. *et al. Curr. Opin. Biotechnol.* **23**, 9–15 (2012).
[5] Desai, N. *et al. Curr. Opin. Biotechnol.* **23**, 72–76 (2012).
[6] Prakash, T. & Taylor, T. D. *Brief. Bioinformatics* **13**, 711–727 (2012).
[7] Fu, L. *et al. Bioinformatics* **28**, 3150–3152 (2012).
[8] Edgar, R. C. *Bioinformatics* **26**, 2460–2461 (2010).
[9] Hauser, M. *et al. Bioinformatics* **32**, 1323–1330 (2016).
[10] Steinegger, M. & Soeding, J. *bioRxiv* doi: 10.1101/079681 (2017).
[11] Buchfink, B. *et al. Nature Methods* **12**, 59–60 (2015).
[12] Zhao, Y. *et al. Bioinformatics* **28**, 125–126 (2012). URL http://dx.doi.org/10.1093/bioinformatics/btr595.
[13] Mirdita, M. *et al. Nucleic Acids Res.* **45**, D170–D176 (2017).
[14] Hyatt, D. *et al. BMC Bioinformatics* **11**, 119 (2010).
[15] Kodama, Y. *et al. Nucleic Acids Res.* **40**, D54–D56 (2012).
[16] Sunagawa, S. *et al. Science* **348**, 1261359–1–9 (2015).
[17] Moult, J. *et al. Proteins* **84**, 4–14 (2016).
[18] Ovchinnikov, S. *et al. Science* **355**, 294–298 (2017).
[19] Hopf, T. A. *et al. Nature Biotechnology* (2017).
[20] Bairoch, A. *et al. Nucleic Acids Res.* **33**, D154–D159 (2005).
[21] Zhao, M. *et al. PLoS One* **8** (2013).
[22] Hauser, M. *et al. BMC Bioinformatics* **14**, 248+ (2013).
[23] Deorowicz, S. *et al. Sci Rep* **6**, 33964 (2016).

Online methods

The Linclust algorithm consists of five stages (**Fig. 1**):

1. Finding exact k-mer matches. The first stage finds exact k-mers matches between sequences that are extended in later stages 2-4. We transform the sequences into a reduced alphabet of 13 letters to increase the number of k-mer matches and hence the k-mer sensitivity at a moderate reduction in selectivity. (The alphabet optimization is described below.) We set by default $k = 10$ for clustering thresholds below 90% sequence identity and $k = 14$ otherwise. For each sequence we extract $m = 20$ k-mers, as described in "Selection of k-mers". Increasing m (option -m increases sensitivity at the cost of a moderately decreasing speed (**Supplemental Fig. S5**).

We store each extracted k-mer index (8 bytes), the sequence identifier (4 bytes), its length (2 bytes), and its position j in the sequence (2 bytes) in an array of length mN. Therefore, Linclust has a memory footprint of $mN \times 16$ bytes. We sort this array by the k-mer index using insertion sort from the OpenMP template library (http://freecode.com/projects/omptl). The sorting has a quasi-linear time complexity of $O(mN \log(mN))$ and typically takes less than 10% of the total runtime. The sorting groups sequences together that contain the same k-mer. For each such k-mer group we select the longest sequence as its centre sequence. For each array entry we overwrite the position j with the diagonal $i-j$ of the k-mer match with the centre sequence, where i is the position of the group's k-mer in the centre sequence.

2. Hamming distance pre-clustering. For each k-mer group we compute the Hamming distance (the number of mismatches) in the full amino acid alphabet between the centre sequence and each sequence in the group along the stored diagonals $i - j$. This operation is fast as it needs no random memory or cache access and uses AVX2/SSE4.1 vector instructions. Members that already satisfy the specified sequence identity and coverage thresholds on the entire diagonal are removed from the results passed to stage 3 and are added to the cluster of their centre sequence after step 5.

3. Ungapped alignment filtering. For each k-mer group we compute the optimal ungapped, local alignment between the centre sequence and each sequence in the group along the stored diagonals $i - j$, using one-dimensional dynamic programming with the Blosum62 matrix. We filter out matches between centre and member sequences if the ungapped alignment score divided by the length of the diagonal is very low. We set a conservative threshold, such that the false negative rate is 1 %, i.e., only 1 % of the alignments below this threshold would satisfy the two criteria, sequence identity and coverage. For each combination on a grid $\{50, 55, 60, \ldots, 100\} \otimes \{0, 10, 20, \ldots, 100\}$, we determined these thresholds empirically on 4 million local alignments sampled from an all-against-all comparison of the UniProt database [20].

4. Local gapped sequence alignment. Sequences that pass the ungapped alignment filter are aligned to their centre sequence using the AVX2/SSE4.1-vectorized alignment module with amino acid compositional bias correction from MMseqs2 [10], which builds on code from the SSW library [21]. Sequences satisfying the specified sequence identity and coverage thresholds are linked by an edge. These edges (neighbor relationships) are written in the format used by MMseqs2 for clustering results.

5. Clustering using greedy set cover. The sequences are clustered using the greedy set cover algorithm in MMseqs [9]. It processes the sequences in order of decreasing number of linked sequences. The top sequence and its linked neighbours are assigned to a new cluster and removed from all lists of neighbours. The next sequence having the most linked neighbours is picked until no sequence remains unassigned. Greedy set cover is fast and performs well in practice. Its runtime is proportional to the total number of edges, which is bounded by Nm.

Reduced amino acid alphabet We iteratively constructed reduced alphabets starting from the full amino acid alphabet. At each step we merged the two letters $\{a, b\} \rightarrow a' = (a \text{ or } b)$ that conserve the maximum mutual information, $\text{MI} = \sum_{x,y=1}^{A} p(x, y) \log_2 (p(x, y)/p(x)/p(y))$. Here A is the new alphabet size, $p(x)$ is the probability of observing letter x at any given position, and $p(x, y)$ is the probabilities of observing x and y aligned to each other. These probabilities are extracted from the Blosum62 matrix. When a and b are merged into a', for example, $p(a') = p(a) + p(b)$ and $p(a', y) = p(a, y) + p(b, y)$. The default alphabet with $A = 13$ merges (L,M), (I,V), (K,R), (E, Q), (A,S,T), (N, D) and (F,Y).

Selection of k-mers. To be able to cluster together sequences together we need to find a k-mer in the reduced alphabet that occurs in both. Because we extract only a small fraction of k-mers from each sequence, we need to avoid picking different k-mers in each sequence. Our first criterion for k-mer selection is therefore to extract the same subset of k-mers in all sequences. Second, we need to avoid positional clustering of selected k-mers in order to be sensitive to detect local homologies in every region of a sequence. Third, we would like to extract k-mers that tend to be conserved between homologous sequences. We note that the k-mers to be selected cannot simply be stored due to their sheer number ($\approx A^k\, m/L$).

We can satisfy the first two criteria by computing hash values for all k-mers in a sequence and selecting the m k-mers that obtain the lowest hash values. Since appropriate hash functions can produce values that are not correlated in any simple way with the hash keys, i.e. our k-mers, this method should randomly select k-mers from the sequences such that the same k-mers always tend to get selected in all sequences. We developed a simple 16 bit rolling hash function with good mixing properties, which we can compute very efficiently using the hash value of the previous k-mer (**Supplemental Fig. S8**).

In view of the third criterion, we experimented with combining the hash value with a k-mer conservation score $S_{\text{cons}}(x_{1:k}) = \sum_{i=1}^{k} S(x_i, x_i)/k$. This score ranks

k-mers $x_{1:k}$ by the conservation of their amino acids, according to the diagonal elements of the Blosum62 substitution matrix $S(\cdot,\cdot)$. We scaled the hash function with a rectified version of the conservation score: hash-value$(x_{1:k})/\max\{1, S_{\text{cons}}(x_{1:k}) - S_{\text{offset}}\}$. Despite its intuitive appeal, we did not succeed in obtaining significant improvements and reverted to the simple hash function.

Parallelization and supported platforms. We used OpenMP to parallelize all stages by applying the "#pragma omp parallel for" directive to the loops over the input sequences (stage 1a,b) or centre sequences (stages 2, 3). Linclust supports Linux and Mac OS X and CPUs with AVX2 or SSE4.1 instructions.

Tools and command line options for benchmark comparison. The sequence identity in UCLUST and CD-HIT is defined as fraction of identical residues relative to the length of the shorter sequence. In Linclust, we use a highly correlated measure (Fig. S2 in [22]) that is better suited to distinguish homologous from non-homologous sequences: the local alignment score divided by the maximum length of the two aligned sequence segments. To ensure comparable acceptance criteria, we demanded in addition a minimum coverage of 90 % of the shorter by the longer sequence.

We tested CD-HIT (version 4.6) with the parameters -T 16 -M 0 and -n 5 -c 0.9, -n 4 -c 0.7, and -n 3 -c 0.5 for 90%, 70% and 50% respectively. UCLUST (version 7.0.1090) was run with --id 0.9, 0.7, 0.5, and Linclust (commit fe2369c) was executed using --target-cov and 0.9 --min-seq-id 0.9 or --min-seq-id 0.7 or --min-seq-id 0.5 for 90%, 70% and 50% respectively. Runtimes were measured with the Linux time command.

Functional consistency benchmark. We evaluated the functional cluster consistency based on Gene Ontology (GO) annotations of the UniProt knowledge base. We carried out two tests: one based on (1) experimentally validated GO annotations and (2) general functional GO annotations (mostly inferred from homologous proteins). The UniProt 2016_03 release was clustered by each tool at 90%, 70% and 50% sequence identity level and then evaluated. For CD-HIT we computed only the clustering at 90% sequence identity because of run time constraints. For each cluster, we computed the 'worst' and 'mean' cluster consistency scores, as described earlier [13]. These cluster consistency scores are defined respectively as the minimum and the mean of all pairwise annotation similarity scores between the cluster's representative sequence and the other sequences in the cluster.

Clustering and assembling metagenomic sequences. We downloaded ∼1800 metagenomic and ∼400 metatranscriptomic datasets with assembled contigs from IMG/M [3] and NCBI's Sequence Read Archive [15] (ftp://ftp.ncbi.nlm.nih.gov/sra/wgs_aux) using the script metadownload.sh from https://bitbucket.org/martin_steinegger/linclust-analysis. We predicted genes and protein sequences using Prodigal [14] in the contigs and added the 40.2 million protein sequences from the Ocean Microbiome Reference Gene Catalog (OM-RGC) [16].

Since many of the predicted metagenomic protein sequences are fragments, we first eliminated fragments by running Linclust with the following acceptance criteria: (1) The shorter of the two sequences has at least 99% of its residues aligned and (2) the fraction of identical residues among the aligned ones is at least 95% (Linclust options --target-cov 99 --min-seq-id 0.95 --cluster-mode 2). Based on the similarity graph, we clustered the sequences using the simple greedy clustering algorithm implemented in MMseqs [9]. (We used this algorithm rather than the greedy set-cover algorithm to ensure that the representative sequence of each cluster is also always the longest one in the cluster.) The fragment elimination step took 10 hours on 2 x 14 cores and reduced the number of sequences to 736 million.

For the fragment assembly, we then ran stage 1 of Linclust (**Fig. 1**) to find k-mer matches (options -c 0.0 –min-seq-id 0.9) and we aligned all matched pairs using stage 4 (options –min-seq-id 0.9 -e 0.00001). We accepted only sequence matches that extended the centre sequence either to the left or right by at least 5 residues, that achieved an E-value below 10^{-5}, and that had a fraction of at least 90% identical residues in the aligned region. This resulted in 115 million centre sequences with at least one accepted match. We ran FAMSA[23] to compute a multiple sequence alignment (MSA) for each of these centre sequences. The assembled sequences were computed from the MSAs by taking the most frequent amino acid of each column in the MSA. The assembly increased the average length of the 115 million centre sequences from 168 to 208 amino acids. All sequences in the 115 million MSAs were removed and replaced by the 115 million assembled sequences. We repeated the assembly step and could further assemble 23.5 million centre sequences, increasing their average length from 245 to 298. The final "Metaclust" database contains 711 million sequences of metagenomic origin.

Metaclust protein sequence sets. The Metaclust database is available as FASTA formatted file at https://metaclust.mmseqs.org/.

Code availability. Linclust has been integrated into our free GPLv3-licensed MMseqs2 software suite [10]. The source code and binaries for Linclust can be download at https://github.com/soedinglab/mmseqs2.

Data availability. All scripts and benchmark data including command-line parameters necessary to reproduce the benchmark and analysis results presented here are available at https://bitbucket.org/martin_steinegger/linclust-analysis.