TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Bioinformatik

Ultrafast and sensitive sequence search and clustering methods in the era of next generation sequencing

Martin Steinegger

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Michael Georg Bader

Prüfer der Dissertation:

1. Prof. Dr. Burkhard Rost
2. Priv.-Doz. Dr. Hanjo Täubig

Die Dissertation wurde am 07.05.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13.08.2018 angenommen.

Abstract

Environmental metagenomic studies are a rich source for unraveling genetic diversity of yet unknown and mostly unculturable microbes, with terabytes of short read sequencing data already publicly available. The growth rate of such data has been increasing steadily, driven by the exponential increase in throughput, which has been outperforming Moore's law by two-fold in the last decade. To date, the main bottleneck is not the generation but rather the analysis of sequence data.

To address this analysis bottleneck we propose three novel methods: (1) MMseqs, a fast and sensitive clustering method, (2) MMseqs2, a fast and sensitive homology search method, and (3) Linclust, the first sequence clustering algorithm whose runtime scales linearly with the number of sequences and independently of the number of clusters obtained.

Clustering is the process of assigning sequences to distinct groups based on similarity. Thus, clustering serves as a means to discover biological connections or define families of homologous sequences. Furthermore, clustering can speed up downstream analysis considerably by reducing highly similar sequences to a single sequence. MMseqs is a sensitive all-against-all search-based clustering tool that can cluster huge protein sets with high sensitivity. Current fast state-of-the-art clustering methods lack sensitivity below 50% while MMseqs detects similarities at 30% sequence identity. We show that our method is not only more sensitive but also faster than other state-of-the-art methods.

Homology search is widely used in life science research to infer the functions and structures of the novel protein sequences by identifying significant similarities to known sequences. Sensitive methods such as BLAST/PSI-BLAST are too slow to handle huge data sets. Fast methods trade sensitivity for runtime speed, leading to many missed annotations. To address this dilemma, we developed MMseqs2, a successor to MMseqs, which is as sensitive as BLAST but 35 times faster. Moreover, MMseqs2 is the first fast iterative profile search tool with even higher sensitivity than that of PSI-BLAST at 400 times the speed. MMseqs2 also supports fast reverse profile searches, which we used to annotate 1.1 billion metagenomic protein sequences in 8.3 hours on a single 28 core computer. The speed and sensitivity of MMseqs2 makes it a powerful tool to annotate protein sequences in the era of big data.

State-of-the-art sequence clustering methods scale in $O(NK)$ where N is the input set size and K is the amount of clusters. Since K is often close to N, these methods follow a nearly quadratic runtime complexity. Linclust is the first algorithm that overcomes this quadratic runtime and runs in linear time $O(N)$. Linclust can cluster protein sets down to 50% sequence identity three to four orders of magnitude faster than other methods. We clustered 1.6 billion protein sequences from about ~2200 metagenome and metatranscriptome assemblies down to 50% sequence identity using Linclust, which took 10 hours on a single server with 28 cores. The near quadratic run time of state-of-the-art methods would make this analysis nearly infeasible.

Zusammenfassung

Daten aus metagenomischen Studien sind eine reichhaltige Quelle für Informationen über genetische Diversität. Viele Terabyte an Sequenzdaten dieser Studien sind öffentlich frei verfügbar und wachsen stetig an. Der Grund für das schnelle Wachstum sind die exponentiell abnehmenden Kosten und der stetig steigende Durchsatz der Sequenzierungsverfahren. Technische Innovationen in diesem Feld haben das Moore'sche Gesetz in den letzten zehn Jahren um ein Zweifaches übertroffen. Aktuell ist nicht mehr die Erzeugung, sondern die Analyse der Sequenzdaten der größte Zeit- und Kostenfaktor.

In dieser Arbeit stellen wir drei neue Methoden der Analyse metagenomisher Daten vor, die schneller und somit kostengünstiger sind als bisherige Methoden: (1) MMseqs, eine schnelle und sensitive Clustering-Methode, (2) MMseqs2, eine schnelle und sensitive Methode zur Suche nach homologen (verwandten) Sequenzen und (3) Linclust, den ersten Sequenz-Clustering-Algorithmus, dessen Laufzeit linear mit der Anzahl der Sequenzen und unabhängig von der Anzahl der Cluster wächst.

Clustering teilt Sequenzen in biologisch verwandte Gruppen ein. MMseqs ist ein sensitives Clustering-Tool, dass hierfür alle Sequenzen der gegebenen Menge miteinander vergleicht und diese dann auf Grundlage der ermittelten paarweisen Sequenzidentität in Gruppen einteilt. MMseqs kann Sequenzähnlichkeiten von bis zu 30% der paarweisen Sequenzidentität erkennen, wobei es bis zu 2500 Mal schneller ist, als vergleichbare Clustering-Methoden.

Suche nach homologen Sequenzen ist in den Biowissenschaften ein fundamentaler Baustein zur Vorhersage der Funktion eines Proteins. Viel genutzte sensitive Such-Methoden, wie PSI-BLAST und BLAST, sind zu langsam, um viele Suchen in einer großen Datenbank durchführen zu können. Gleichzeitig sind andere schnelle Methoden nicht sensitiv genug und können viele Sequenzen somit nicht zuordnen. MMseqs2 ist 35 Mal schneller als BLAST und weist eine ähnliche Empfindlichkeit der Suche auf. Die Profil-basierte Suche in MMseqs2 ist ähnlich empfindlich wie PSI-BLAST und etwa 400 Mal schneller. Wir konnten damit 1,1 Milliarden metagenomischer Proteinsequenzen in 8,3 Stunden auf einem Computer mit 28 Kernen zu bekannten Sequenzen zuordnen.

Ein großes Problem für das Clustering der metagenomischen Sequenzdaten stellt die Laufzeit der derzeit verfügbaren Clustering-Algorithmen dar. Aktuelle Methoden skalieren mit einer Laufzeitkomplexität von $O(NK)$, wobei N die Anzahl der analysierten Sequenzen und K die resultierende Anzahl der Cluster darstellt. Hiermit ergeben sich fast quadratische Laufzeiten, da K oft eine ähnliche Größenordnung wie N hat. Linclust ist der erste Clustering-Algorithmus, dessen Laufzeit linear in N und unabhängig von K ist. Linclust kann Proteinsequenzen mit bis zu 50% Sequenzidentität einem Cluster zuweisen und ist dabei drei bis vier Größenordnungen schneller als andere vergleichbare Methoden. Wir haben 1,6 Milliarden Proteinsequenzen von ~2200 Metagenom- und Metatranscriptom-Assemblierungen mit Linclust in 10 Stunden auf einem einzelnen Server mit 28 Kernen auf 50% Sequenzidentität geclustert. Mit anderen Methoden wäre diese Analyse aufgrund der quadratischen Laufzeit nahezu unmöglich gewesen.

List of Publications

This work constitutes a cumulative dissertation based on peer-reviewed publications. Chapters 2, 3, and 4 describe the published and submitted articles included in this dissertation:

- Maria Hauser[#], **Martin Steinegger**[*]#, Johannes Söding[*], *MMseqs software suite for fast and deep clustering and searching of large protein sequence sets*. Bioinformatics 2016. doi: 10.1093/bioinformatics/btw006 (#Equal contributions.) ([*]Corresponding authors.)
- **Martin Steinegger**, Johannes Söding, *MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets*. Nature Biotechnology 2017. doi: 10.1038/nbt.3988.
- **Martin Steinegger**, Johannes Söding, *Clustering huge protein sequence sets in linear time*. Nature Communications, 2018. accepted

During the dissertation work, I was co-/corresponding author of other peer-reviewed publications:

- Milot Mirdita[#], Lars von den Driesch[#], L., Galiez, G., Maria Martin, Johannes Söding[*], and **Martin Steinegger**[*] (2017) *Uniclust databases of clustered and deeply annotated protein sequences and alignments.* Nucleic Acids Res. 2017, doi: 10.1093/nar/gkw1081. (#Equal contributions.) ([*]Corresponding authors.)
- Yannick Mahlich[*], **Martin Steinegger**, Burkhard Rost, Yana Bromberg[*] *HFSP: High speed homology-driven function annotation of proteins.* Bioinformatics, 2018. accepted ([*]Corresponding authors.)

Acknowledgments

First and foremost, I want to thank Dr. Johannes Söding for guiding me through academia and life in general. I am very thankful for all the freedom and guidance you gave me. Our scientific discussions helped me shape my own thinking and are now the basis for my future academic career.

All the talented people at the Söding Lab, for supporting my projects and accepting my high usage of CPU/hard disk of the compute cluster. Special thanks to Milot Mirdita for many hours of reviewing my texts, discussing ideas and helping me when I was feeling down. Not only this thesis, but even my life, would not be the same without your help. To many more discussions about crazy ideas. Thank you to Maria Hauser for the great collaboration on MMseqs. Nikos Papadopoulos for helping me to review this thesis. Susann Vorberg for presenting in my name at the CASP12 meeting. Eli Levy Karin for the great discussion about benchmarking and her offer to show me around the beautiful Tel Aviv. Clovis Galiez for the great collaboration and all the work you did for MMseqs2.

Thank you Prof. Dr. Burkhard Rost for helping me constantly, especially encouraging me to pursue a career in academia and for accepting me as your Ph.D student. To all people at the Rostlab who helped me find my way in academia. Especially Laszlo Kajan, who supported my first scientific projects and introduced me to Johannes. Yannick Mahlich for the great collaboration at the HFSP project. Juan Miguel Cejuela for providing me the Latex template for my Ph.D.

I want to thank Prof. Dr. Cedric Notredame for hosting me for a year at the Centre for Genomic Regulation in Barcelona. The time at your lab inspired me. Seeing your passion for science helped me build my own. Thank you to everybody at the Notredame Lab for welcoming me and helping me develop as a scientist. Especially to Dr. Maria Chatzou, for teaching me the three basic rules for having a relationship and Dr. Pablo Prieto Barja for being a great friend and helping me constantly whenever I got stuck with the Catalan/Spanish system.

I am grateful to Prof. Dr. Chaok Seok for having me at the Seoul National University for such a long time. Your calm, supportive and friendly way inspires me. The time is flying so fast in your lab, since it has such a great environment. I am grateful to be part of it. Your constant support helped me a lot. I also want to thank the members of the Seok Lab who took their time to explain to me how life works in South Korea. Thank you Minkyung Baek for helping me, whenever I had a hard time with Korean bureaucracy or general communication issues. Taeyong for our great discussions about longevity and start-ups.

I am thankful to Prof. Dr. Martin Frith for helping to improve MMseqs2 and inviting me to Japan, Prof. Dr. Yana Bromberg and Prof. Dr. Shini Sunagawa for giving me great career advice, Prof. Dr. Ernst Mayr for helping me with my Ph.D. Committee and Hanjo Täubig for agreeing to grade my thesis.

Thank you to everyone that builds open source databases, software and open science. Your public efforts change the world to a better place. My work would not be possible without your efforts.

I am thankful to all my friends, who supported me throughout the way.

Last but not least, a thank you to my parents. You are the most important people in my life. You taught me how to live my life and you always supported my decisions. You cannot believe how important this was for me. I love you.

♥

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Next generation sequencing and the rise of metagenomics

The human genome project started in 1990 and released the first human draft genome in 2001 (IHGSC 2001). DNA extraction and sequencing depended heavily on human labor. The genome was split into 150 kilobase-long fragments and each fragment was then sequenced using shotgun Sanger sequencing (Sanger, Nicklen, and Coulson 1977). Retrieving this genetic information was a billion dollar effort. Eighteen years later, thanks to advances in sequencing technology, these costs dropped to 1,000$ per genome. (Hayden 2014a; Hayden 2014b). During the last decade, costs and throughput of next-generation sequencing have dropped two-fold each year – twice faster than computational costs decreased (Lander 2009) – thus, reducing sequencing costs by several orders of magnitude. The human labor costs involved in sequencing also became negligible. As a result, in the last years, we had a massive increase of DNA sequence data generation (Muir et al. 2016).

Aside from human genetics, the rapid development of sequencing opened up various new fields in biology, like metagenomics (Desai et al. 2012). In metagenomics, DNA is sequenced directly from the environment, allowing us to study the vast majority of microbes that cannot be cultivated *in-vitro* (Rappe and Giovannoni 2003). Since the year 2006, there is an exponential increase of publications (see Figure 1.2) containing the word "metagenome" in the life sciences publication index PubMed (McEntyre and Lipman 2001).

Metagenomic studies impact various fields, finding links between human health and microbial communities on skin, gut, or the reproductive system. They enrich our understanding of ecology and climate: 50 % of oxygen produced on earth originates from plankton communities (Field et al. 1998). They enable the discovery of new enzymes for biotechnology (Levin et al. 2017), such as new CRISPR systems (Burstein et al. 2016), and of novel bioactive natural drug compounds, such as an anti-HIV compound (Smith et al. 2018), or novel antibiotics (Banik and Brady 2010). They aid in the detection of missing evolutionary links between archaea and eukaryotes (Zaremba-Niedzwiedzka et al. 2017) and offer ways to study prehistoric diets of the *Homo neandertaler* (Weyrich et al. 2017).

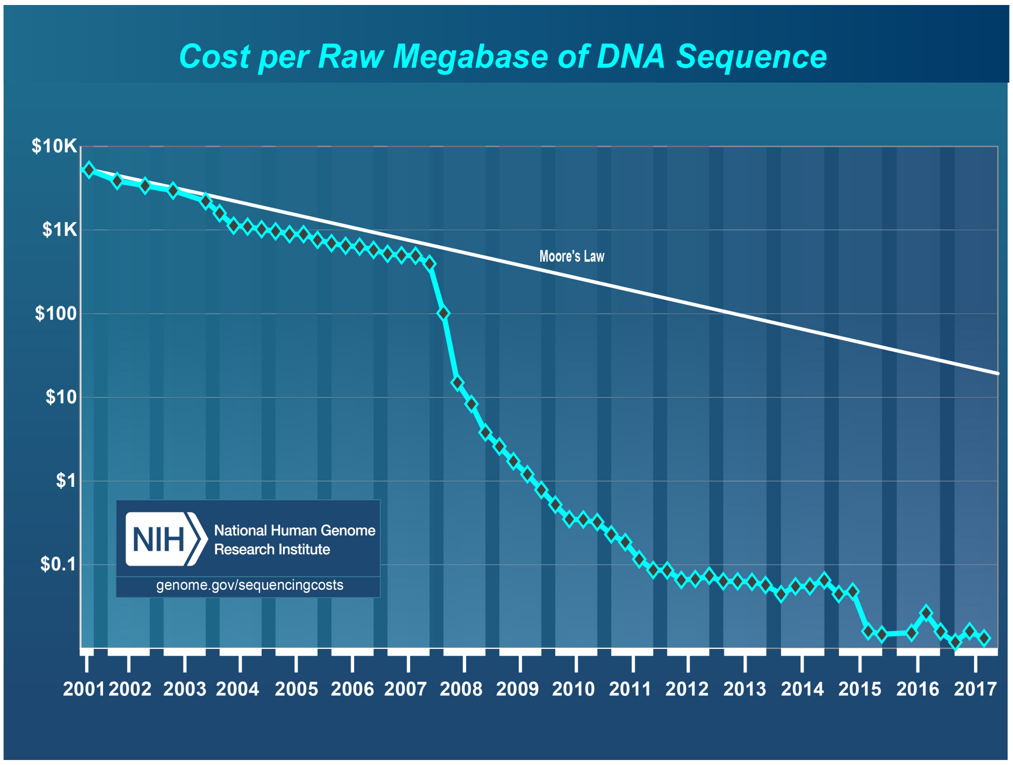


Fig. 1.1. Sequencing price per megabase is falling faster than computing power is increasing. The cost to sequence a base pair is dropping faster than Moore's Law. Moore's Law describes a technology trend in which the number of transistors in microprocessors double approximately every two years. Therefore, at least for well parallelizable problems, the available computational power doubles every two years. Sequencing costs of second and third generation sequencing technology are halving faster than computational power is doubling, with an ever increasing gap. The graphic was taken from the NHGRI Genome Sequencing Program (Wetterstrand 2017).

These studies depend heavily on high throughput sequencing and only became feasible with the introduction of next generation sequencing in 2005 (Margulies et al. 2005) . Estimates are claiming that to sequence the genomes of all bacteria present in a single gram of soil terabases of sequencing data would be needed (Rodriguez-R and Konstantinidis 2014b; Rodriguez-R and Konstantinidis 2014a; Howe et al. 2014). This is due to the high genomic diversity; estimates for the number of distinct genomes in a single gram of soil range from the thousands (Delmont et al. 2011) to even tens of millions (Gans, Wolinsky, and Dunbar 2005).

Since the effort to sequence every microbial environment is too large for a single laboratory to handle, consortiums were started to work on different microbial environments. The focus of the NIH Human Microbiome Project (Peterson et al. 2009) is to characterise the human microbiome. The TARA Oceans Consortium (Karsenti 2015) sails around the globe, sampling and sequencing the oceans. The Earth Microbiome Project (Gilbert, Jansson, and Knight 2014) aims to characterise all microbial life on earth.

These enormous efforts have resulted in hundreds of thousands of metagenomes and tens of billions of putative genes and protein sequences (Wilke et al. 2016; Markowitz et al. 2014). The metagenomic sample data sets hosted by the European Bioinformatics Institute (EBI) had an eightfold increase from 2015 to 2016, from 8000 samples to over 75 000 samples (Cook et al. 2018). The EBI estimates it will be using over an exabyte of storage in the year 2022 at the current sequencing rate, compared to around 120 petabytes of metagenomic samples it is hosting now.

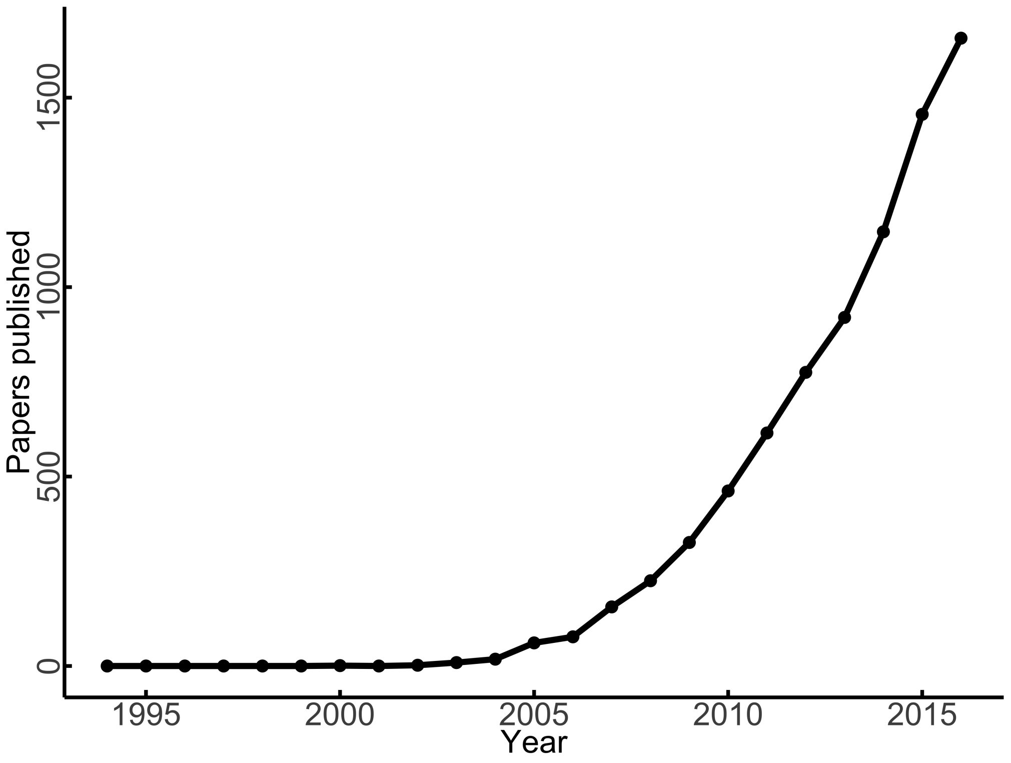


Fig. 1.2. Since the introduction of next generation sequencing technology, metagenomic studies are flourishing. The figure shows a time series of published papers containing the word "metagenomic", extracted from PubMed using Medlinetrend (Corlan 2018). Metagenomic studies depend heavily on high throughput sequencing methods. Indeed, the introduction of the first next generation sequencer GS20 (Margulies et al. 2005) was in 2005 followed by an exponential increase in publications from 2006.

As computing and storage costs are now dominating metagenomics (Scholz, Lo, and Chain 2012; Prakash and Taylor 2012), the need for fast and sensitive analysis methods is greater than ever. One of the major bottlenecks in computational analysis are the clustering (Li et al. 2012) and homology search steps (Scholz, Lo, and Chain 2012). Clustering protein sequences can considerably reduce the redundancy of sequence sets and costs of downstream analyses and storage (Sikic and Carugo 2010). Homology search (Pearson 2013) is used to infer biological attributes, such as common function from evolutionary related sequences, also called homologous sequences.

In this thesis I propose methods for fast and sensitive clustering and homology search techniques. They can help close the gap between sequencing and analysis costs and address this need. The next chapter will introduce and explain the state of the art in homology search and clustering of protein sequences.

1.2 Homology Search

Homology search is the backbone of bioinformatics. It is often used to infer or predict function (Rost 2002), taxonomy (Huson et al. 2007), phylogeny (Huerta-Cepas et al. 2011), secondary (Rost and Sander 1993; Jones 1999) and tertiary structure (Söding 2005), cellular localization (Goldberg et al. 2014), single point mutation effect strength (Bromberg and Rost 2007; Ng and Henikoff 2003), or protein interactions (Snel et al. 2000). Homology can be detected by comparing a novel sequence against a database of known reference sequences like the NR (Pruitt, Tatusova, and Maglott 2007), PFAM (Finn et al. 2016), UniProt (Bairoch et al. 2005; Apweiler et al. 2004), UniRef (Suzek et al. 2007), KEGG (Kanehisa and Goto 2000), PDB (Kouranov et al. 2006), EggNOG (Jensen et al. 2007) or Uniclust (Mirdita et al. 2016).

```
G1SJM8_2 YTSAGVSVTVQ----ELFRVPVLRASSPLPLQEGSAVTLSCETKLLPDSPPLRLYFS
Q8R142_1 YTSAGVSITVKAFPLELFTTPVLRASV---FPEGSLVTLNCETNLLLQRPGLQLYFS
```

Fig. 1.3. Visualization of a pairwise sequence alignment of two homologous sequences. An alignment contains exact matches or substitutions (also called point mutations), and deletions or insertions (represented by the dash character "-"). This alignment has a sequence identity of 61.4 %, since 35 out of 57 aligned residues including gaps are identical. A local alignment does not necessarily include all residues from the sequences that are aligned. In such cases, an important attribute of the alignment is the "length overlap", which is computed by (alignment end position − alignment start position)/sequence length. In the example depicted above, the length overlap is 100% since all residues of both sequences are included in the alignment.

Pairwise alignments (as shown in figure 1.3) are the basic building blocks of homology search. An alignment between two sequences can be used to measure the evolutionary distance by assessing the substitutions, insertions and deletions that occurred over time. Figure 1.3 shows a pairwise alignment of a query and a target sequence. A simple measure for closeness would, for example, be sequence identity. It is the fraction of exact matching residues divided by the alignment length. A low sequence identity implies a long evolutionary distance.

Substitution matrices

Substitution matrices are used to describe the evolutionary distance of two amino acids. The computation of such matrices usually relies on some data from which the tendency of each amino acid to mutate to another (joint probability) can be estimated as well as the occurrence of each amino acid (independent probability). Most substitution matrices such as PAM (Dayhoff and Schwartz 1978) and BLOSUM (Henikoff and Henikoff 1992) are given as the log of the odds ratio between the estimated joint probability and the independent probabilities of the amino acids.

The most used protein substitution matrix is the BLOSUM matrix family (BLOcks SUbstitution Matrix), which estimates the frequencies of amino acid substitutions from pair occurrences in alignments. BLOSUM matrices are denoted as BLOSUMXX, where XX is the sequence identity threshold that is used to filter the alignments. For example, BLOSUM50 is the BLOSUM matrix created by alignments with sequence identity of 50% and above.

Equation 1.1 describes how to compute a BLOSUM log odds substitution matrix. The indices i and j describe the residues in the alignment, p is the background frequency, and $M_{i,j}$ is the probability of transition from i to j.

$$S_{i,j} = \log \frac{p_i M_{i,j}}{p_i p_j} = \log \frac{M_{i,j}}{p_j} = \log \frac{\text{Observed Frequency}}{\text{Expected Frequency}} \tag{1.1}$$

A reduced substitution matrix groups amino acids by similarity. This means that the alphabet describing the residues of each sequence has less characters than the customary 20 letters. Working in a reduced alphabet can increase the speed of methods or increase the sensitivity for homology search. Several reduction schemes have been proposed over the years (Murphy, Wallqvist, and Levy 2000; Li et al. 2003; Edgar 2004; Peterlongo et al. 2008).

Sequence alignment

A pairwise alignment can be thought of as a table in which each row refers to a single sequence and each column contains homologous characters with respect to the common ancestor of the sequences (Kumar and Filipski 2007). The number of possible alignments between a pair of sequences is vast and thus algorithms to compute a single (optimal under specific criteria) alignment were proposed. One pioneering algorithm is due to Levenshtein (Levenshtein 1966). His algorithm computed an edit distance between a pair of sequences. Edit distance measures the edits needed to transform one sequence into another. This algorithm, which was originally designed as an error correction method for binary signals, considers three single-character edit operations: substitution, deletion and insertion.

Equation 1.2 presents the Levenshtein score computation, which minimizes the number of edits (operations) to transform one sequence (indexed by i) to another (indexed by j). If the character at position i is equal to j, then this is considered a match, if not - a mismatch.

The runtime of optimization is $O(N^2)$, where each of the aligned sequences is of length N. Backurs and Indyk (2014) (Backurs and Indyk 2014) prove that $O(n^{2-\epsilon})$ is also the lower bound of this problem, *iff.* $P \neq NP$.

$$M(i,j) = \min \begin{cases} M(i-1,j-1)+0 & \text{Match} \\ M(i-1,j-1)+1 & \text{Mismatch} \\ M(i-1,j)+1 & \text{Deletion} \\ M(i,j-1)+1 & \text{Insertion} \end{cases} \tag{1.2}$$

Equation 1.2: Levenshtein algorithm to compute the edit distance between two sequences. Every edit (Mismatch/Deletion/Insertion) is penalized with a cost of 1.

Later, Needleman and Wunsch (1970) (Needleman and Wunsch 1970) introduced an optimal global alignment algorithm to the field of computational biology. The Needleman-Wunsch dynamic programming equation 1.3 is based on a principle similar to the Levenshtein algorithm. The difference lies in the consideration of substitutions and an arbitrary gap penalty function f. Compared to computing the edit distance, the Needleman-Wunsch algorithm maximises the score, though Sellers showed (Sellers 1974) that finding the maximum or minimizing the edit distance is equivalent.

$$M(i,j) = \max \left\{ \begin{array}{ll} M(i-1,j-1)+\ w(a_i,b_j) & \text{Match or Mismatch} \\ \max_{1\leq k\leq i}\{M(i-k,j)+\ f(k)\} & \text{Deletion} \\ \max_{1\leq l\leq j}\{M(i,j-l)+\ f(l)\} & \text{Insertion} \end{array} \right\},\ 1 \leq i \leq m, 1 \leq j \leq n \tag{1.3}$$

Equation 1.3: Needleman-Wunsch global alignment algorithm optimally maximizes the matching score between two sequences. Match/Mismatch are expressed by a substitution function w, Deletions and Insertions are penalized by a function f.

An underlying assumption in computing a global alignment is that the sequences are homologous to each other from start to end. This assumption is less likely to hold with greater evolutionary timescales as reflected in reduced global alignment scores. Since, for example, protein sequence domains can be shuffled, or non-conserved parts can diverge. Therefore, local alignments are often more biologically relevant. Smith and Waterman (Smith and Waterman 1981) presented a method in 1981 to detect optimal matching subsequences. This type of alignment is called local alignment. It applies the same dynamic programming principles as the Needleman-Wunsch algorithm, but it "restarts" the alignment every time the score becomes negative. It does so by introducing an additional zero value (equation 1.4), resulting in scores that can never be negative. Smith and Waterman have shown that introducing this change leads to locally optimal solution.

A year later in 1982, Gotoh (Gotoh 1982) introduced a way to model affine gap costs, that is, choosing the gap open and gap extend penalties independently. Affine gap costs lead to a higher sensitivity, since a consecutive block of characters that appear in one sequence

$$H(i,j) = \max \left\{ \begin{matrix} 0 & \\ H(i-1,j-1) + s(q_i,t_j) & \text{Match or Mismatch} \\ \max_{k\geq 1}\{H(i-k,j) + W_k\} & \text{Deletion} \\ \max_{l\geq 1}\{H(i,j-l) + W_l\} & \text{Insertion} \end{matrix} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n \tag{1.4}$$

Equation 1.4: The Smith-Waterman algorithm computes a local alignment between two sequences. An additional zero condition in the maximisation of the dynamic programming formula, forces a restart of the alignment, leading to a locally optimal solution.

but not the other are more likely to be a result of a single event than several independent ones.

Speeding up the Gotoh algorithm

The Gotoh algorithm is the most sensitive sequence-sequence comparison method, since it computes an optimal solution. Its major drawback is its time complexity of $O(N^2)$. Current protein databases have roughly 10^8 entries with 350 amino acid residues as the average entry. Searching an average length protein against these databases would result in roughly $10^8 \times 350 \times 350$ character comparisons. Assuming that each cell can be computed in a single cycle (in reality it is more than one), running these comparisons on a single 3 GHz CPU would result in a runtime of more than 60 minutes.

Besides heuristic algorithm optimisation such as banded alignments (Chao, Pearson, and Miller 1992) hardware optimizations efforts have been made to speed up the Gotoh algorithm (Wozniak 1997; Rognes 2011; Zhao et al. 2013; Farrar 2007; Li, Shum, and Truong 2007; Liu, Maskell, and Schmidt 2009; Manavski and Valle 2008; Rognes and Seeberg 2000; Szalkowski et al. 2008). These solutions often involve Single Instruction Multiple Data (SIMD) approaches, which are CPU or GPU extensions to process several data units with a single instruction. The crucial bottleneck is to overcome is the data dependency among matrix cells in the dynamic programming equations (see figure 1.4).

Seed-and-extend

Between the release of the first Intel Pentium CPU in March 1993 and the last Pentium III CPU in Juli 2001, the CPU clock speed increased 20 times, from 66 MHz to 1.4 GHz. Over the same time span the NCBI Genbank increased over 100 times in size from about 100 000 protein sequences to over 10 million. Altschul et al. (Altschul et al. 1990) were early to recognize the computational issue associated with exact sequence comparisons and introduced BLAST (Basic Local Alignment Search Tool), a heuristic method to speed up sequence searches. BLAST became the most widly adapted search method with currently over 70 000 citations. It introduced for the first time the seed-and-extend technique, which became the standard for heuristic sequence search methods (Buchfink, Xie, and Huson 2015;

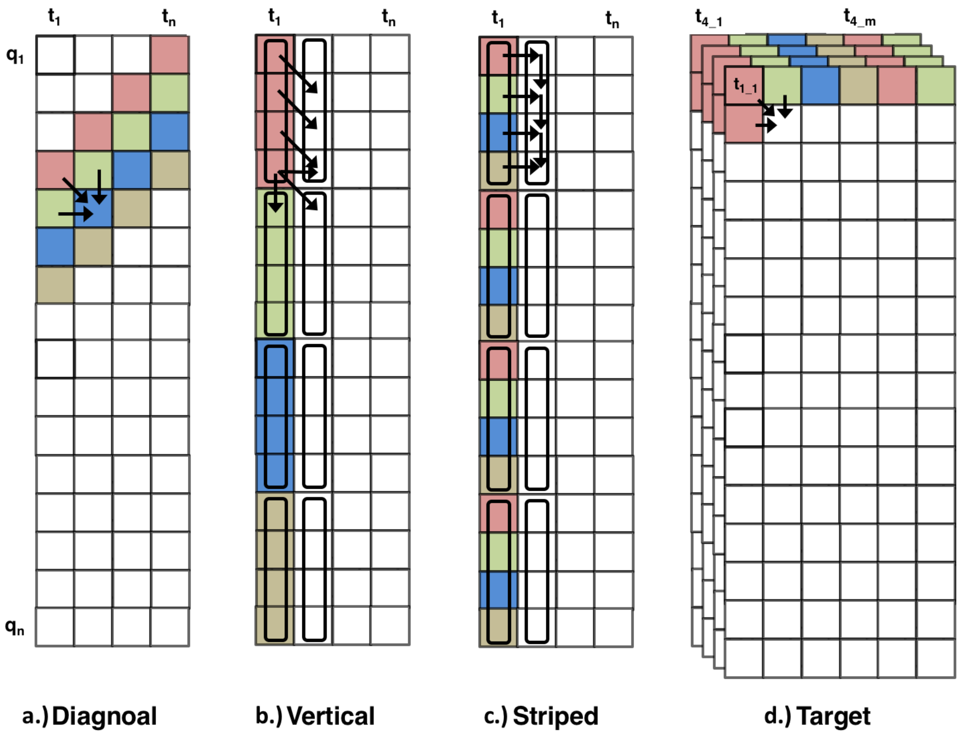


Fig. 1.4. Speeding up the Gotoh algorithm. Solving the data dependency within the dynamic programing equations is the main challenge for acceleration, to effectively use current hardware support for processing multiple data units with a single instruction (SIMD). The figure shows from left to right four possible solutions to solve the data dependency. In each of these solutions the matrix cells that are computed in parallel are shown with the same color. Diagonalization (**a**) of the parallelization was the first approach, chosen by Wozniak (Wozniak 1997). Rognes and Seeberg (**b**) proposed to span the vectors along the query sequence (Rognes and Seeberg 2000). Farrar (**c**) proposed later a striped version which solves the majority of data dependencies (Farrar 2007). In 2011, Rognes (**d**) used an inter sequence approach (Rognes 2011). In this approach the data dependencies are completely solved. However the aligned sequences need to have a similar length. Figure was taken from master thesis (Steinegger 2014).

Hauswedell, Singer, and Reinert 2014; Tan et al. 2012; Zhao, Tang, and Ye 2012; Ye, Choi, and Tang 2011; Edgar 2010; Kiełbasa et al. 2011).

Seed-and-extend is the idea of a fast detection of single *k*-mer (short words of length *k*) matches, which are then extended to full gapped alignments. The main acceleration of this heuristic stems from the notion that most targets are not homologous to the query and thus, computational operations can be saved if those are quickly filtered out. In accordance, the BLAST algorithm will continue computation only for pairs that meet a minimal similarity criterion. This criterion is defined as having two non-overlapping matches on the same diagonal. The k-mer matches in query-target pairs that met this criterion are next extended.

BLAST implemented this seed-and-extend in the following way: First, the seed stage (or *k*-mer prefilter) generates a list of all overlapping *k*-mers from the query sequence. For each *k*-mer, similar *k*-mers are computed by enumerating all possible *k*-mers that have a higher substitution score than the given threshold. All *k*-mers are indexed as a finite state machine following the Aho-Corasick algorithm (Aho and Corasick 1975). This query state machine runs in linear time over all target sequences to detect *k*-mer matches. BLAST stores the diagonal on which a *k*-mer match occurred. The diagonal is defined as $i-j$, where i is the *k*-mer position in the query and j is the position in the target. Afterwards, each sequences with at least two non-overlapping *k*-mer matches on the same diagonal are extended. Both sides (prefix, suffix) of the *k*-mer match are extended, until the total substitution score of the extended segment is negative. All extensions above a certain score threshold are saved as possible hits. If multiple overlapping matches are detected, then these matches are merged into one alignment.

A 20-30% faster way of seeding was proposed in 2007 (Shiryev et al. 2007). The idea was to use a reduced alphabet of size 10 and a *k*-mer length of 7 and using a hash based index, instead of using a finite state machine. Longer *k*-mers have a higher specificity and create therefore less random database matches. While a reduced alphabet increases the sensitivity of a *k*-mer. They optimized both alphabet size and *k*-mer length to achieve a better signal to noise ratio and therefore avoiding more non-homologous extensions. In this work we chose to explore the approach of longer *k*-mers.

Iterative sequence profile searches

The heuristic formulation of BLAST, as described above, is indeed faster than the full Gotoh computation while being just slightly less sensitive. To increase sensitivity, PSI-BLAST (Position-Specific Iterated BLAST), a more sensitive albeit slower variant of the algorithm, was developed. It was the first method to introduce the concept of iterative profile searches (Altschul et al. 1997). The major difference between BLAST and PSI-BLAST is how similarities between *k*-mers are computed. Instead of using a normal substitution matrix it uses a Position Specific Substitutions Matrix (PSSM) which describes the substitution probability at for each query sequence position. PSI-BLAST also supports iterative searches, where the position specific substitution matrix is accumulating more information after each iteration. The first iteration is a normal sequence sequence BLAST search. The input probabilities for the next iteration are computed from a multiple sequence alignment created from the search results. Evolving information improves the sensitivity of detecting remotely homologous proteins. Additional searches are performed as long as new hits are found.

The iterative profile search concept was later applied to more sensitive software, such as the Hidden Markov Model based tools HMMer (Eddy 2009), which uses profiles as query or target database or HHblits (Remmert et al. 2011), which searches with profiles through profile databases represented as multiple sequence alignments.

Multiple Sequence Alignments

A multiple sequence alignment is performed on more than two sequences at a time. Figure 1.5 illustrates a multiple sequence alignment of transmembrane proteins. Unlike with pairwise alignments, finding the optimal multiple sequence alignment was shown to be an NP-complete problem (Wang and Jiang 1994). Two equally long sequence can be aligned in $O(N^2)$, where N is the length of the sequence. Finding the optimal solution for M sequences would require $O(N^M)$. Thus, proposed solutions to the multiple alignment problem are all heuristic in nature. One class of solutions are the progressive alignment methods (Hogeweg and Hesper 1984), which try to make this problem computationally feasible. These methods first compute a binary guide-tree based on sequence distance and then perform pairwise sequences alignments, while traversing the tree bottom-up. The progressive approach is implemented as part of the state-of-the-art algorithms for multiple sequence alignment T-Coffee (Notredame, Higgins, and Heringa 2000), MAFFT (Katoh et al. 2002), Kalign2 (Lassmann, Frings, and Sonnhammer 2009) and Clustal Omega (Sievers et al. 2011).

```
G1SJM8_2 YTSAGVSVTVQ----ELFRVPVLRASSPLPLQEGSAVTLSCETKLLPDSPPLRLYFS
Q8R142_1 YTSAGVSITVKAFPLELFTTPVLRASVSSPFPEGSLVTLNCETNLLLQRPGLQLYFS
P12314_2 YTSAGISVTVK----ELFPAPVLNASVTSPLLEGNLVTLSCETKLLLQRPGLQLYFS
H2PZV3_1 YTSAGISVTVK----ELFPAPVLNASVTSPLLEGNLVTLSCETKLLLQRPGLQLYFS
         *****:*:**:    *** .***.**   *  **. ***.***:** : * *:****
```

Fig. 1.5. Multiple sequence alignment of transmembrane proteins. Every line contains is a single homologe protein. The last row describes indicates how conserved each sequence column is, if the column contains only one letter it is marked with a star character *.

The state of the art in search methods

The majority of state of the art homology search methods use the seed-and-extend technique (introduced in section 1.2). Most of the speedup results from the seed stage. Seeding can reduce the search space by five to six orders of magnitude, from 10^8 to just a few hundred hits that have to be optimally aligned.

Fastest search methods apply common seeding techniques to speed up the homology search: they use index structures like hashing, suffix arrays or sorted lists to index the k-mers of the target database instead of the query sequences like BLAST (Altschul et al. 1997). Some index on both the query and the target side (Buchfink, Xie, and Huson 2015; Hauswedell, Singer, and Reinert 2014).

Some tools increase sensitivity by allowing one or two mismatches in the k-mer (Hauswedell, Singer, and Reinert 2014; Kent 2002), others employ reduced alphabets (Buchfink, Xie, and Huson 2015; Hauswedell, Singer, and Reinert 2014; Tan et al. 2012; Zhao, Tang, and Ye 2012; Ye, Choi, and Tang 2011). Many methods avoid strongly overlapping *k*-mers by using spacing in the *k*-mer (Keich et al. 2004). Spaced *k*-mers are non-continuous *k*-mers with positions, which are excluded from the comparison ("don't care" positions). The spaced *k*-mer pattern `1101` would consider the residues at position 0, 1 and 3 in the k-mer comparison, but would disregard the residue at position 2. The following chapter describes some of the state of the art search methods in detail.

UBLAST

UBLAST and USEARCH are both search algorithms in the USEARCH software suite (Edgar 2010). USEARCH is designed to detect high similarities while UBLAST aims to identify remote homologs. UBLAST uses spaced *k*-mers in a reduced alphabet. The exact implementation details of the UBLAST algorithm were not made public by its developers.

LAST

LAST (Kiełbasa et al. 2011) is a software for protein and nucleotide alignments. It follows the seed-and-extend strategy. The seeding uses a suffix array index structure for the target sequences. A suffix array contains every possible suffix of the database. It is therefore possible to vary the k-mer length, which is an advantage of the LAST algorithm. Each position of the query sequence is extended through lookups in the suffix array. Extending the seeds leads to a higher specificity and therefore reducing the number of target sequence hits. Each seed is extended until only $\leq m$ target sequence hits are left containing this suffix, where m is a parameter which can be adjusted by the user. All m hits are then aligned using the banded Gotoh alignment. The runtime of LAST for one query is almost independent of the target database size. However, the sensitivity is not constant. LAST is the only search tool that guarantees a similar runtime per query at varying database sizes.

RAPsearch2

RAPsearch2 (Zhao, Tang, and Ye 2012) is a seed-and-extend protein search method. It uses a reduced alphabet of size 10 and stores all overlapping 6-mers in a collision-free hash table. For each 6-mer of the query, a lookup in the hash table is done, and later extended. All hits over a certain extension threshold are locally aligned with the Gotoh algorithm.

DIAMOND

DIAMOND (Buchfink, Xie, and Huson 2015) is yet another seed-and-extend approach. The seeding takes advantage of a double indexing strategy to detect matches. It uses two lexicographically sorted lists for the query and target sequences of all overlapping spaced *k*-mers in a reduced alphabet of size 11. Next, the two lists are accessed co-linearly and common seeds are detected between query and target. By default, DIAMOND repeats this seeding strategy for four different spaced *k*-mer patterns. All matched seeds are extended using a hardware accelerated banded alignment algorithm following a similar approach as SWIPE (Rognes 2011), presented in Figure 1.4 **d**.

1.3 Sequence Clustering

Clustering is the process of discovering and grouping of structures in data, based on similarity measures. The groups of similar objects found are called clusters and the group assignment process is called clustering.

There are three major clustering types: (1) partitioning clustering, which clusters the data in a predefined number of clusters k, with k-means (MacQueen 1967) as the most important representative of this type, (2) hierarchical clustering, which builds a hierarchy of clusters, for example through linkage, or (3) density-based clustering, which clusters all elements that are at most ϵ distant from each other such as DBSCAN (Ester et al. 1996).

In the following text, I use the term "clustering" to refer to density-based approaches. The relatedness of protein sequences is usually defined by alignment attributes, such as sequence identity and sequence length overlap between the aligned sequences. Vector clustering methods rely on extracting a set of numerical features to describe each object (data point) to be clustered and then computing a metric between data points (e.g., Euclidean distance). However in the protein homology space, the distance between two non-related, or evolutionary diverged protein sequences cannot be measured sensibly (Rost 1999). Therefore, classical clustering approaches, such as DBSCAN, are not a good fit for proteins. Approaches such as greedy incremental clustering became the standard. The set of proteins to be clustered is compared to an initially empty set of seed sequences. The algorithm tries to assign each sequence to one of the clusters. If the clustering criteria are fulfilled, then the sequence will be assigned to this cluster; if the sequence is not accepted by any existing seed sequence, then the sequence becomes a new seed sequence.

Sequence clustering, next to homology search, became one of the first steps to analyse large amounts of sequencing data. It can, for example, be used to reduce redundancy, thus speeding up downstream analysis. This way, highly similar sequences are represented by a single sequence, the cluster centroid. Clustering can also be used to define new protein families, especially when clustering at lower sequence identities (often less than 50%). This clustering is often done down to sequence identities of less than 50%. Each such protein family can be represented as a multiple sequence alignment, which can be used to create profile a database to drive homology search (Remmert et al. 2011).

The state of the art in clustering methods

State of the art clustering methods and their runtime.

Homology Search and Markov Clustering

A popular way to cluster protein sequences is to combine a homology search method with Markov clustering (Ester et al. 1996). The first step is to perform an all-against-all comparison with a given homology search method. The resulting hits of this search describe a graph. The nodes of this graph are sequences, which are connected by an edge, if they fulfill the clustering criteria such as sequence identity and sequence length overlap. This resulting similarity graph is clustered by Markov clustering. The quality of this clustering is high, since the clustering method can use all possible connections (edges) and can therefore optimize globally.

The most important drawback is the runtime complexity of $O(N^2)$, where N is the number of sequences.

BLASTclust

Another commonly used approach is BLASTclust (Altschul et al. 1990; Altschul et al. 1997), which is part of the BLAST package. BLASTclust uses a connected component clustering, which operates on a homology graph. The first step is also an all-against-all search and graph construction. The graph is clustered by iteratively clustering connected components (subgraphs). All member of the subgraph build a cluster. BLASTclust is prone to errors since a single wrong edge in the graph could connect clusters of, for example, two different protein families.

Runtime of BLASTclust is $O(N^2)$, where N are the number of sequences.

CD-HIT

CD-HIT was first published in 2001 (Li, Jaroszewski, and Godzik 2001) and further improved in 2002 (Li, Jaroszewski, and Godzik 2002). CD-HIT introduced a greedy clustering approach. It is an incremental clustering. All sequences are sorted by length and processed one by one. Each sequence is compared against a set of seed sequences, which is initially empty. The first sequence will automatically become a seed. The next sequence is compared against the seed set. The sequence will be assigned to the first seed sequence that fulfills the clustering criteria. If the sequence does not fulfill the criteria to belong to any of the existing clusters, it becomes a new seed. In addition, CD-HIT offers a more accurate but slower mode, in which the sequence is not assigned to the first cluster that satisfies the criteria but rather to the cluster that fulfills the criteria and has the highest sequence identity.

CD-HIT applies a fast k-mer based prefilter followed by a global alignment. The k-mer size has to be chosen by the user based on the sequence identity of each cluster. The lower the sequence identity is, the smaller the k-mer length should be. At 50% sequence identity CD-HIT recommends 2-mers. Smaller k-mer sizes increase the sensitivity of the software but also increase the runtime significantly, due to non-homologous random matches.

The runtime of CD-HIT is $O(NK)$, where N is the number of sequence and K is the number of seed sequences.

UCLUST

UCLUST is a clustering method, which is part of the USEARCH package (Edgar 2010). UCLUST was built to be a faster version of the CD-HIT algorithm. It also follows the greedy clustering strategy. Its reduced speed is a result of a faster prefilter. UCLUST uses the USEARCH algorithm. The prefilter sorts hits by frequency of 5-mer occurrence. Closely related sequences are more likely to share a high fraction same 5-mers. UCLUST computes a global alignment for the hits in descending k-mer occurrence order util the clustering criteria is fulfilled. USEARCH is closed source software and the exact implementation is unknown.

The runtime complexity is $O(NK)$.

kClust

kClust (Hauser, Mayer, and Söding 2013) was designed to cluster sequences with very low sequence identities. It is the predecessor to MMseqs (Hauser, Steinegger, and Söding 2016) and shares some concepts. It consists of two stages: the prefilter and the alignment stage. kClust also implements the greedy clustering algorithm used in CD-HIT (Li, Jaroszewski, and Godzik 2001). Input sequences are sorted by length and then searched against the seed sequences. The main difference between CD-HIT and kClust is the prefilter. The kClust k-mer size of the prefilter is independent of the target sequence identity. kClust does not only use exact 6-mers, but also computes similar 6-mers based on a substitution matrix. Considering similar and longer k-mers helps to keep specificity and sensitivity high. A heuristic is applied to speed up the alignment. The k-mer matches between two sequences are combined to a gapped alignment using dynamic programming, without the need to realign the sequences. The major drawback of kClust is the fact that it is missing support for multiple CPU cores.

The runtime of kClust is $O(NK)$, similar to that of CD-HIT and UCLUST.

1.4 Overview of this Work

In Chapter 2, I describe *MMseqs* (Hauser, Steinegger, and Söding 2016), a method to search and cluster huge protein sets down to 30% sequence identity. It is a flexible software suite, through its modular architecture. The three core modules of the software are (1) the prefilter, which scans a target database very fast for potential hits, (2) the striped vectorized Gotoh alignment and (3) the greedy set cover (Chvatal 1979) clustering module.

In Chapter 3, I describe *MMseqs2* (Steinegger and Söding 2017), a new method for homology search. MMseqs2 builds upon MMseqs and is designed for fast sequence clustering and search for globally alignable sequences. The fast *k*-mer based prefilter algorithm of MMseqs2 improved the sequence search sensitivity and speed. The prefilter detects local double *k*-mer matches and extends the hits by an ungapped alignment. Through utilizing the L1 cache to detect the local double matches, it scales nearly linearly with increase of CPU cores. MMseqs2 is the first fast sequence search method that supports iterative profile searches. This way, MMseqs2 reaches sensitivity above PSI-BLAST at hundreds of times its speed.

In Chapter 4, I describe *Linclust* (Steinegger and Söding 2018), a new method designed to cluster protein sets in linear time. Previous methods to cluster databases require $O(NK)$ runtime, where N is the size of protein sequences and K is the number of clusters. For huge protein sets, this leads to a near quadratic runtime, since K will become very large. Linclust is the first method that can cluster protein sets in linear time. The major principle of Linclust is to pick one representative sequence in an early stage to reduce the number of computations. Linclust runs four orders of magnitude faster than the current state of the art methods UCLUST and CD-HIT.

In Chapter 5, I present a novel clustered database called Uniclust (Mirdita et al. 2016). Uniclust contains the clustered databases Uniclust90, Uniclust50 and Uniclust30, which are clustered versions of the UniProt database at 90%, 50%, and 30% sequence identity respectively. In that paper we demonstrate that Uniclust produces more consistent clusters compared to UniRef90 and UniRef50.

In Chapter 6, I present a fast and accurate functional distance measure using sequence identity and alignment length which we term ”Homology-derived Functional Similarity of Proteins” (HFSP). HFSP (Mahlich et al. 2018) can annotate enzyme proteins on the fourth level of the Enzyme Commission category (Bairoch 2000) at a 83% accuracy and reaches a 40-fold speed up over the previously published method HSSP (Rost 2002).

The final chapter, the Chapter 7, summarizes the dissertation and discusses future developments.

1.5 References

Aho, A. V. and M. J. Corasick (1975). “Efficient string matching: an aid to bibliographic search”. In: *Communications of the ACM* 18.6, pp. 333–340. doi: 10.1145/360825.360855.

Altschul, S. F. et al. (1990). “Basic local alignment search tool.” In: *J. Mol. Biol.* 215.3, pp. 403–410. doi: 10.1006/jmbi.1990.9999.

Altschul, S. F. et al. (1997). “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.” In: *Nucleic acids research* 25.17, pp. 3389–402.

Apweiler, R. et al. (2004). “UniProt: the Universal Protein knowledgebase.” In: *Nucleic Acids Res.* 32.Database issue, pp. D115–D119. doi: 10.1093/nar/gkh131.

Backurs, A. and P. Indyk (2014). “Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)”. In: *arxiv*.

Bairoch, A. (2000). “The ENZYME database in 2000.” In: *Nucleic acids research* 28.1, pp. 304–5.

Bairoch, A. et al. (2005). “The Universal Protein Resource (UniProt)”. In: *Nucleic Acids Res.* 33.suppl 1, pp. D154–D159. doi: 10.1093/nar/gki070.

Banik, J. J. and S. F. Brady (2010). “Recent application of metagenomic approaches toward the discovery of antimicrobials and other bioactive small molecules”. In: *Current Opinion in Microbiology* 13.5, pp. 603–609. doi: 10.1016/J.MIB.2010.08.012.

Bromberg, Y. and B. Rost (2007). “SNAP: Predict effect of non-synonymous polymorphisms on function”. In: *Nucleic Acids Research* 35, pp. 3823–3835. doi: 10.1093/nar/gkm238.

Buchfink, B., C. Xie, and D. H. Huson (2015). “Fast and sensitive protein alignment using DIAMOND”. In: *Nature Methods* 12.1, pp. 59–60.

Burstein, D. et al. (2016). “New CRISPR–Cas systems from uncultivated microbes”. In: *Nature* 542.7640, pp. 237–241. doi: 10.1038/nature21059.

Chao, K. M., W. R. Pearson, and W. Miller (1992). “Aligning two sequences within a specified diagonal band.” In: *Computer applications in the biosciences : CABIOS* 8.5, pp. 481–7.

Chvatal, V. (1979). *A Greedy Heuristic for the Set-Covering Problem.* doi: 10.2307/3689577.

Cook, C. E. et al. (2018). “The European Bioinformatics Institute in 2017: data coordination and integration”. In: *Nucleic Acids Research* 46.D1, pp. D21–D29. doi: 10.1093/nar/gkx1154.

Corlan, A. D. (2018). “Medline trend: automated yearly statistics of PubMed results for any query.” In:

Dayhoff, M. O. and R. M. Schwartz (1978). “A model of evolutionary change in proteins”. In: *IN ATLAS OF PROTEIN SEQUENCE AND STRUCTURE*.

Delmont, T. O. et al. (2011). “Accessing the Soil Metagenome for Studies of Microbial Diversity”. In: *Applied and Environmental Microbiology* 77.4, pp. 1315–1324. DOI: 10.1128/AEM.01526-10.

Desai, N. et al. (2012). “From genomics to metagenomics”. In: *Curr. Opin. Biotechnol.* 23.1, pp. 72–76.

Eddy, S. R. (2009). “A new generation of homology search tools based on probabilistic inference.” In: *Genome informatics. International Conference on Genome Informatics* 23.1, pp. 205–11.

Edgar, R. C. (2004). “Local homology recognition and distance measures in linear time using compressed amino acid alphabets”. In: *Nucleic Acids Research* 32.1, pp. 380–385. DOI: 10.1093/nar/gkh180.

— (2010). “Search and clustering orders of magnitude faster than BLAST.” In: *Bioinformatics* 26.19, pp. 2460–2461. DOI: 10.1093/bioinformatics/btq461.

Ester, M. et al. (1996). *A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise*.

Farrar, M. (2007). “Striped Smith-Waterman speeds database searches six times over other SIMD implementations.” In: *Bioinformatics (Oxford, England)* 23.2, pp. 156–61. DOI: 10.1093/bioinformatics/btl582.

Field, C. B. et al. (1998). “Primary production of the biosphere: integrating terrestrial and oceanic components”. In: *Science (New York, N.Y.)* 281.5374, pp. 237–40.

Finn, R. D. et al. (2016). “The Pfam protein families database: towards a more sustainable future”. In: *Nucleic Acids Res.* 44.D1, pp. D279–D285.

Gans, J., M. Wolinsky, and J. Dunbar (2005). “Computational Improvements Reveal Great Bacterial Diversity and High Metal Toxicity in Soil”. In: *Science* 309.5739, pp. 1387–1390. DOI: 10.1126/science.1112665.

Gilbert, J. A., J. K. Jansson, and R. Knight (2014). “The Earth Microbiome project: successes and aspirations.” In: *BMC biology* 12, p. 69. DOI: 10.1186/s12915-014-0069-1.

Goldberg, T. et al. (2014). “LocTree3 prediction of localization.” In: *Nucleic acids research* 42.Web Server issue, W350–5. DOI: 10.1093/nar/gku396.

Gotoh, O. (1982). “An improved algorithm for matching biological sequences.” In: *Journal of molecular biology* 162.3, pp. 705–8.

Hauser, M., C. E. Mayer, and J. Söding (2013). “kClust: fast and sensitive clustering of large protein sequence databases.” In: *BMC bioinformatics* 14.1, p. 248. DOI: 10.1186/1471-2105-14-248.

Hauser, M., M. Steinegger, and J. Söding (2016). “MMseqs software suite for fast and deep clustering and searching of large protein sequence sets”. In: *Bioinformatics* 32.9, pp. 1323–1330.

Hauswedell, H., J. Singer, and K. Reinert (2014). “Lambda: the local aligner for massive biological data”. In: *Bioinformatics* 30.17, pp. i349–i355.

Hayden, E. C. (2014a). *HiSeq X Ten — Thousand dollar genome sequencer — Illumina*. (Visited on).

— (2014b). “Is the $1,000 genome for real?” In: *Nature News*. DOI: `doi:10.1038/nature.2014.14530`.

Henikoff, S. and J. G. Henikoff (1992). “Amino acid substitution matrices from protein blocks.” In: *Proceedings of the National Academy of Sciences of the United States of America* 89.22, pp. 10915–9.

Hogeweg, P. and B. Hesper (1984). “The alignment of sets of sequences and the construction of phyletic trees: an integrated method.” In: *Journal of molecular evolution* 20.2, pp. 175–86.

Howe, A. C. et al. (2014). “Tackling soil diversity with the assembly of large, complex metagenomes.” In: *Proceedings of the National Academy of Sciences of the United States of America* 111.13, pp. 4904–9. DOI: `10.1073/pnas.1402564111`.

Huerta-Cepas, J. et al. (2011). “PhylomeDB v3.0: an expanding repository of genome-wide collections of trees, alignments and phylogeny-based orthology and paralogy predictions.” In: *Nucleic acids research* 39.Database issue, pp. D556–60. DOI: `10.1093/nar/gkq1109`.

Huson, D. H. et al. (2007). “MEGAN analysis of metagenomic data.” In: *Genome research* 17.3, pp. 377–86. DOI: `10.1101/gr.5969107`.

IHGSC (2001). “The sequence of the human genome”. In: *Science* 291.5507, pp. 1304–51. DOI: `10.1126/science.1058040`.

Jensen, L. J. et al. (2007). “eggNOG: automated construction and annotation of orthologous groups of genes”. In: *Nucleic Acids Research* 36.Database, pp. D250–D254. DOI: `10.1093/nar/gkm796`.

Jones, D. T. (1999). “Protein secondary structure prediction based on position-specific scoring matrices.” In: *Journal of molecular biology* 292.2, pp. 195–202. DOI: `10.1006/jmbi.1999.3091`.

Kanehisa, M. and S. Goto (2000). “KEGG: Kyoto Encyclopedia of Genes and Genomes”. In: *Nucleic Acids Res.* 28.1, pp. 27–30. DOI: `10.1093/nar/28.1.27`.

Karsenti, E. (2015). “The making of Tara Oceans: funding blue skies research for our Blue Planet.” In: *Molecular systems biology* 11.5, p. 811. DOI: `10.15252/MSB.20156271`.

Katoh, K. et al. (2002). “MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform.” In: *Nucleic acids research* 30.14, pp. 3059–66.

Keich, U. et al. (2004). “On spaced seeds for similarity search”. In: *Discrete Applied Mathematics* 138.3, pp. 253–263. DOI: `10.1016/S0166-218X(03)00382-2`.

Kent, J. J. (2002). “BLAT–the BLAST-like alignment tool.” In: *Genome Res* 12.4, pp. 656–664. DOI: `10.1101/gr.229202`.

Kiełbasa, S. M. et al. (2011). “Adaptive seeds tame genomic sequence comparison.” In: *Genome research* 21.3, pp. 487–93. DOI: `10.1101/gr.113985.110`.

Kouranov, A. et al. (2006). “The RCSB PDB information portal for structural genomics.” In: *Nucleic acids research* 34.Database issue, pp. D302–5. DOI: 10.1093/nar/gkj120.

Kumar, S. and A. Filipski (2007). “Multiple sequence alignment: in pursuit of homologous DNA positions.” In: *Genome research* 17.2, pp. 127–35. DOI: 10.1101/gr.5232407.

Lander, E. (2009). “Moore´s law sequencing”. In: *Broad Institute*.

Lassmann, T., O. Frings, and Erik L. L. Sonnhammer (2009). “Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features”. In: *Nucleic Acids Research* 37.3, pp. 858–865. DOI: 10.1093/nar/gkn1006.

Levenshtein (1966). “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady, Vol. 10, p.707* 10, p. 707.

Levin, B. J. et al. (2017). “A prominent glycyl radical enzyme in human gut microbiomes metabolizestrans-4-hydroxy-l-proline.” In: *Science (New York, N.Y.)* 355.6325, eaai8386. DOI: 10.1126/science.aai8386.

Li, I. T. S., W. Shum, and K. Truong (2007). “160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA).” In: *BMC bioinformatics* 8.1, p. 185. DOI: 10.1186/1471-2105-8-185.

Li, T. et al. (2003). “Reduction of protein sequence complexity by residue grouping”. In: *Protein Engineering Design and Selection* 16.5, pp. 323–330. DOI: 10.1093/protein/gzg044.

Li, W., L. Jaroszewski, and A. Godzik (2001). “Clustering of highly homologous sequences to reduce the size of large protein databases.” In: *Bioinformatics* 17.3, pp. 282–283. DOI: 10.1093/bioinformatics/17.3.282.

— (2002). “Sequence clustering strategies improve remote homology recognitions while reducing search times.” In: *Protein Eng* 15.8, pp. 643–649.

Li, W. et al. (2012). “Ultrafast clustering algorithms for metagenomic sequence analysis.” In: *Briefings in bioinformatics* 13.6, pp. 656–68. DOI: 10.1093/bib/bbs035.

Liu, Y., D. L. Maskell, and B. Schmidt (2009). “CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units.” In: *BMC research notes* 2.1, p. 73. DOI: 10.1186/1756-0500-2-73.

MacQueen, J. (1967). “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, pp. 281–297.

Mahlich, Yannick et al. (2018). “HFSP: high speed homology-driven function annotation of proteins”. In: *Bioinformatics* 34.13, pp. i304–i312.

Manavski, S. A. and G. Valle (2008). “CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment.” In: *BMC bioinformatics* 9 Suppl 2.Suppl 2, S10. DOI: 10.1186/1471-2105-9-S2-S10.

Margulies, M. et al. (2005). “Genome sequencing in microfabricated high-density picolitre reactors.” In: *Nature* 437.7057, pp. 376–80. DOI: 10.1038/nature03959.

Markowitz, V. M. et al. (2014). “IMG/M 4 version of the integrated metagenome comparative analysis system”. In: *Nucleic Acids Res.* 42.D1, pp. D568–D573.

McEntyre, J. and D. J. Lipman (2001). “PubMed: bridging the information gap.” In: *CMAJ : Canadian Medical Association journal = journal de l’Association medicale canadienne* 164.9, pp. 1317–9.

Mirdita, M. et al. (2016). “Uniclust databases of clustered and deeply annotated protein sequences and alignments”. In: *Nucleic Acids Res.* doi: 10.1093/nar/gkw1081.

Muir, P. et al. (2016). “The real cost of sequencing: scaling computation to keep pace with data generation”. In: *Genome Biology* 17.1, p. 53. doi: 10.1186/s13059-016-0917-0.

Murphy, L. R., A. Wallqvist, and R. M. Levy (2000). “Simplified amino acid alphabets for protein fold recognition and implications for folding”. In: *Protein Engineering, Design and Selection* 13.3, pp. 149–152. doi: 10.1093/protein/13.3.149.

Needleman, S. B. and C. D. Wunsch (1970). “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of Molecular Biology* 48.3, pp. 443–453. doi: 10.1016/0022-2836(70)90057-4.

Ng, P. and S. Henikoff (2003). “SIFT: Predicting amino acid changes that affect protein function.” In: *Nucleic acids research* 31.13, pp. 3812–4.

Notredame, C., D. G. Higgins, and J. Heringa (2000). “T-coffee: a novel method for fast and accurate multiple sequence alignment 1 1Edited by J. Thornton”. In: *Journal of Molecular Biology* 302.1, pp. 205–217. doi: 10.1006/jmbi.2000.4042.

Pearson, W. R. (2013). “An introduction to sequence similarity (”homology”) searching.” In: *Current protocols in bioinformatics* Chapter 3, Unit3.1. doi: 10.1002/0471250953.bi0301s42.

Peterlongo, P. et al. (2008). “Optimal neighborhood indexing for protein similarity search”. In: *BMC Bioinformatics* 9.1, p. 534. doi: 10.1186/1471-2105-9-534.

Peterson, J. et al. (2009). “The NIH Human Microbiome Project”. In: *Genome Research* 19.12, pp. 2317–2323. doi: 10.1101/gr.096651.109.

Prakash, T. and T. D. Taylor (2012). “Functional assignment of metagenomic data: challenges and applications”. In: *Brief. Bioinformatics* 13.6, pp. 711–727.

Pruitt, K. D., T. Tatusova, and D. R. Maglott (2007). “NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins”. In: *Nucleic Acids Research* 35.Database, pp. D61–D65. doi: 10.1093/nar/gkl842.

Rappe, M. S. and S. J. Giovannoni (2003). “The uncultured microbial majority”. In: *Ann. Rev. Microbiol.* 57.1, pp. 369–394.

Remmert, M. et al. (2011). “HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment”. en. In: *Nature Methods* 9.2, pp. 173–175. doi: 10.1038/nmeth.1818.

Rodriguez-R, L. M. and K. T. Konstantinidis (2014a). “Estimating coverage in metagenomic data sets and why it matters”. In: *The ISME Journal* 8.11, pp. 2349–2351. DOI: 10.1038/ismej.2014.76.

— (2014b). “Nonpareil: a redundancy-based approach to assess the level of coverage in metagenomic datasets”. In: *Bioinformatics* 30.5, pp. 629–635. DOI: 10.1093/bioinformatics/btt584.

Rognes, T. (2011). “Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation.” In: *BMC bioinformatics* 12.1, p. 221. DOI: 10.1186/1471-2105-12-221.

Rognes, T. and E. Seeberg (2000). “Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors”. In: *Bioinformatics* 16.8, pp. 699–706. DOI: 10.1093/bioinformatics/16.8.699.

Rost, B. (1999). “Twilight zone of protein sequence alignments.” In: *Protein engineering* 12.2, pp. 85–94.

— (2002). “Enzyme Function Less Conserved than Anticipated”. In: *Journal of Molecular Biology* 318.2, pp. 595–608. DOI: 10.1016/S0022-2836(02)00016-5.

Rost, B. and C. Sander (1993). “Prediction of protein secondary structure at better than 70% accuracy.” In: *Journal of molecular biology* 232.2, pp. 584–99. DOI: 10.1006/jmbi.1993.1413.

Sanger, F, S Nicklen, and A R Coulson (1977). “DNA sequencing with chain-terminating inhibitors.” In: *Proceedings of the National Academy of Sciences of the United States of America* 74.12, pp. 5463–7.

Scholz, M. B., C. C. Lo, and P. S. Chain (2012). “Next generation sequencing and bioinformatic bottlenecks: the current state of metagenomic data analysis”. In: *Curr. Opin. Biotechnol.* 23.1, pp. 9–15.

Sellers, P. H. (1974). “On the Theory and Computation of Evolutionary Distances”. In: *SIAM Journal on Applied Mathematics* 26.4, pp. 787–793. DOI: 10.1137/0126070.

Shiryev, S. A. et al. (2007). “Improved BLAST searches using longer words for protein seeding”. In: *Bioinformatics* 23.21, pp. 2949–2951. DOI: 10.1093/bioinformatics/btm479.

Sievers, F. et al. (2011). “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega.” In: *Molecular systems biology* 7.1, p. 539. DOI: 10.1038/msb.2011.75.

Sikic, K. and O. Carugo (2010). “Protein sequence redundancy reduction: comparison of various method.” In: *Bioinformation* 5.6, pp. 234–9.

Smith, T. E. et al. (2018). “Accessing chemical diversity from the uncultivated symbionts of small marine animals”. In: *Nature Chemical Biology* 14.2, pp. 179–185. DOI: 10.1038/nchembio.2537.

Smith, T. and M. Waterman (1981). “Identification of common molecular subsequences”. In: *Journal of Molecular Biology* 147.1, pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5.

Snel, B et al. (2000). “STRING: a web-server to retrieve and display the repeatedly occurring neighbourhood of a gene.” In: *Nucleic acids research* 28.18, pp. 3442–4.

Söding, Johannes (2005). “Protein homology detection by HMM-HMM comparison.” In: *Bioinformatics (Oxford, England)* 21.7, pp. 951–60. DOI: 10.1093/bioinformatics/bti125.

Steinegger, M. (2014). “Accelerated pairwise HMM alignment using SIMD programming and probabilistic secondary structure scoring for remote protein homology detection”. In: *Master thesis*.

Steinegger, M. and J. Söding (2017). “MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets”. In: *Nature Biotechnology* 35, 1026 EP -. DOI: 10.1038/nbt.3988;https://www.nature.com/articles/nbt.3988#supplementary-information.

Steinegger, Martin and Johannes Söding (2018). “Clustering huge protein sequence sets in linear time”. In: *Nature Communications* 9.1, p. 2542. DOI: 10.1038/s41467-018-04964-5.

Suzek, B. et al. (2007). “UniRef: comprehensive and non-redundant UniProt reference clusters.” In: *Bioinformatics* 23.10, pp. 1282–1288. DOI: 10.1093/bioinformatics/btm098.

Szalkowski, A. et al. (2008). “SWPS3 - fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2.” In: *BMC research notes* 1.1, p. 107. DOI: 10.1186/1756-0500-1-107.

Tan, J. et al. (2012). “Tachyon search speeds up retrieval of similar sequences by several orders of magnitude.” In: *Bioinformatics* 28.12, pp. 1645–1646. DOI: 10.1093/bioinformatics/bts197.

Wang, L. and T. Jiang (1994). “On the Complexity of Multiple Sequence Alignment”. In: *Journal of Computational Biology* 1.4, pp. 337–348. DOI: 10.1089/cmb.1994.1.337.

Wetterstrand, KA (2017). “DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)”. In:

Weyrich, L. S. et al. (2017). “Neanderthal behaviour, diet, and disease inferred from ancient DNA in dental calculus”. In: *Nature* 544.7650, pp. 357–361. DOI: 10.1038/nature21674.

Wilke, A. et al. (2016). “The MG-RAST metagenomics database and portal in 2015”. In: *Nucleic Acids Res.* 44.D1, pp. D590–D594.

Wozniak, A (1997). “Using video-oriented instructions to speed up sequence comparison.” In: *Computer applications in the biosciences : CABIOS* 13.2, pp. 145–50.

Ye, Y., J. H. Choi, and H. Tang (2011). “RAPSearch: a fast protein similarity search tool for short reads.” In: *BMC Bioinformatics* 12.1, pp. 159+. DOI: 10.1186/1471-2105-12-159.

Zaremba-Niedzwiedzka, K. et al. (2017). “Asgard archaea illuminate the origin of eukaryotic cellular complexity”. In: *Nature* 541.7637, pp. 353–358. DOI: 10.1038/nature21031.

Zhao, M. et al. (2013). “SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications”. In: *PLoS One* 8.12. DOI: 10.1371/journal.pone.0082138.

Zhao, Y., H. Tang, and Y. Ye (2012). “RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data.” In: *Bioinformatics* 28.1, pp. 125–126. DOI: 10.1093/bioinformatics/btr595.

Chapter 2

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

2.1 Introduction

MMseqs (Hauser, Steinegger, and Söding 2016) (Many-against-Many sequence searching) is a novel algorithm and software to search and cluster large protein sets, containing billion of sequences, making it capable of handling the volumes of data in metagenomic studies.

MMseqs is a modular software suite consisting of three core modules. The prefilter module (1) filters rejects most of non-homologous sequences based on their k-mer composition. It sums up all similar 6-mer scores and estimates homology by applying a z-score statistic. The local alignment module (2) is an implementation of the striped Gotoh algorithm (Farrar 2007). Module (3) of MMseqs can cluster sequences based on a similarity graph. In this graph, nodes are sequences and edges are alignments fulfilling clustering criteria, such as a sequence identity threshold and sequence length overlap. The graph is then clustered by a greedy set cover implementation (Chvatal 1979). This set cover algorithm creates less clusters than the greedy incremental clustering, as implemented in UCLUST (Edgar 2010) or CD-HIT (Li and Godzik 2006). This is one of the key advantages clustering with MMseqs offers over state-of-the-art clustering methods.

The modular architecture of MMseqs allows users creating workflows tailored for specific clustering and homology search tasks. Specifically, we used this modularity to design four workflows for the most common tasks. The clustering workflow clusters the input databases by running modules (1) - (3) with predefined parameters, set based on experiences clustering the Uniprot (Apweiler et al. 2004), which reflect an average case. The second, cascaded clustering, clusters the input database incrementally in multiple steps, increasing the clustering sensitivity in each step. The third workflow compares protein sets to each other by running modules (1) and (2). Finally, the fourth workflow is an updating workflow. It takes a new version of the database that was clustered and a previously clustered database together with its clustering result and updates the clustering by deleting deprecated sequences and adding new sequences to the existing clusters.

In our homology search benchmarks, MMseqs proved to be much more sensitive and 4 to 30 times faster than UBLAST (Edgar 2010) and RAPsearch2 (Zhao, Tang, and Ye 2012). MMseqs can cluster large databases down to 30% sequence identity at around 2000 times the speed of BLASTclust (Altschul et al. 1990) and is more sensitive than CD-HIT (Li and Godzik 2006) or UBLAST (Edgar 2010).

Maria Hauser and Martin Steinegger performed the research and programming, Maria Hauser, Martin Steinegger and Johannes Söding jointly designed the research and wrote the manuscript.

2.2 References

Altschul, S. F. et al. (1990). "Basic local alignment search tool." In: *J. Mol. Biol.* 215.3, pp. 403–410. DOI: 10.1006/jmbi.1990.9999.

Apweiler, R. et al. (2004). "UniProt: the Universal Protein knowledgebase." In: *Nucleic Acids Res.* 32.Database issue, pp. D115–D119. DOI: 10.1093/nar/gkh131.

Chvatal, V. (1979). *A Greedy Heuristic for the Set-Covering Problem.* DOI: 10.2307/3689577.

Edgar, R. C. (2010). "Search and clustering orders of magnitude faster than BLAST." In: *Bioinformatics* 26.19, pp. 2460–2461. DOI: 10.1093/bioinformatics/btq461.

Farrar, M. (2007). "Striped Smith-Waterman speeds database searches six times over other SIMD implementations." In: *Bioinformatics (Oxford, England)* 23.2, pp. 156–61. DOI: 10.1093/bioinformatics/btl582.

Hauser, M., M. Steinegger, and J. Söding (2016). "MMseqs software suite for fast and deep clustering and searching of large protein sequence sets". In: *Bioinformatics* 32.9, pp. 1323–1330.

Li, Weizhong and Adam Godzik (2006). "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences." In: *Bioinformatics* 22.13, pp. 1658–1659. DOI: 10.1093/bioinformatics/btl158.

Zhao, Y., H. Tang, and Y. Ye (2012). "RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data." In: *Bioinformatics* 28.1, pp. 125–126. DOI: 10.1093/bioinformatics/btr595.

2.3 Journal article

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

Sequence analysis

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

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

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

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

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

Abstract

Motivation: Sequence databases are growing fast, challenging existing analysis pipelines. Reducing the redundancy of sequence databases by similarity clustering improves speed and sensitivity of iterative searches. But existing tools cannot efficiently cluster databases of the size of UniProt to 50% maximum pairwise sequence identity or below. Furthermore, in metagenomics experiments typically large fractions of reads cannot be matched to any known sequence anymore because searching with sensitive but relatively slow tools (e.g. BLAST or HMMER3) through comprehensive databases such as UniProt is becoming too costly.
Results: MMseqs (*M*any-against-*M*any *seq*uence *s*earching) is a software suite for fast and deep clustering and searching of large datasets, such as UniProt, or 6-frame translated metagenomics sequencing reads. MMseqs contains three core modules: a fast and sensitive prefiltering module that sums up the scores of similar *k*-mers between query and target sequences, an SSE2- and multi-core-parallelized local alignment module, and a clustering module.
In our homology detection benchmarks, MMseqs is much more sensitive and 4–30 times faster than UBLAST and RAPsearch, respectively, although it does not reach BLAST sensitivity yet. Using its cascaded clustering workflow, MMseqs can cluster large databases down to $\sim$30% sequence identity at hundreds of times the speed of BLASTclust and much deeper than CD-HIT and USEARCH. MMseqs can also update a database clustering in linear instead of quadratic time. Its much improved sensitivity-speed trade-off should make MMseqs attractive for a wide range of large-scale sequence analysis tasks.
Availability and implementation: MMseqs is open-source software available under GPL at https://github.com/soedinglab/MMseqs
Contact: martin.steinegger@mpibpc.mpg.de, soeding@mpibpc.mpg.de
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

During the last 8 years, sequencing costs have come down from 10 000 000$ to less than 1000$ for a human genome at 30 times coverage (http://www.genome.gov/sequencingcosts). As a consequence, protein sequence databases such as the UniProt (Bairoch *et al.*, 2005) database have been growing by a factor of 2 every two years (Apweiler *et al.*, 2004), leading to longer search times, inflated and redundant results list, large memory requirements and saturating or decreasing sensitivities for informative sequence matches (Chubb *et al.*, 2010).

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

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

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

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

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

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

2 Methods

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

2.1 Prefilter module

2.1.1 Prefilter score

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

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

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

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

2.1.2 Z-score

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

2.1.3 Local compositional bias correction

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

2.1.4 Core algorithm

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

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

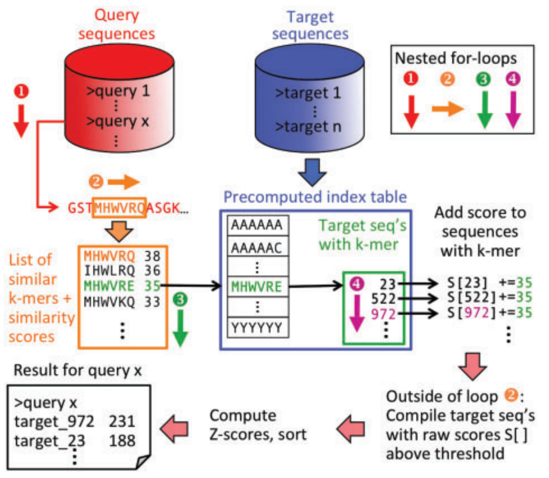


Fig. 1. Core prefiltering algorithm. The algorithm computes for all query-target sequence pairs the score in Eq. (1). See Section 2.1 for details

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

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

2.1.5 Time complexity

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

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

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

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

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

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

2.1.6 Memory requirements

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

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

2.2 Alignment module

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

2.3 Clustering module

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

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

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

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

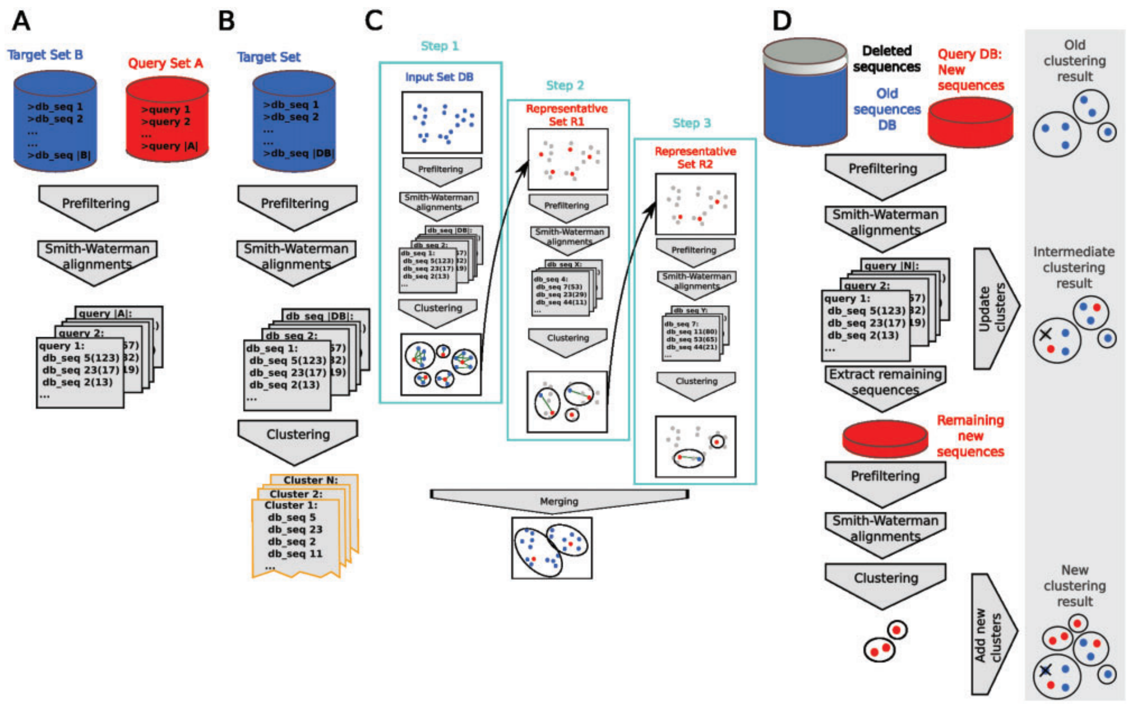


Fig. 2. MMseqs workflows are designed to facilitate the most frequent use cases: (**A**) batch sequence searching with one query set through a target set, (**B**) clustering a set of sequences by similarity and other criteria starting from the raw sequence file, (**C**) an efficient *cascaded clustering*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

2.4 Workflows

2.4.1 Sequence search workflow

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

2.4.2 Clustering workflow

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

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

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

2.4.3 Updating workflow

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

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

3 Results

3.1 Sensitivity and speed of protein searches

3.1.1 Benchmark dataset

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

3.1.2 Benchmarked tools

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

3.1.3 Speed measurements

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

3.1.4 Searching with short peptide reads

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

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

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

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

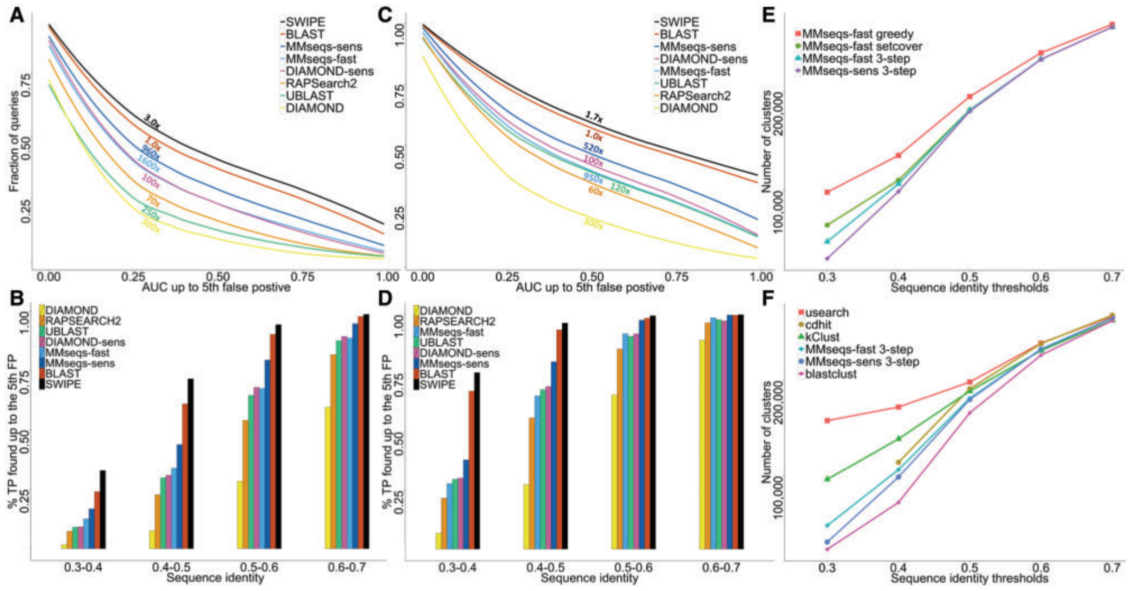


Fig. 3. Sensitivity of sequence search tools and clustering performance. (**A**) AUC5 analysis for short peptide queries: Each of 7616 query fragments of length 50 sampled from SCOP25 was searched against the 283 406 sequences of the scop25db set and the area under the ROC curve up to the fifth false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

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

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

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

3.1.5 Searching with full-length SCOP25 sequences

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

3.2 Clustering performance

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

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

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

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

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

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

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

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

4 Discussion and outlook

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

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

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

Acknowledgement

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

Funding

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

Conflict of Interest: none declared.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–4

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

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

[1]Computational Biology, Max Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077 Göttingen, Germany. [2]Gene Center, Ludwig-Maximilians-Universität München, Feodor-Lynen-Str. 25, 81377 Munich, Germany. [3]TUM, Department of Informatics, Bioinformatics & Computational Biology-I12, Boltzmannstraße 3, 85748 Garching, Germany

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

1 SENSITIVITY AND SPEED OF SEQUENCE SEARCHES

Our goal was to compare the performance and the speed of MMseqs with various other fast tools for protein sequence searching: SWIPE, BLAST, UBLAST, RAPsearch2 and DIAMOND.

For tools that had a limit on the maximum number of matches report, we increased this limit to ensure that at least five false positive will be listed for each query. This is necessary for the calculation of ROC5 values,

All searches were made on a computer with 128 GB RAM and two 8-core Intel Xeon E5-2680 CPUS with 2.70GHz.

MMseqs We used only the prefiltering module and the alignment module of MMseqs for the protein search. We tested two different sensitivities in the prefiltering module, $s = 4$ (default setting) and $s = 7$. Besides, we set the maximum prefiltering list length to 1000 using `--max-seqs 1000`, and the Z-score threshold to 10.0 using `--z-score-thr 10.0` in order to increase the length of the result lists for each query. The alignment module is run with the maximum e-value threshold 10.0 using `-e 10.0` and the alignment coverage is switched off using `-c 0.0`. Both modules use all 16 cores of the machine by default.

Smith-Waterman alignments with SWIPE We use the SSSE3-, multi-core-paralellized Smith-Waterman alignment calculation with SWIPE. In order to get many database matches for a query, we set the e-value to 100 using `-e 100.0` and the number of sequence descriptions and sequence alignments to 1000 using `-v 1000 -b 1000`. Additionally, swipe is instructed to use all the 16 cores of the machine with `-a 16`.

BLAST We ran BLAST using `-e 10.0` and `-v 1000 -b 1000` in order to increase the number of results, and with `-a 16` to parallelize the calculation.

UBLAST We ran UBLAST with `-evalue 10.0` option. Therefore, UBLAST outputs all significant alignments regardless of the sequence identity and alignment coverage. UBLAST uses all available cores per default. Since UBLAST does not have an option to set the maximum number of results shown, we presume that it does not have such a limit.

RAPsearch We ran RAPsearch with with `-z 16` option to paralellize the calculation, and with `-e 10.0` and `-v 1000 -b 1000` options.

DIAMOND We ran DIAMOND with `-e 10.0` option, with `--threads 16` option to paralellize the calculation, and with `--max-target-seqs 1000` option.

2 SEQUENCE CLUSTERING PERFORMANCE

We benchmarked the ability of MMseqs, blastclust, CD-HIT, kClust and USEARCH to cluster sequences based on their global similarity. We benchmarked the clustering performance with different sequence identity thresholds in the range $[0.3 : 0.7]$ in 0.1 increments. All tools were instructed to only merge sequences that had an alignment covering at least a fraction of 0.8 of the residues of both sequences.

All clustering runs (except clustering of UniProt) were made on a computer with 128 GB RAM and two 8-core Intel Xeon E5-2680 CPUS with 2.70GHz. The UniProt was clustered on a computer with 512 GB RAM and four 8-cores CPUs (Intel Xeon CPU E5-4620, 2.20GHz).

2.1 Parameters of tested tools

MMseqs We use the clustering workflow for calculating the clustering of the database. We tested simple and cascaded (option `--cascaded`) clustering each with sensitivity 4 and 7 (`-s 4` and `-s 7`) respectively. For each sensitivity, we set the target clustering sequence identity using the option `--id` and values from 0.3 to 0.7 in 0.1 increments.

blastclust Blastclust is the clustering software in the BLAST package. We set the number of used cores to 16 with the option '`-a16`', length coverage threshold to 0.8 using `-L 0.8` and the minimum sequence identity in the clusters with the option `-S` and values from 30 to 70 in 10 increments.

CD-HIT We set the minimum alignment coverage of the longer sequence to 80% with the `-aL 0.8` option and the number of threads used for the calculation to 16 with the `-T 16` option. The minimum possible clustering sequence identity in CD-HIT is 0.4. For the clustering down to the different minimum sequence identities in the range $[0.4 : 0.8]$, we used the option `-c` for the sequence identity setting and adjusted the k-mer word length with the option `-n`. For the sequence identity 0.4 the word length was set to 2, for the sequence identity 0.5 to 3, for the sequence identity 0.6 to 4 and for the sequence identity 0.7 to 5.

kClust We used the `-s` option to set the minimum sequence identity in the cluster. According to kClust recommendations, we set `-s` to 1.12, 1.73, 2.33, 2.93, 3.53, and 4.14 for the sequence identities 0.3, 0.4, 0.5, .06,

*These authors contributed equally to this work.

†to whom correspondence should be addressed: soeding@mpibpc.mpg.de

	#clusters	#seqs per cluster	#corrupted clusters
MMseqs s=4 greedy clustering	85 780	3.4	1
MMseqs s=4 set cover	60 915	4.7	1
MMseqs s=4 3-step	41 173	7.0	3
MMseqs s=7 greedy clustering	41 572	7.0	3
MMseqs s=7 set cover	29 801	9.7	2
MMseqs s=7 3-step	22 541	12.9	1
blastclust	21 890	13.3	1
CD-HIT	114 386	2.5	260
kClust	91 681	3.2	1
USEARCH	157 981	1.8	11

Table 1. Clustering results on the protein database consisting of SCOP25 and related UniProtKB sequences. Sequences put into the same cluster, but stemming from different folds are considered to be false positives.

and 0.7, respectively. Since kClust is single-threaded, we did not use any parallelization options.

USEARCH We ran USEARCH with -cluster_fast option and set the minimum sequence identity in a cluster to 50% with `--id` option to values from 0.3 to 0.8 in 0.1 increments. We set the query and target sequence coverage in USEARCH to 0.8 using the options `-query_cov 0.8` and `-target_cov 0.8`.

2.2 Cluster quality

We benchmarked the clustering quality by clustering the protein clustering benchmark dataset with the clustering tools MMseqs-sens and MMseqs-fast each with three clustering algorithms (greedy clustering as used in CD-HIT and kClust, set cover and 3-step cascaded clustering), blastclust, CD-HIT, kClust and USEARCH. We clustered the dataset down to 30% sequence identity with each tool, except for CD-HIT, where we used the minimum possible sequence identity threshold of 40%. We use the same method to define false positive sequence pairs as in the other protein search and clustering benchmarks. A cluster is considered as corrupt if it contains at least one false positive sequence pair.

All methods except CD-HIT produce clusters of very high quality with a negligible number of sequences assigned to a cluster by mistake. CD-HIT produced 260 corrupted clusters due to an occasional error in the calculation of the sequence identity. Cascaded clustering and default straight-forward clustering in MMseqs uses set-cover as the default clustering algorithm. We compared the performance of set-cover and the greedy clustering, as used by kClust. Table 1 demonstrates that set-cover performs much better.

2.3 Clustering of the UniProt database

We clustered the UniProt database version containing 54 790 250 sequences with MMseqs and USEARCH, the only two tools that are able to cluster such a large database down to sequence identities of 50% or lower. MMseqs is able to use all 32 cores for the clustering procedure, while USEARCH is able to only use one core.

We use MMseqs cascaded clustering workflow (option `--cascaded`) with default settings to evaluate the clustering procedure.

In USEARCH, we set the lowest sequence identity of clusters to 50%, since it is the lowest recommended value corresponding to the documentation (option `--id 0.5`). We only want to have sequences with pairwise global similarity in one cluster, so we set the query and target sequence coverage in USEARCH to 0.8 using the options `-query_cov 0.8` and `-target_cov 0.8`.

	time	**#clusters**
blastclust	58y	?
MMseqs	8d 17h	6 374 156
USEARCH	11d 2h	9 822 910

Table 2. UniProtKB clustering results: Time and number of clusters for BLAST, MMseqs and USEARCH. kClust, BLAST and kClust times are estimated.

BLAST is much too slow to cluster the UniProtKB database. We estimated the runtime of BLAST clustering using a BLAST run with a small query set against the whole UniProtKB database, and extrapolated the measured runtime to the clustering of the whole UniProtKB database using an all-against-all comparison. We extrapolated that clustering based on all-against-all BLAST using all 32 cores would need about 58 years based on run times for BLAST searches of the UniProt database.

MMseqs requires 8 days and 17 hours and 118 G of memory for the clustering procedure. It produces 6 374 156 clusters, i. e. an average of $8,5$ sequences per cluster. USEARCH, on the other hand, requires, for the same job, 11 days and 2 hours and 42 GB of memory, while it produces 9 822 910 clusters, i. e. an average of $5,5$ sequences per cluster. The results are shown in Table 2.

Although MMseqs is parallelized and uses all 32 cores of the computer and USEARCH is single-threaded, USEARCH runs almost as fast. This is in part explained by the efficiency of the greedy agglomerative clustering algorithm used by USEARCH, which reduces the total number of comparisons from N^2_{seqs} to $N_{\text{clus}}N_{\text{seqs}}$, where N_{seqs} is the number of sequences in the database and N_{clues} is the number of representative sequences, i.e., the number of clusters in the clustered database. However, loosing a factor $N_{\text{seqs}}/N_{\text{clues}}$ in speed over the simple greedy algorithm is more than counterbalanced by the possibility to parallelize the set-cover clustering and by its superior clustering performance in comparison to the simple greedy algorithm.

3 PREFILTERING ALGORITHM DETAILS

3.1 Prefilter Z-score

Expected prefiltering scores between non-homologous sequences will be proportional to the product of both their lengths, since there is a small but non-negligible probability for any pair of query-target k-mers to attain a similarity score above the k-mer cut-off score. It makes sense to correct for the score expected from such background k-mer matches by subtracting it from the actual score. The background score may also depend on the amino acid distribution of the query sequence and on whether it contains regions with strongly biased amino acid composition, as these regions can cause many k-mer matches with unrelated sequences containing similarly biased regions.

Instead of simply estimating the background score from the product of the lengths of query and target sequences, we measure the expected k-mer score of the query sequence *per column of a target sequence*. For that purpose, we can assume that the overwhelming majority of the database sequences are not homologous to a given query sequence.

For each query, perform a calibration search through a database of 100 000 randomly sampled target sequences, whose sum of lengths

$$\text{sum}_L \;=\; \sum_{t=1}^{N}(L_t - k + 1)$$

we recored. We then sum up all prefiltering scores for query sequence q with the database,

$$\mathrm{sum}_S = \sum_{t=1}^{N} S_{qt} ,$$

where N is the database size, L_t is the length of the database sequence t and S_{qt} is the prefiltering score of the query sequence q with a database sequence t. Then, the expected chance prefiltering score between q and a target database sequence t is

$$S_0 = (L_t - k + 1)\,\frac{\mathrm{sum}_\mathrm{S}}{\mathrm{sum}_\mathrm{L}}$$

We also correct for the lower score relative dispersion at high lengths by dividing S_0 by the estimate of its standard deviation. We assume that the number of k-mer matches is Poisson-distributed, and the standard deviation of the score should therefore be proportional to the square root of the number of expected k-mer matches, which is

$$n_\mathrm{match} \approx (L_t - k + 1)\,\frac{\mathrm{sum}_\mathrm{S}}{\mathrm{sum}_\mathrm{L}} / S_\mathrm{match} ,$$

where S_match is the expected score per chance k-mer match. Under the assumption that the number of k-mer matches is Poisson-distributed, the standard deviation of the chance score of the query sequence with a database sequence t is

$$\begin{aligned}\sigma_S &= \sqrt{n_\mathrm{match}}\,S_\mathrm{match}\\ &= \sqrt{(L_t - k + 1)\,\frac{\mathrm{sum}_\mathrm{S}}{\mathrm{sum}_\mathrm{L}}\,S_\mathrm{match}}\end{aligned}$$

Therefore, the offset- and scale-corrected score Z_{qt} for a query sequence q and a database sequence t is

$$Z_{qt} = \frac{S_{qt} - S_0}{\sigma_S} .$$

We are interested in all sequence pairs, where $Z_{qt} > Z_{thr}$, i. e. the prefiltering score should fulfill the condition

$$\begin{aligned}S_{qt} &\geq Z_{thr}\,\sigma_S + S_0\\ &\geq Z_{thr}\sqrt{(L_t - k + 1)\,\frac{\mathrm{sum}_\mathrm{S}}{\mathrm{sum}_\mathrm{L}} S_\mathrm{match}} + (L_t - k + 1)\,\frac{\mathrm{sum}_\mathrm{S}}{\mathrm{sum}_\mathrm{L}} .\end{aligned}$$

Calculating the offset- and scale-corrected score threshold for each pair q, t would slow down the retrieving of prefiltering results. Since the threshold value $Z_{thr}\,\sigma_S + S_0$ depends only on the length of the database sequence t for a fixed q, the database sequences are ordered by length, the threshold is calculated once and recalculated only if the length of the next sequence falls below 95% of the reference sequence length.

The statistical analysis of the scores gets unreliable for small databases of fewer than 100 000 sequences, since in this case there will not be enough sequences to calculate the expected score and the standard deviation reliably. We use pseudo-counts to make the estimate of S_0 and σ_S robust. We define the size of the pseudo-database as 100 000 with an average sequence length 350 (average sequence length in UniProtKB) and an average sequence composition. We estimate k-mer match probability and set the score of a chance k-mer match S_match to be slightly above the k-mer similarity threshold. k-mer match probability estimation is explained in detail in section Automatic sensitivity setting. Then, we calculate n_{match}, sum_S and sum_L by adding the pseudo counts to the empirical counts.

3.2 Amino acid local composition bias correction

Some sequences have regions of low complexity with an amino acid composition that differs considerably from the background amino acid distribution assumed in the amino acid substitution matrix. Low complexity regions of a sequence can lead to high prefiltering scores sequences containing similarly biased low complexity regions. To alleviate this effect, we correct for the local compositional bias in the sequences by assigning lower scores to the matches of *locally* frequent amino acids. We examine $d = 20$ amino acids on both sides of the the amino acid x_i at position i in the sequence. Score correction ΔS_i at position i is

$$\Delta S_i(x_i) = -\frac{1}{2d}\sum_{j=i-d, j\neq i}^{i+d} S(x_i, x_j) + \sum_{a=1}^{20} f(a) S(a, x_i)$$

where $S(x_i, x_j)$ is the amino acid substitution score between amino acids x_i and x_j, and $f(a)$ is the background frequency of the amino acid a. Then, the final corrected score S_c for the match of x_i with another amino acid y_j is

$$S_c(x_i, y_j) = S(x_i, y_j) + \Delta S_i(x_i)$$

Therefore, amino acids that are less frequent in the window $\pm d$ around the sequence position i than the background frequency of this amino acid contribute more to the score, and the more frequent less.

4 PARALLELIZATION WITH OPENMP

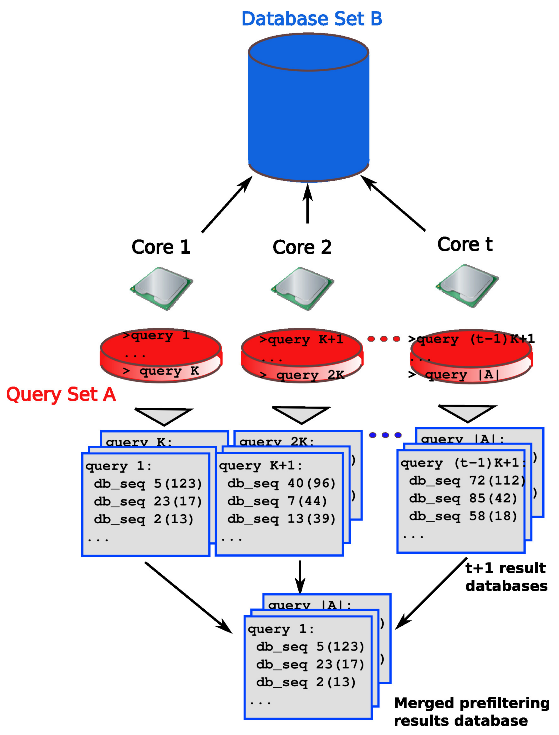


Fig. 1. Parallelization scheme of the prefilter module using OpenMP.

The parallelization approach of the prefiltering and alignment modules is shown in Fig. 1 . Parts of the query sequence set are matched against the database in parallel. Each thread writes to its own output database. In the end, all results are merged into one output database.

We benchmarked the multicore scaling performance of the MMseqs prefilter module with the dataset used in the speed measurements. We tested the run time behavior with five different threads settings 1, 2, 4, 8 and 16 shown in Fig. 2. The prefilter performance scales nearly linearly up to 4 threads. Beyond this the performance scales sublinearly because of the high amount of random memory accesses to the loops 3 and 4 of Figure 1.

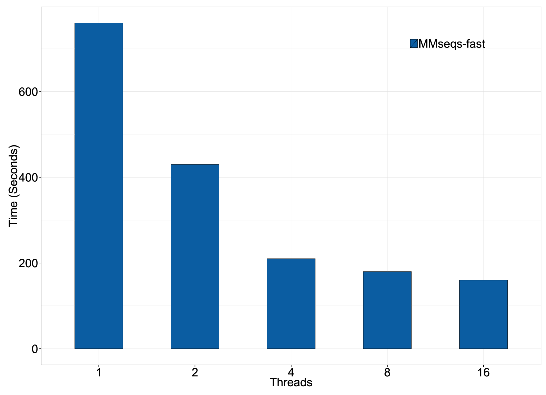


Fig. 2. Multithread scaling of the MMseqs prefilter.. Runtime in seconds for the MMseqs prefilter to run 7616 query sequences against 54 790 250 sequences on different threads setting.

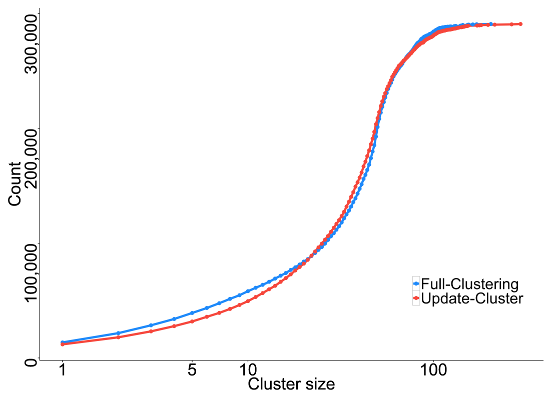


Fig. 4. Cumulative cluster size distribution after ten updating steps versus after a single cascaded clustering, using the parameter -s 7.

5 CLUSTERING RESULTS

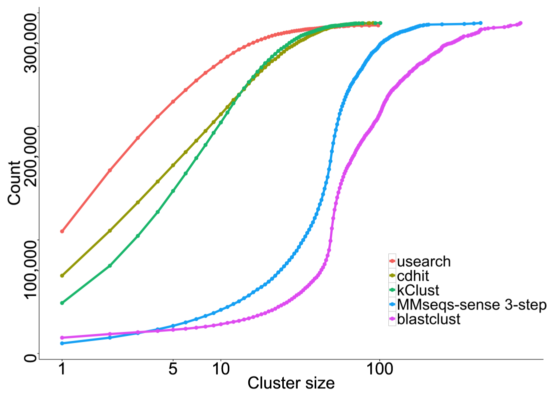


Fig. 3. Cumulative cluster size distributions of blastclust, MMseqs, kClust, CD-HIT, usearch for a clustering threshold of 0.3, corresponding to the leftmost point in Fig. 3F.

Chapter 3

MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets

3.1 Introduction

The dramatic drop in sequencing costs is leading to the creation of huge metagenomics and metatranscriptomics data sets transforming agricultural, microbiological, biotechnological and medical research. One of the main goals in such studies is to annotate the function of the Open Reading Frames (ORFs) predicted from these data sets. However, state-of-the-art sensitive homology-based functional annotation tools are not applicable to the size of such data sets due to their severe speed and cost limits. Therefore fast search tools much less sensitive than BLAST (Altschul et al. 1990) are commonly used for the homology searches. In viral metagenomics data sets for example, 65% to 90% of sequences cannot be annotated at all.

In this manuscript we present the software package MMseqs2 (Steinegger and Söding 2017) that addresses this need. In profile searches, it is hundreds of times faster than BLAST yet 50% more sensitive. It is the first fast search method to support iterative sequence profile searches and to dramatically surpass BLAST in sensitivity.

We showed that MMseqs2 consistently outperforms other state of the art search methods like BLAST (Altschul et al. 1990), PSI-BLAST (Altschul et al. 1997), LAST (Kiełbasa et al. 2011), Rapsearch2 (Zhao, Tang, and Ye 2012), DIAMOND (Buchfink, Xie, and Huson 2015), UBLAST (Edgar 2010) and SWIPE (Rognes 2011) in sensitivity and speed. The MMseqs2 benchmark uses full length protein sequences instead of single domain proteins (Murzin et al. 1995). We previously have shown in the evaluation of MMseqs, that using full length proteins helps to estimate errors occurring through alignments with regions of biased amino acid compositions. MMseqs2 therefore not only enables the metagenomics community to annotate a much higher fraction of sequences. More than that, we believe, it will open up new analysis strategies of the entire protein sequence space, allowing us to link homologous sequences across all metagenomic and genomic data sets, in order to systematically cluster even remotely homologous proteins and better annotate and order the known protein universe.

Martin Steinegger performed the research and programming, Martin Steinegger and Johannes Söding jointly designed the research and wrote the manuscript.

3.2 References

Altschul, S. F. et al. (1990). “Basic local alignment search tool.” In: *J. Mol. Biol.* 215.3, pp. 403–410. DOI: 10.1006/jmbi.1990.9999.

Altschul, S. F. et al. (1997). “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.” In: *Nucleic acids research* 25.17, pp. 3389–402.

Buchfink, B., C. Xie, and D. H. Huson (2015). “Fast and sensitive protein alignment using DIAMOND”. In: *Nature Methods* 12.1, pp. 59–60.

Edgar, R. C. (2010). “Search and clustering orders of magnitude faster than BLAST.” In: *Bioinformatics* 26.19, pp. 2460–2461. DOI: 10.1093/bioinformatics/btq461.

Kiełbasa, S. M. et al. (2011). “Adaptive seeds tame genomic sequence comparison.” In: *Genome research* 21.3, pp. 487–93. DOI: 10.1101/gr.113985.110.

Murzin, A. G. et al. (1995). “SCOP: A structural classification of proteins database for the investigation of sequences and structures”. In: *J. Mol. Biol.* 247.4, pp. 536–540. DOI: http://dx.doi.org/10.1016/S0022-2836(05)80134-2.

Rognes, T. (2011). “Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation.” In: *BMC bioinformatics* 12.1, p. 221. DOI: 10.1186/1471-2105-12-221.

Steinegger, M. and J. Söding (2017). “MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets”. In: *Nature Biotechnology* 35, 1026 EP -. DOI: 10.1038/nbt.3988;https://www.nature.com/articles/nbt.3988#supplementary-information.

Zhao, Y., H. Tang, and Y. Ye (2012). “RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data.” In: *Bioinformatics* 28.1, pp. 125–126. DOI: 10.1093/bioinformatics/btr595.

3.3 Journal article

MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets

To the Editor: The throughput of DNA sequencing has increased much faster than computational speed in the past decade, and sensitive-sequence searching has become the main bottleneck in the analysis of large metagenomic data sets. We therefore developed MMseqs2 (https://github.com/soedinglab/mmseqs2), which improves on current search tools over the full range of speed-sensitivity trade-off, achieving sensitivities better than PSI-BLAST at more than 400 times its speed.

As a result of the drop in sequencing costs by four orders of magnitude since 2007, many large-scale metagenomic projects are being performed, each producing terabytes of sequences with applications in medical, biotechnological, microbiological, and agricultural research[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 function. 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 come at the expense of lower 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 65–90%[2,13], and the widening gap between sequencing and computational costs quickly aggravates this problem.

To address this challenge, we developed MMseqs2, a parallelized, open-source software suite. 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 1**).

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 similar-*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 *k*-mer matches between similar *k*-mers. This similar-*k*-mer matching allows MMseqs2 to use a large word size *k* = 7 without losing 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). For MMseqs2's speed it was crucial to have found a way (explained in **Supplementary Fig. 1** and the

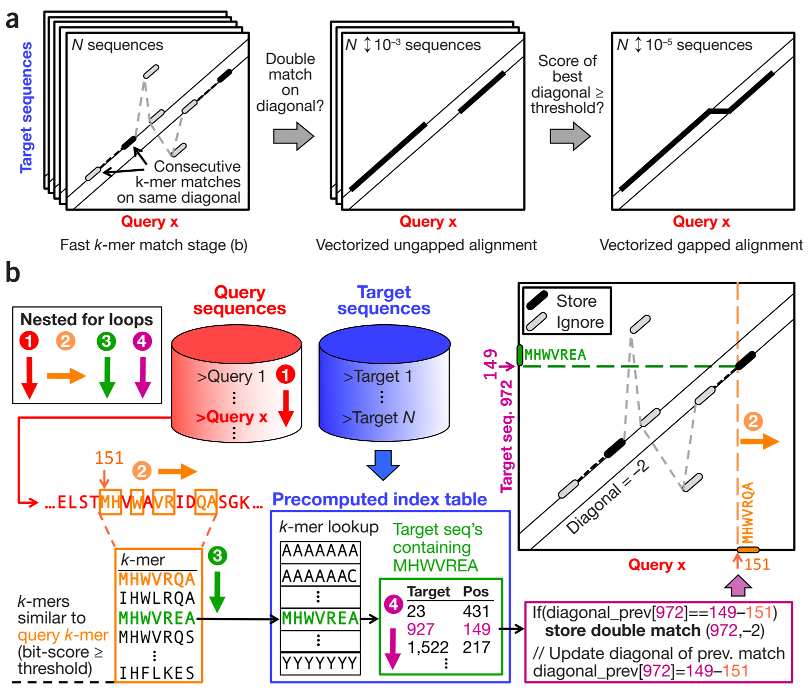


Figure 1 MMseqs2 searching in a nutshell. (**a**) Three increasingly sensitive search stages find similar sequences in the target database. (**b**) The short word (*k*-mer) match stage detects consecutive similar-*k*-mer matches occurring on the same diagonal. The diagonal of a *k*-mer match is the difference 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). For details, see **Supplementary Methods**.

Supplementary Methods) to eliminate the random memory access in the last line of the innermost loop 4 (magenta frame).

The critical insight for the prefilter performance was to combine the double-match criterion with making *k*-mers as long as possible, which required finding similar and not just exact *k*-mers. This effectively bases our decision on up to 2 × 7 = 14 residues instead of just 2 × 3 in BLAST or 12 letters on a size-11 alphabet in DIAMOND.

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. 2**).

MMseqs2 requires 13.4 GB plus 7 bytes 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. 3**).

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], 6,370 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 gave similar results (**Supplementary Figs. 4–7**). 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, that is, the fraction of true-positive matches found with better *E*-value than the first false-positive match. MMseqs2 in sensitive mode (MMseqs2-sens) reaches BLAST's sensitivity while being 36 times faster. 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** and **Supplementary Fig. 4**). 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[19] with a small negative-score offset (**Fig. 2d** and **Supplementary Fig. 7**). All tools except MMseqs2 and LAST have reported far too optimistic *E*-values (**Supplementary Fig. 8**). For example, in 6,370 searches DIAMOND reported 69,211 false-positive matches with *E*-values below 10^{-3} (versus 0.637 expected) in 5% of the searches (versus 0.1% expected), while MMseqs2 produced 54 false-positive matches in only 0.1% of the searches (**Supplementary Table 2**). In automatic functional annotation pipelines, such unreliable *E*-values will lead to an increased fraction of false annotations.

In a comparison of AUC sensitivity and speed (**Fig. 2b**), MMseqs2 with four sensitivity settings (red) showed 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. 4–7, 9** and **10**).

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,c**) using two to four iterations of profile searches through the target database. As expected, MMseqs2 profile searches were much faster and more sensitive than BLAST sequence searches. But MMseqs2 was also considerably more sensitive than PSI-BLAST, despite being 433 times faster at three iterations. This is partly due to its effective suppression of high-scoring false positives and more accurate *E*-values (**Fig. 2d** and **Supplementary Fig. 7**).

The MMseqs2 suite offers workflows for various standard-use cases of sequence and profile searching and clustering of huge sequence data sets, and includes many utility scripts. We illustrate its power with three application examples.

In the first example, we tested MMseqs2 for annotating proteins in the Ocean Microbiome

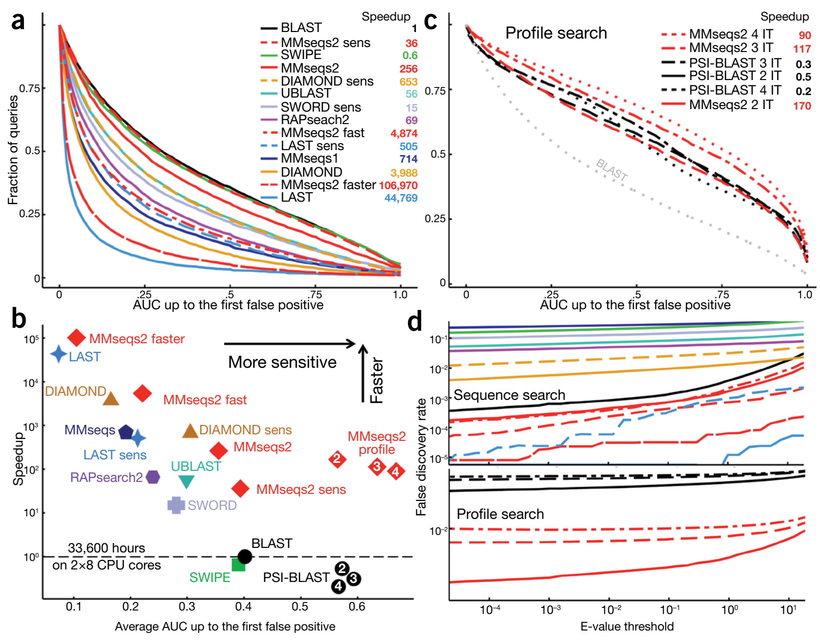


Figure 2 MMseqs2 pushes the boundaries of sensitivity-speed trade-off. (**a**) Cumulative distribution of AUC sensitivity for all 6,370 searches with UniProt sequences through the database of 30.4 M full-length sequences. Higher curves signify higher sensitivity. Legend: speed-up factors relative to 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 7.1 min for 30.3 M 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). The command line parameters of all tools are listed in **Supplementary Table 3**.

3. MMseqs2 *enables sensitive protein sequence searching for the analysis of massive data sets*

Reference Gene Catalog (OM-RGC)[1]. The speed and quality bottleneck is the search through the eggNOGv3 database[20]. 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. 11**). 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% of 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 obtained at least one match for 3.5 M (28.3%) profiles 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 h. In summary, MMseqs2 matched 78% of 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 protein sequences with Pfam[21] domains. We predicted these sequences with an average length of 134 amino acids in ~2,200 metagenome and metatranscriptome data sets[22]. We searched with each sequence through the 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 Figure 12** explains the adaptations to the k-mer prefilter and search workflow. The entire search took 8.3 h, or 0.76 ms per query sequence per core and resulted in 370 M domain annotations with E-values < 0.001. A search of 1,100 randomly sampled sequences from the same set with HMMER3 (ref. 23) through Pfam took 10.6 s per sequence 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 4,000 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 facilitate the analysis of large data sets and even the entire genomic and metagenomic protein sequence space at once. MMseqs2 source code is available in **Supplementary Source Code** and at mmseqs.org and at https://doi.org/10.5281/zenodo.839602. A compressed tar file containing all databases, evaluation tools, tested method binaries (excluding UBLAST), and benchmark scripts is available at http://wwwuser.gwdg.de/~compbiol/mmseqs2/mmseqs2-benchmark.tar.gz.

Editor's note: This article has been peer-reviewed.

Note: Any Supplementary Information and Source Data files are available in the online version of the paper (http://dx.doi.org/10.1038/nbt.3988).

ACKNOWLEDGMENTS
We are grateful to C. Notredame and C. Seok for hosting M.S. at the CRG in Barcelona and at Seoul National University for 12 and 18 months, respectively, and to Burkhard Rost at TU Munich for accepting the formal supervision of his PhD thesis. 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 laboratory 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.

Martin Steinegger & Johannes Söding

Martin Steinegger and Johannes Söding are in the Quantitative and Computational Biology group, Max-Planck Institute for Biophysical Chemistry, Göttingen, Germany. Martin Steinegger is in the Department for Bioinformatics and Computational Biology, Technische Universität München, Garching, Germany.
e-mail: johannes.soeding@mpibpc.mpg.de

Published online 16 October 2017; doi:10.1038/nbt.3988

1. Sunagawa, S. *et al. Science* **348**, 1261359 (2015).
2. Afshinnekoo, E. *et al. Cell Syst.* **1**, 72–87 (2015).
3. Howe, A.C. *et al. Proc. Natl. Acad. Sci. USA* **111**, 4904–4909 (2014).
4. Franzosa, E.A. *et al. Nat. Rev. Microbiol.* **13**, 360–372 (2015).
5. Scholz, M.B., Lo, C.C. & Chain, P.S. *Curr. Opin. Biotechnol.* **23**, 9–15 (2012).
6. Desai, N., Antonopoulos, D., Gilbert, J.A., Glass, E.M. & Meyer, F. *Curr. Opin. Biotechnol.* **23**, 72–76 (2012).
7. Tang, W. *et al. 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., Wan, R., Sato, K., Horton, P. & Frith, M.C. *Genome Res.* **21**, 487–493 (2011).
11. Zhao, Y., Tang, H. & Ye, Y. *Bioinformatics* **28**, 125–126 (2012).
12. Buchfink, B., Xie, C. & Huson, D.H. *Nat. Methods* **12**, 59–60 (2015).
13. Hurwitz, B.L. & Sullivan, M.B. *PLoS One* **8**, e57355 (2013).
14. Hauser, M., Steinegger, M. & Söding, J. *Bioinformatics* **32**, 1323–1330 (2016).
15. Murzin, A.G., Brenner, S.E., Hubbard, T. & Chothia, C. *J. Mol. Biol.* **247**, 536–540 (1995).
16. Karplus, K., Barrett, C. & Hughey, R. *Bioinformatics* **14**, 846–856 (1998).
17. Rognes, T. *BMC Bioinformatics* **12**, 221 (2011).
18. Frith, M.C. *Nucleic Acids Res.* **39**, e23–e23 (2011).
19. Frith, M.C., Park, Y., Sheetlin, S.L. & Spouge, J.L. *Nucleic Acids Res.* **36**, 5863–5871 (2008).
20. Jensen, L.J. *et al. Nucleic Acids Res.* **36**, D250–D254 (2008).
21. Finn, R.D. *et al. Nucleic Acids Res.* **44 D1**, D279–D285 (2016).
22. Steinegger, M. & Söding, J. Preprint at bioRxiv https://dx.doi.org/10.1101/104034 (2017).
23. Eddy, S.R. *PLOS Comput. Biol.* **7**, e1002195 (2011).

MMseqs2: sensitive protein sequence searching for analysis of massive data sets

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

e-mail: johannes.soeding@mpibpc.mpg.de; martin.steinegger@mpibpc.mpg.de

Supplementary 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[8], 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 S2** 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 processes 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[21], 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[13;9;2]. To increase the seeds' sensitivity, some methods allow for one or two mismatched positions[12;9], others employ reduced alphabets[20;23;9;2]. Many use spaced k-mer seeds to reduce spatial clustering of chance matches[9;2].

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.[17].)

The double k-mer match criterion effectively bases our decision to trigger the next search stage on 11 to 14 amino acids (depending on the number of common residues between seeds) instead of just 2×3 in BLAST or 12 letters on a size-11 reduced alphabet in DIAMOND. (Even though DIAMOND uses 4 seed patterns with 12 letters, it takes independent decisions for each of the patterns.) The information content in each of the reduced-alphabet letters is 3.0 in contrast to 4.2 bits for the full 20-letter amino acid alphabet.

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[7].

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 $B = \texttt{ceil}(\texttt{log2}(N/\texttt{L2\_size}))$.

Index table generation. For the k-mer match stage we preprocess the target database into an index table. 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)[4].

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 transform the k-mer count array into an offset table. Each thread picks a new sequence from the database, parallelized by OpenMP using `#pragma omp parallel for`, and processes all k-mers in the sequence. Each k-mer is now written into the entry array. We can prevent two threads writing into the same position pointed to by the offset pointer by fetching and incrementing it at the same time using the atomic `__sync_fetch_and_add` instruction. Building the index table file for 3×10^7 sequences takes 7.1 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 $S_{\max} = \texttt{max}(S_{\max}, S)$. 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.[22], 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[8], 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 $S(i, \mathrm{aa})$ for amino acid aa at position i is corrected to $S_{\mathrm{corr}}(i, \mathrm{aa}) = S(i, \mathrm{aa}) - \frac{1}{40} \sum_{j=i-20, j \neq i}^{i+20} S(j, \mathrm{aa}) + \frac{1}{L_{\mathrm{query}}} \sum_{j=1}^{L_{\mathrm{query}}} S(j, \mathrm{aa})$.

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 search mode. The first iteration of the profile-to-sequence search is a straightforward MMseqs2 sequence-to-sequence search. We then realign all matched sequences with E-values below the specified inclusion threshold (option `--e-profile <value>`, default value 0.1). At this stage, we add a score offset of -0.1 bits per matched residue pair to the scores of the substitution matrix to avoid *homologous overextensions* of the alignments, a serious problem causing many false positives in iterative profile searches[5;6]. In all further iterations, we remove from the prefilter results sequences that were previously included in the profile 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[15]. 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[18]. All matches included in the profile or achieving an E-value in the last iteration below the value given by `-e <value>` are displayed.

Sequence-to-profile 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 $S_{sp} > 15\mathrm{bit}$. 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, $\{s : S_{ps} > 15\mathrm{bit}\}$, 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[13]. 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[10;9;2;23]. 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[16] 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[14] 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[19]) 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[19]. 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[16] 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[11]. 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 S3**.

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`.

References

[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. A new repeat-masking method enables specific detection of homologous sequences. Nucleic Acids Res., 39(4):e23–e23, 03 2011.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

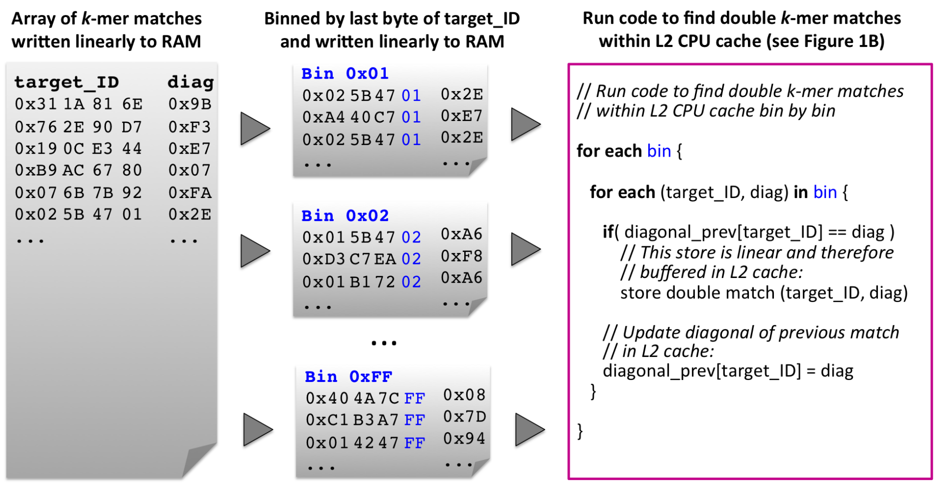
[20] J. Tan, D. Kuchibhatla, F. L. Sirota, W. A. Sherman, T. Gattermayer, C. Y. Kwoh, F. Eisenhaber, G. Schneider, and S. M. Stroh. Tachyon search speeds up retrieval of similar sequences by several orders of magnitude. Bioinformatics, 28(12):1645–1646, June 2012.

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

[22] 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.

[23] 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.

Supplementary Figures and Tables


Array of *k*-mer matches written linearly to RAM
Binned by last byte of target_ID and written linearly to RAM
Run code to find double *k*-mer matches within L2 CPU cache (see Figure 1B)

```
target_ID     diag
0x31 1A 81 6E 0x9B
0x76 2E 90 D7 0xF3
0x19 0C E3 44 0xE7
0xB9 AC 67 80 0x07
0x07 6B 7B 92 0xFA
0x02 5B 47 01 0x2E
...           ...
```

```
Bin 0x01
0x02 5B 47 01 0x2E
0xA4 40 C7 01 0xE7
0x02 5B 47 01 0x2E
...           ...

Bin 0x02
0x01 5B 47 02 0xA6
0xD3 C7 EA 02 0xF8
0x01 B1 72 02 0xA6
...           ...

...

Bin 0xFF
0x40 4A 7C FF 0x08
0xC1 B3 A7 FF 0x7D
0x01 42 47 FF 0x94
...           ...
```

```
// 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 during k-mer match stage in MMseqs2 Numbers in this figure are represented in hexadecimal notation (e.g. 0xFF is equal to 255 in 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 purposes, we set $B = 8$. 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 contains `ceil`($N/2^B$) entries, where N is the number of sequences in the target database.

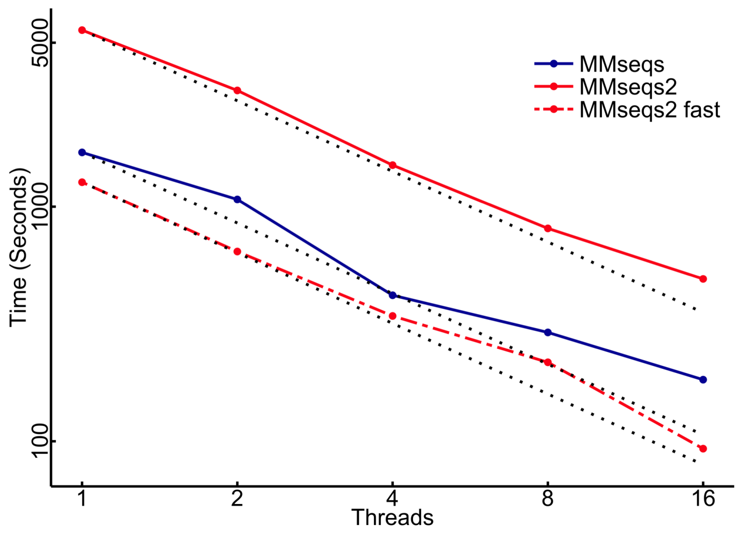


Figure S 2: **Multi-core scaling of MMseqs2** Runtimes of MMseqs and MMseqs2 searches in fast and default sensitivity using 1, 2, 4, 8 and 16 threads on a 2 × 8 core server with 128 GB main memory. Theoretically optimal scaling is indicated as a dashed black line for each method. We searched with 6370 full length protein queries against 30 Mio. UniProt sequences. On 16 cores, MMseqs achieves 58% and MMseqs2 85% of their theoretical maximum performance interpolated from the single core measurement. The improvement in scaling behaviour from MMseqs to MMseqs2 is owed to minimizing random main memory accesses, as explained in **Fig. S1**.

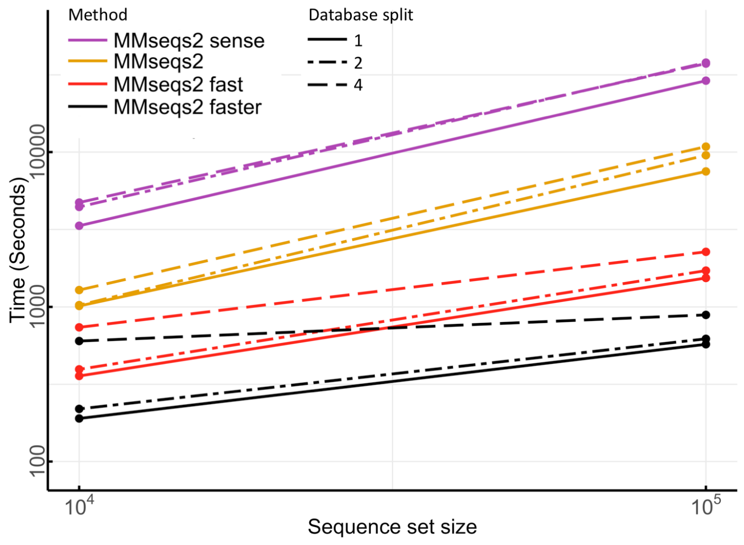


Figure S 3: **Runtime of MMseqs2 against the UniProt at different sensitivity and database split settings.** We measured the search time with query sets of 10 000 and 100 000 sequences through the UniProt database (Release 2017_03 with 80 204 488 sequences) using four sensitivity settings (faster, fast, default, and sensitive) and splitting the database into 1, 2, and 4 chunks. Runtimes for Refseq/Genbank (Release March 3, 2017 with 81 027 309 sequences) are very similar. The memory consumption of the index table for the split levels of 1, 2, and 4 was 190GB, 101GB, and 57GB respectively. All searches ran on a 2×14-core server with 768GB main memory.

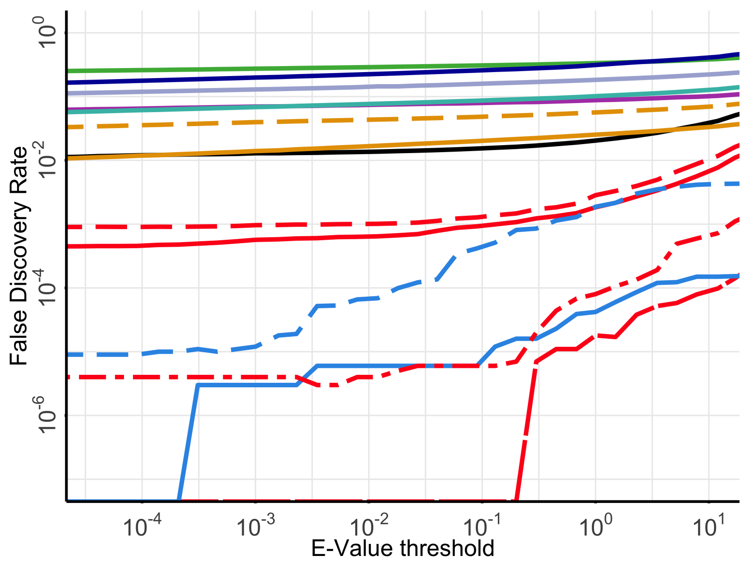


Figure S 4: False discovery rate versus E-value threshold in version 2 of the sequence search sensitivity benchmark using unshuffled query sequences. Colors are the same as in **Fig. 2a**

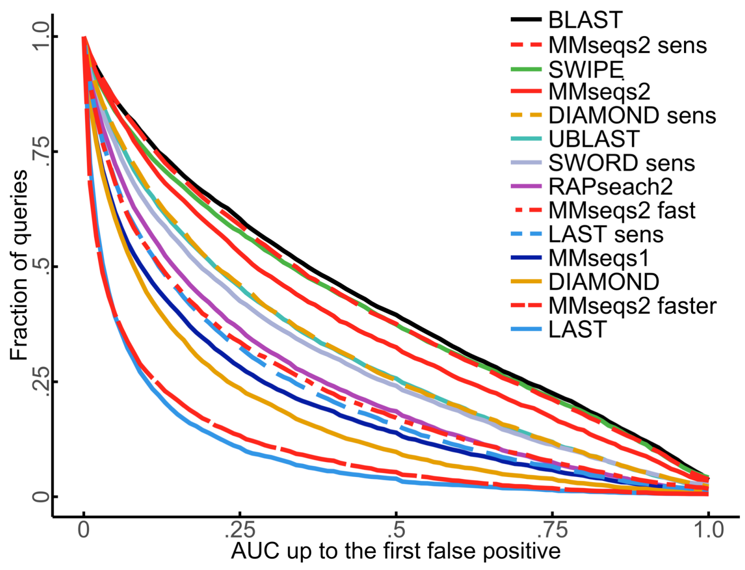


Figure S 5: **Sequence searching sensitivity assessment with unshuffled query sequences.** Cumulative distribution of area under the curve (AUC) sensitivity for all 6324 queries in version 2 of the sequence search sensitivity benchmark using unshuffled query sequences. Higher curves signify higher sensitivity.

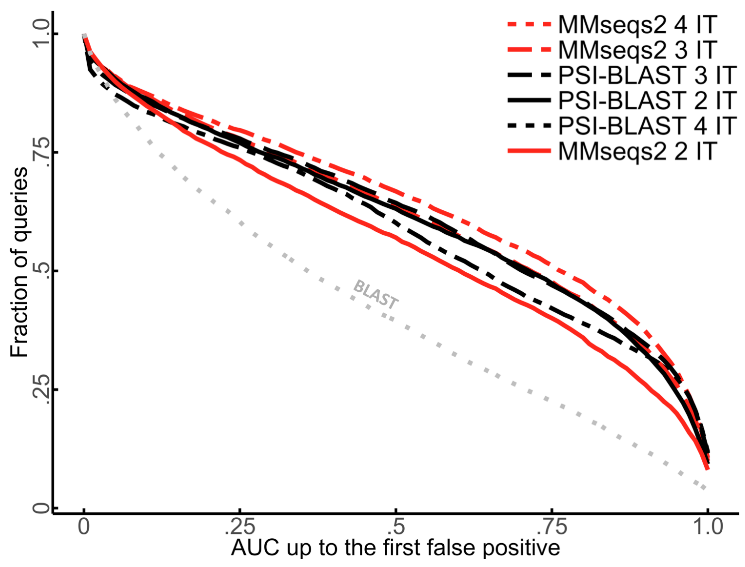


Figure S 6: **Sequence profile searching sensitivity assessment with unshuffled query sequence profiles.** Cumulative distribution of area under the curve (AUC) sensitivity for all 6324 unshuffled query sequences in version 2 of the sequence search sensitivity benchmark using unshuffled query sequences. Higher curves signify higher sensitivity. Higher curves signify higher sensitivity. 2 IT: 2 search iterations etc.

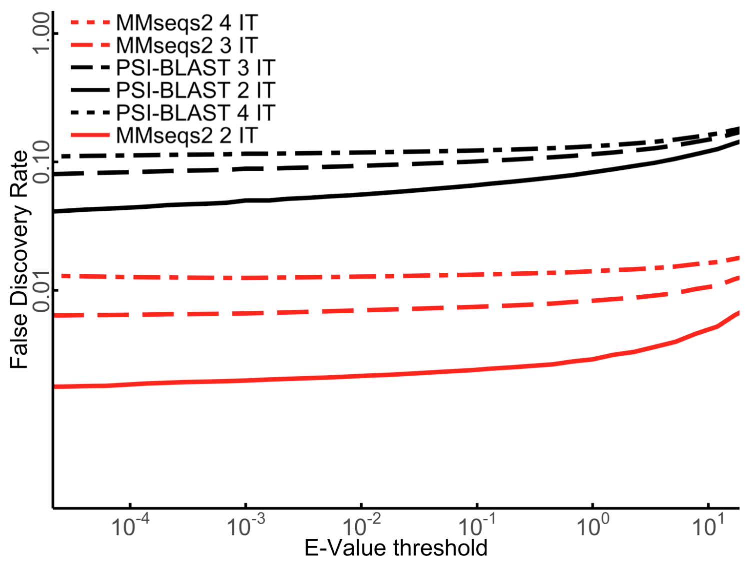


Figure S 7: False discovery rate versus E-value threshold in version 2 of the sequence profile search sensitivity benchmark using unshuffled query sequences.

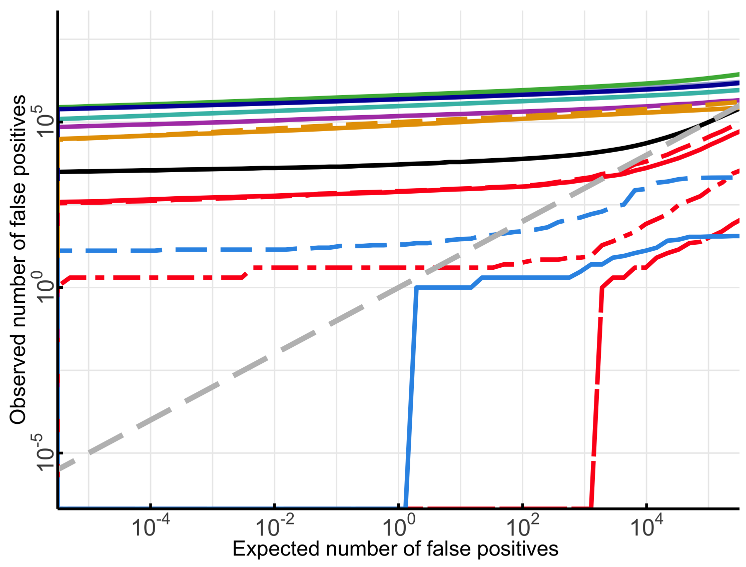


Figure S 8: **Accuracy of reported *E*-values.** The expected number of false positives is the *E*-value threshold times the number of searches, $E \times 6324$. The observed number of false positives is the total number of false positives below the E-value threshold in all 6324 searches. If *E*-values were accurate, observed and expected numbers of false positives would coincide (diagonal grey line). LAST and MMseqs2 report the most accurate *E*-values. The false positives shown were obtained with version 2 of the sequence search sensitivity benchmark. Colors are the same as in **Fig. 2a**.

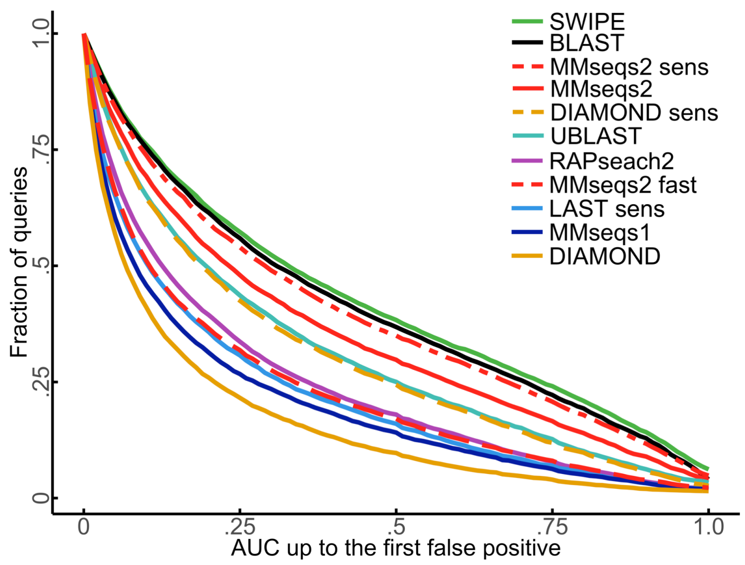


Figure S 9: **Sequence searching sensitivity assessment with single-domain SCOP sequences.** Cumulative distribution of area under the curve (AUC) sensitivity for all 7616 single domain SCOP sequences. Higher curves signify higher sensitivity. AUC up to the first false positive is the fraction of true positive matches found with better E-value than the first false positive match.

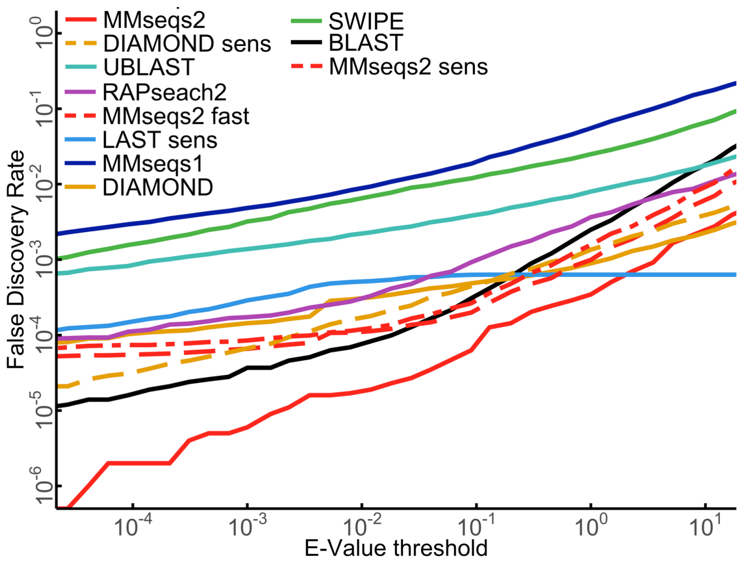


Figure S 10: False discovery rate versus E-value threshold for the single-domain SCOP sequence search sensitivity benchmark.

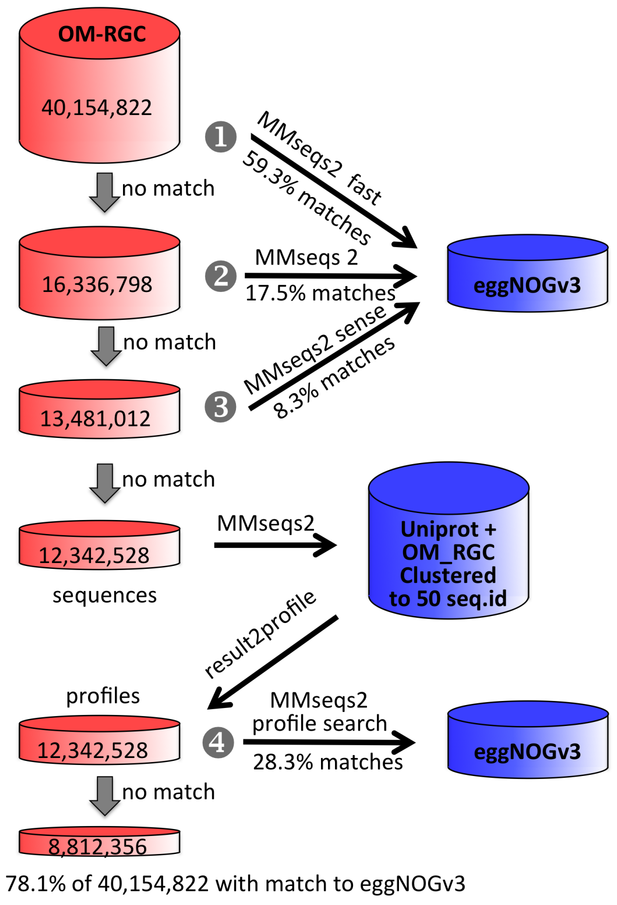

OM-RGC
40,154,822
no match
16,336,798
no match
13,481,012
no match
12,342,528
sequences
1 MMseqs2 fast 59.3% matches
2 MMseqs 2 17.5% matches
3 MMseqs2 sense 8.3% matches
eggNOGv3
MMseqs2
Uniprot + OM_RGC Clustered to 50 seq.id
result2profile
profiles
12,342,528
4 MMseqs2 profile search 28.3% matches
eggNOGv3
no match
8,812,356
78.1% of 40,154,822 with match to eggNOGv3

Figure S 11: Workflow for fast and deep annotations of the Ocean Microbiome Reference Gene Catalog (OM-RGC) using MMseqs2.

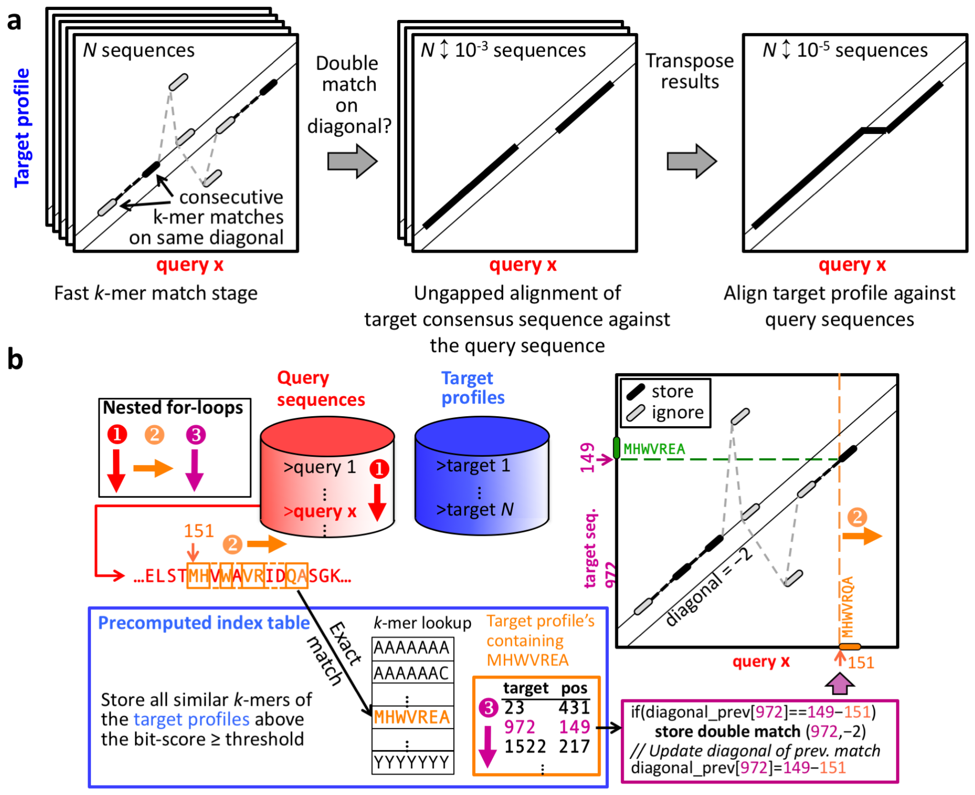

a
Target profile
N sequences
consecutive k-mer matches on same diagonal
query x
Fast k-mer match stage
Double match on diagonal?
N ↕ 10^-3 sequences
Ungapped alignment of target consensus sequence against the query sequence
Transpose results
N ↕ 10^-5 sequences
Align target profile against query sequences
b
Nested for-loops
Query sequences
>query 1
>query x
Target profiles
>target 1
>target N
151
…ELSTMHVWAVRIDQASGK…
Exact match
Precomputed index table
Store all similar k-mers of the target profiles above the bit-score ≥ threshold
k-mer lookup
AAAAAAA
AAAAAAC
MHWVREA
YYYYYYY
Target profile's containing MHWVREA
target pos
23 431
972 149
1522 217
store
ignore
149
MHWVREA
target seq. 972
diagonal = −2
MHWVRQA
query x
151
if(diagonal_prev[972]==149−151)
store double match (972,−2)
// Update diagonal of prev. match
diagonal_prev[972]=149−151

Figure S 12: **Algorithmic changes to perform fast sequence profile searches using MMseqs2.** We precompute all similar k-mers above a similarity threshold for each target profile and store them into the index table. For each query sequence we run over its overlapping, spaced k-mers (loop 2) and look up in the index table (blue frame) only the exact same k-mer. At the ungapped alignment stage we use the target profile consensus sequence. We transpose the results, i.e., we exchange the role of query and target in the results and then, as the last step, align the profiles against all query sequences and transpose back.

Feature	**MMseqs**	**MMseqs2**	**Remark**
Iterative profile searches	no	yes	Iterative profile searches increase sensitivity far beyond the sensitivity of BLAST
Sequence-to-profile searches	no	yes	Protein sequences can be annotated very fast by searching through databases of profiles, e.g. for Pfam, eggNOG, or PDB
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's consecutive double-diagonal k-mer match criterion suppresses most spurious matches and also works well 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 better utilizing low-level CPU caches (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 set-cover, single-linkage with depth cut-off	MMseqs2 has an option to reassign cluster members to the best representative
Utility scripts	3	37 (see MMseqs2 userguide on GitHub)	MMseqs2 has added utility tools for format conversion, multiple sequence alignment, sequence profile calculation, ORF extraction, 6-frame translation, 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
Split target database among servers	no	yes	Allows MMseqs2 to search or cluster arbitrarily large databases with limited memory
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 large proportion of the MMseqs code has been rewritten from scratch and considerably modified for better performance.

Table S 1: **Comparison between MMseqs and MMseqs2.**

Method	Fractions of queries containing a false positive with E-value $< 10^{-3}$	Number of reported false positive matches with E-value$< 10^{-3}$ (Expected number $= 6370 \times 10^{-3} = 0.637$)
MMseqs2 sens	0.001	39
MMseqs2	0.001	54
MMseqs2 fast	0.002	126
MMseqs2 faster	0.001	9
UBLAST	0.107	380807
DIAMOND	0.050	69211
DIAMOND sens	0.067	124906
LAST sense	0.001	7
LAST	0	0
BLAST	0.022	1313
RAPsearch2	0.044	141665
SWORD sens	0.131	676566
MMseqs1	0.159	537403
SWIPE	0.124	788356
MMseqs2 2 IT	0.013	6202
MMseqs2 3 IT	0.024	44270
MMseqs2 4 IT	0.031	94747
PSI-BLAST 2 IT	0.253	
PSI-BLAST 3 IT	0.294	1.01498e+06
PSI-BLAST 4 IT	0.299	1.13397e+06

Table S 2: Analysis of high-scoring false positive matches in 6370 searches with UniProt sequences through a database of 30 million UniProt-derived sequences. For most tools, a seemingly significant E-value (e.g. smaller than 0.001) is not a strong indication of a homologous relationship. In automatic functional annotation pipelines, such unreliable E-values will lead to an increased fraction of false annotations. But note that even unreliable E-values can work very well for ranking matches *within a single search*, and the quality of ranking matches within a single search is what is measured by the AUC1 sensitivity.

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 (140 | 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 -s 5.7 -e 10000.0 --max-seqs 4000 --use-index

Table S 3: Program versions and command line parameters of tools used in the benchmark.

Chapter 4

Clustering huge protein sequence sets in linear time

4.1 Introduction

Publicly available metagenomics data sets at repositories such as IMG/M (Markowitz et al. 2014), MG-RAST (Wilke et al. 2016) or the Sequence Read Archive (Leinonen et al. 2011) contain tens of billions of gene sequences. These data sets have been analysed separately within each study. Ideally, these data sets should be examined together cross data set. This would enable us for example, to handle rare sequences and compute more accurate functional clusters. But the huge amount of data makes it challenging to integrate data sets across many metagenomics studies.

In this manuscript we describe an algorithm and software suite that can cluster huge protein sequences sets orders of magnitude faster than has been possible with current tools. Linclust is the first general, linear-time sequence clustering algorithm, with $O(N)$ instead of $O(NK)$ time complexity, where N is the input set size and K is the number of clusters. With this tool, we could cluster 1.6 billion protein sequences, extracted from about 2200 metagenomes and metatranscriptomes, within half a day on a single 24 CPU core server. Metaclust, the resulting clustered database is the largest publicly available protein set. The Linclust algorithm and the Metaclust data sets will find many applications in protein structure prediction, homology searching and sequence annotation.

In addition, we combined Linclust and MMseqs2 search-based clustering into a single workflow. The MMseqs2/Linclust workflow first clusters with Linclust and then performs 3 cascaded clustering steps using MMseqs2 sequence searching. This combined workflow scales linearly with the number of input sequences. 61 million sequences are clustered about 30 times faster than with the previous, pure MMseqs2 search-based clustering and achieves similar sensitivity.

We benchmarked Linclust against the state of the art clustering tools CD-HIT (Li and Godzik 2006), UCLUST (Edgar 2010) and MMseqs (Hauser, Steinegger, and Söding 2016) and also against sequence search tools RAPsearch2 (Zhao, Tang, and Ye 2012), DIAMOND (Buchfink, Xie, and Huson 2015) and MASH (Ondov et al. 2016). At 50% sequence identity, Linclust clusters 123 million sequences 2000 faster than UCLUST, 610 faster than MMseqs, 4000 faster than CD-HIT, 1200 faster than DIAMOND, 59 000 faster than MASH, 20 000 than RAPsearch2.

Such fast clustering abilities is expected to be useful in metagenomics, where usually less than half of all predicted proteins have identifiable similarity with a functionally annotated protein (Sunagawa et al. 2015). We believe Linclust will prove to be an enabling technology to exploit the tremendous value of the many publicly available metagenomic and metatranscriptomic data sets.

Martin Steinegger performed the research and programming, Martin Steinegger and Johannes Söding jointly designed the research and wrote the manuscript.

4.2 References

Buchfink, B., C. Xie, and D. H. Huson (2015). “Fast and sensitive protein alignment using DIAMOND”. In: *Nature Methods* 12.1, pp. 59–60.

Edgar, R. C. (2010). “Search and clustering orders of magnitude faster than BLAST.” In: *Bioinformatics* 26.19, pp. 2460–2461. DOI: `10.1093/bioinformatics/btq461`.

Hauser, M., M. Steinegger, and J. Söding (2016). “MMseqs software suite for fast and deep clustering and searching of large protein sequence sets”. In: *Bioinformatics* 32.9, pp. 1323–1330.

Leinonen, R. et al. (2011). “The sequence read archive.” In: *Nucleic acids research* 39.Database issue, pp. D19–21. DOI: `10.1093/nar/gkq1019`.

Li, Weizhong and Adam Godzik (2006). “Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences.” In: *Bioinformatics* 22.13, pp. 1658–1659. DOI: `10.1093/bioinformatics/btl158`.

Markowitz, V. M. et al. (2014). “IMG/M 4 version of the integrated metagenome comparative analysis system”. In: *Nucleic Acids Res.* 42.D1, pp. D568–D573.

Ondov, B. D. et al. (2016). “Mash: fast genome and metagenome distance estimation using MinHash”. In: *Genome Biology* 17.1, p. 132. DOI: `10.1186/s13059-016-0997-x`.

Sunagawa, S. et al. (2015). “Structure and function of the global ocean microbiome”. In: *Science* 348.6237, pp. 1261359-1–9.

Wilke, A. et al. (2016). “The MG-RAST metagenomics database and portal in 2015”. In: *Nucleic Acids Res.* 44.D1, pp. D590–D594.

Zhao, Y., H. Tang, and Y. Ye (2012). “RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data.” In: *Bioinformatics* 28.1, pp. 125–126. DOI: `10.1093/bioinformatics/btr595`.

4.3 Journal article

Clustering huge protein sequence sets in linear time

Martin Steinegger[1, 2, 3] 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*
[3] *Department of Chemistry, Seoul National University, Seoul, Korea*

Metagenomic datasets contain billions of protein sequences that could greatly enhance large-scale functional annotation and structure prediction. Utilizing this enormous resource would require reducing its redundancy by similarity clustering. However, clustering hundreds of million of sequences is impractical using current algorithms because their runtimes scale as the input set size N times the number of clusters K, which is typically of similar order as N, resulting in runtimes that increase almost quadratically with N. We developed Linclust, the first clustering algorithm whose runtime scales as N, independent of K. It can also cluster datasets several times larger than the available main memory. We cluster 1.6 billion metagenomic sequence fragments in 10 hours on a single server to 50% sequence identity, > 1000 times faster than has been possible before. Linclust will help to unlock the great wealth contained in metagenomic and genomic sequence databases. (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 *in-vitro* [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 tools for clustering and redundancy filtering of protein sequence sets (see [9] for a review). Their goal is to find a representative set of sequences such that each of the input set sequences is represented well enough by one of the K representatives, where "well enough" is quantified by some similarity criteria.

Like most other fast sequence clustering tools, they use a fast prefilter to reduce the number of slow pairwise sequence alignments. An alignment is only computed if two sequences share a minimum number of identical k-mers (substrings of length k). If we denote the average probability by p_{match} that this happens by chance between two non-homologous input sequences, then the prefilter would speed up the sequence comparison by a factor of up to $1/p_{\mathrm{match}}$ at the expense of some loss in sensitivity. This is usually unproblematic: if sequence matches are missed (false negatives) we create too many clusters, but we do not lose information. In contrast, false positives are costly as they can cause the loss of unique sequences from the representative set.

CD-HIT and UCLUST employ the following *greedy incremental clustering* approach: each of the N input sequences is compared with the representative sequences of already established clusters. When the sequence is similar enough to the representative sequence of one of the clusters, that is, the similarity criteria such as sequence identity are satisfied, the sequence is added to that cluster. Otherwise, the sequence becomes the representative of a new cluster. Due to the comparison of all sequences with the cluster representatives, the runtimes of CD-HIT and UCLUST scale as $\mathrm{O}(NK)$, where K is the final number of clusters. In protein sequence clustering K is typically of similar size to N (see for example Fig. 2c) and therefore the total runtime scales almost quadratically with N. The fast sequence prefilters speed up each pairwise comparison by a large factor $1/p_{\mathrm{match}}$ but cannot improve the time complexity of $\mathrm{O}(NK)$. This almost quadratic scaling results in impractical runtimes for a billion or more sequences.

Here we present the sequence clustering algorithm Linclust, whose runtime scales as $O(N)$, independent of the number of clusters found. We demonstrate that it produces clusterings of comparable quality as other tools that are orders of magnitude slower and that it can cluster over a billion sequences within hours on a single server.

RESULTS

Overview of Linclust. The Linclust algorithm is explained in Figure 1 (for details see Methods and Figure 5). As in previous methods, we reduce the number of pair-

[*] soeding@mpibpc.mpg.de

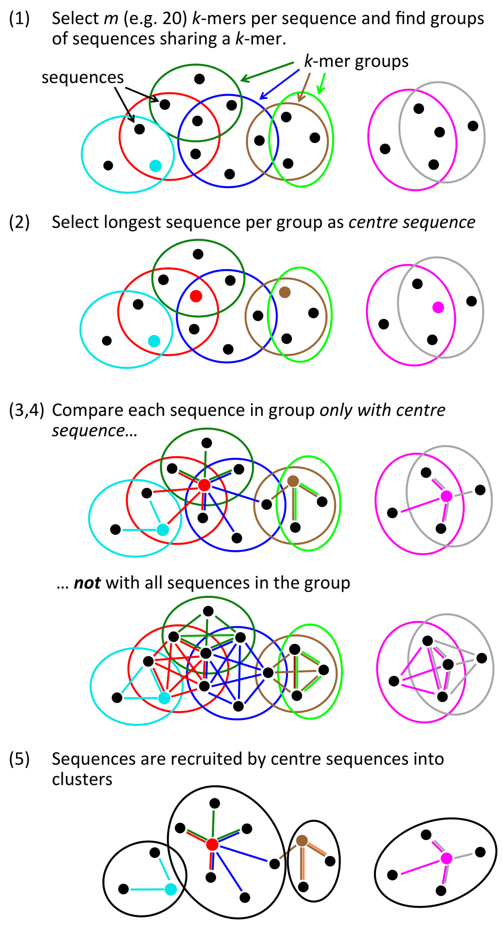

(1) Select m (e.g. 20) k-mers per sequence and find groups of sequences sharing a k-mer.
sequences
k-mer groups
(2) Select longest sequence per group as centre sequence
(3,4) Compare each sequence in group only with centre sequence…
… not with all sequences in the group
(5) Sequences are recruited by centre sequences into clusters

FIG. 1. **Overview of linear-time clustering algorithm.** (1) For each sequence Linclust selects m k-mers (with the lowest hash function values). It sorts the k-mers alphabetically in quasi-linear time to find the groups of sequences sharing a k-mer (colored sets) and (2) it selects the longest sequence per k-mer group as centre. (3,4) It compares each sequence (in three consecutively slower and more sensitive steps) *only with the centre sequences* it shares a k-mer with, not with all sequences it shares a k-mer with. It therefore needs to compute at most m comparisons per sequence and mN in total. Pairs that pass the clustering criteria are linked by an edge. (5) The sequences are clustered in time $O(mN)$ using a greedy incremental algorithm that finds clusters whose members all have an edge with a representative sequence. For a more details see Figure 5.

wise comparisons by requiring the sequences to share at least one identical k-mer substring. A critical insight to achieve linear time complexity is that we need not align every sequence with every other sequence sharing a k-mer (see steps 3,4). We reach similar sensitivities by selecting only a very small subset of sequences as *centre sequences* (colored dots) and only aligning sequences to the centre sequences with which they share a k-mer. Linclust thus requires less than mN sequence comparisons with a small constant m (default value 20), instead of the $\sim NKp_{\text{match}}$ comparisons needed by UCLUST, CD-HIT and other tools.

In most clustering tools, the main memory size severely limits the size of the datasets that can be clustered. UCLUST, for example, needs 10 bytes *per residue* of the representative sequences. Linclust needs $m \times 16$ bytes *per sequence*, but before running it automatically checks available main memory and if necessary splits the table of mN lines into chunks such that each chunk fits into memory (Supplemental Fig. S1 and Methods). It then processes the chunks sequentially. In this way, Linclust can cluster sequence sets that would occupy many times its main memory size at almost no loss in speed.

Linclust and Linclust/MMseqs2 workflows. We integrated Linclust into our MMseqs2 (Many-versus-Many sequence searching) software package [10], and we test two versions of Linclust in our benchmark comparison: the bare Linclust algorithm described in Figure 1 (simply named "Linclust"), and a combined four-step cascaded clustering workflow ("Linclust/MMseqs2"). In this workflow, a Linclust clustering step is followed by one (above 60% sequence identity) or three ($\leq 60\%$) clustering steps, each of which clusters the representative sequences from the previous step by performing an increasingly sensitive all-against-all MMseqs2 sequence search followed by the greedy incremental clustering algorithm. We also include in our benchmark our original MMseqs clustering tool [11].

Runtime and clustering sensitivity benchmark. We measure 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, and UniProt plus all reversed sequences (123 million sequences). Each tool clustered these sets using a minimum pairwise sequence identity of 90%, 70% and 50%. Sequence identity was defined similarly for all three tools. The three tools use somewhat different strategies to try to ensure that only proteins with the same domain architecture are clustered together (see Methods: Clustering criteria).

At 50% identity, Linclust clusters the 123 million sequences 10 times faster than Linclust/MMseqs2 and, by extrapolation, 2300 times faster than UCLUST, 720 times faster than MMseqs, 4600 times faster than CD-HIT, 1600 times faster than DIAMOND [12], 69 000 times faster than MASH [13] , and 26 000 times faster than RAPsearch2 [14] (Fig. 2a,b). At 90% identity, Linclust still clusters these sequences 570 times faster than MMseqs, 100 times faster than UCLUST, 62 times faster than CD-HIT, and 4.5 times faster than Linclust/MMseqs2.

At 90 % sequence identity threshold, we determined how the runtimes scale with the input set size N by fitting a power law ($T \sim aN^b$) to the measured runtimes. Runtimes scale very roughly quadratically for UCLUST ($N^{1.62}$) and

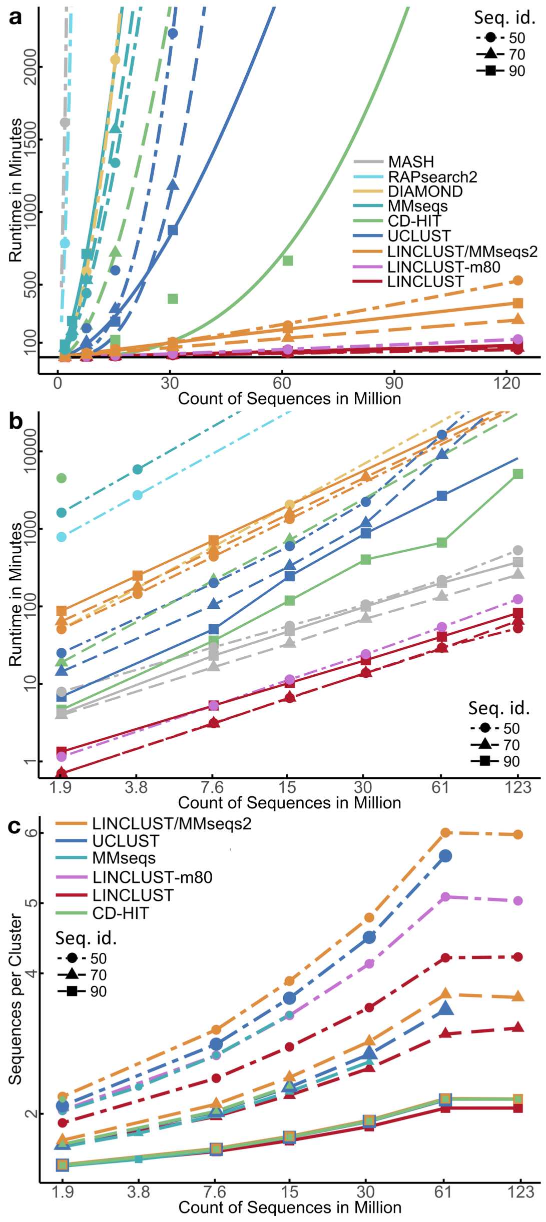


FIG. 2. **Linclust and Linclust/MMseqs2 manifest unique linear scaling of runtime with sequence set size.** (**a**) Runtime versus input set size on linear scales. The plotting symbols indicate the sequence identity threshold for clustering of 90%, 70% and 50%. The curves are fits with a power law, bN^a. For comparison, we include runtimes of all-against-all searches using sequence search tools DIAMOND, RAPsearch2, and MASH. Runtimes were measured on a server with two Intel Xeon E5-2640v3 8-core CPUs and 128 GB RAM. (**b**) Same as (a) but on log-log scales. (**c**) Average number of sequences per cluster at 90%, 70% and 50% sequence identity. Larger average cluster sizes imply higher sensitivities to detect similar sequences.

CD-HIT ($N^{2.75}$) whereas they grow only linearly for Linclust/MMseqs2 ($N^{0.94}$) and Linclust ($N^{1.01}$). The speed-ups due to Linclust's Hamming distance stage and the ungapped alignment filter are analyzed in Supplemental Figure S2.

To assess the clustering sensitivity, we compare the average size of clusters: a deeper clustering with more sequences per cluster implies a higher sensitivity to detect similar sequences. All three tools produce similar numbers of clusters at 90% and 70% sequence identity (Fig. 2c). 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 13% 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, Linclust takes only 1.5 to 2 times longer but attains a sensitivity similar to UCLUST (pink in Fig. 2a-c, Supplemental Fig. S4).

To estimate the fraction of missed sequence pairs that could have been clustered together, we examined the distribution of sequence identities between representative cluster sequences (Fig. 3a-c). For each clustering run, we searched with BLAST [15] a random sample of 1 000 representative sequences against all representative sequences of the clustering. We show the cumulative distribution of sequence identities for the best matches that satisfy the minimum coverage threshold of 90% used in the clustering runs. This coverage threshold is favorable for UCLUST since its own coverage criterion is less strict (see Methods, "Clustering criteria"). Due to the heuristic prefiltering methods employed by all tools, none produces a perfect clustering. This limitation is seen most clearly at 50% sequence identity (Fig. 3c), for which Linclust/MMseqs2, UCLUST, Linclust-m80 and Linclust miss 2%, 10%, 16% and 28% of sequence pairs satisfying the clustering threshold.

Cluster consistency analysis. We measure the quality of the clusterings produced by the tools by analyzing the homogeneity of the functional annotation of the sequences in the clusters [16]. We assess Gene Ontology (GO) annotations [17] (Fig. 4a,b) and Pfam domain annotations [18] (Fig. 4c) provided by the UniProt database. For each of these annotations, we averaged two score variants over all clusters, "mean" and a "worst". The "mean" ("worst") score for a cluster is the mean (minimum) annotation similarity score between the representative sequence and all other cluster members, as described in [16].

Overall, the consistencies of cluster annotations are similar for all tools, which is not surprising since they all use exact Smith-Waterman alignments and similar acceptance criteria (**Supplemental Fig. S3**, Methods). However, Linclust/MMseqs2 and Linclust clusterings have better consistencies than UCLUST and CD-HIT according to purely experimentally derived GO annotations (Fig. 4a) and according to Pfam domain annotations (Fig. 4c). This might be either due to a stricter mini-

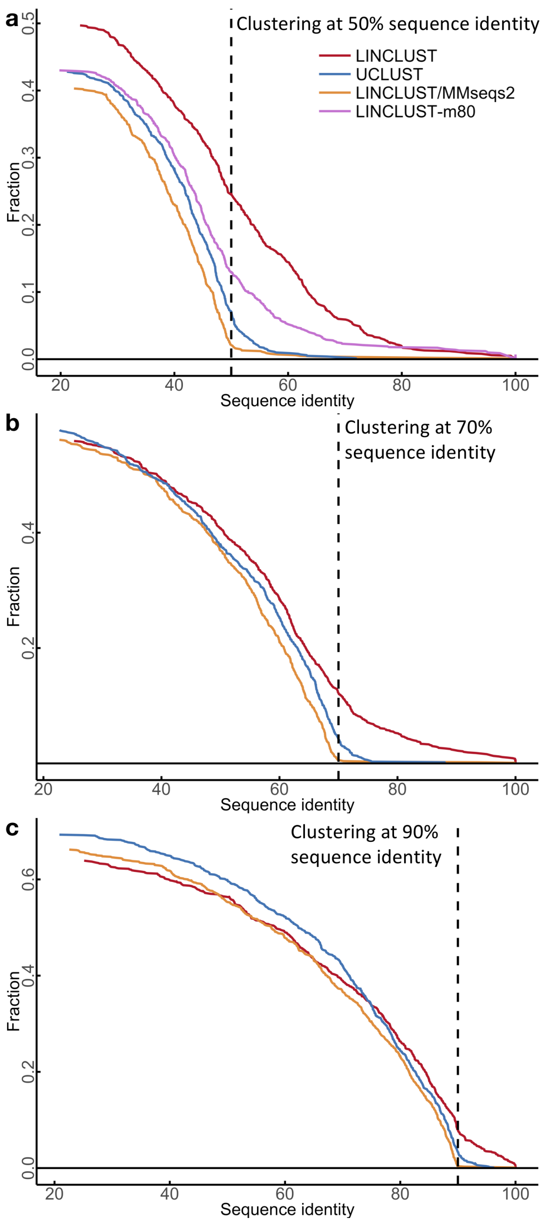


FIG. 3. **Cumulative distance distribution between representative sequences.** We clustered the test set of 123 million sequences at three different sequence identity thresholds (**a-c** at 50%, 70% and 90%, respectively). For each clustering, we randomly sampled 1 000 representative cluster sequences, compared them to all representative sequences of the clustering, and plotted the fraction whose best match (excluding self-matches) with minimum sequence coverage of 90% had a sequence identity above the x-value. The y-value at the clustering threshold (dashed line) is the fraction of false negatives, pairs of sequences whose similarity was overlooked by the clustering method.

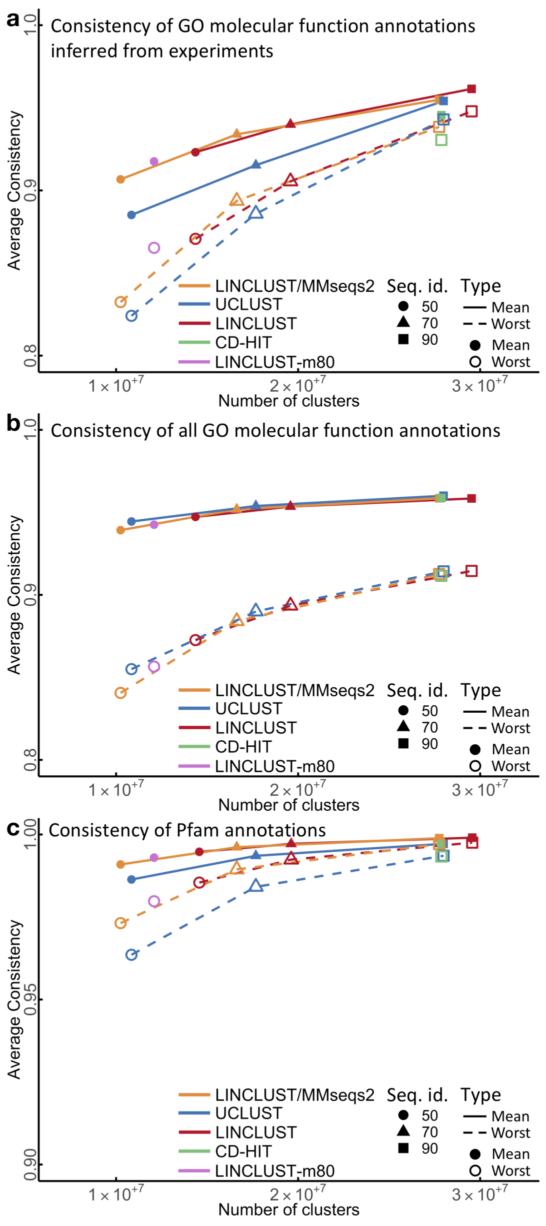


FIG. 4. **Cluster consistency of GO molecular functional and Pfam annotations.** (**a**) Cluster annotation consistency of GO functional annotations inferred from experiments (EXP_F). "Mean" and "worst" refers to the mean and the minimum annotation similarity between each representative sequence and all other cluster members . Plotting symbols indicate the sequence identity threshold for clustering. CD-HIT was only run at 90% sequence identity due to run time constraints. Linclust-m80 was only run at 50% sequence identity. (**b**) Same as (a) but using manually and computationally assigned functional GO annotations. (**c**) Consistency of Pfam annotation from the representative sequences to the cluster members.

mum coverage criterion in Linclust or due to its slightly different definition of sequence similarity, which translates the sequence identity threshold into an approximately equivalent threshold for the similarity score of the local alignment divided by the maximum length of the two aligned segments (Methods: Clustering criteria). This similarity measure is more appropriate than sequence identity to cluster together sequences with conserved functions, as it also accounts for gaps and for the degree of similarity between aligned residues. The cluster consistencies of all tools are similar when GO annotations based on computational predictions are included (Fig. 4b).

Clustering 1.6 billion metagenomic sequences. As a demonstration of Linclust's ability to cluster huge sets, we applied it to cluster 1.59 billion protein sequence fragments predicted by Prodigal [19] in 2200 metagenomic and metatranscriptomic datasets [3, 20, 21] downloaded mainly from the Joint Genome Institute. We clustered these sequences with a minimum sequence identity of $\geq 50\%$ and minimum coverage of the shorter sequence of 90% (Methods: Clustering criteria), producing 424 million clusters in 10 hours on a 2×14-core server.

Our Metaclust database of 424 million representative sequences will improve the sensitivity of profile sequence searches by increasing the diversity of the underlying multiple sequence alignments. It will thereby raise the fraction of annotatable sequences in genomic and metagenomic datasets [6, 21]. It could also increase the number protein families for which reliable structures can be predicted de novo, as shown by Ovchinnikov et al. [22], who used an unpublished dataset of 2 billion metagenomic sequences. Metaclust should also allow us to predict more accurately the effects of mutations on proteins [23].

DISCUSSION

Clustering a set of N items is challenging when both N and the number of clusters K are large, due to the time complexity of existing clustering algorithms. Hierarchical agglomerative clustering approaches have a time complexity of $O(N^2 \log N)$ [24], others with a predefined number of clusters such as K-means or expectation maximization clustering have complexity $O(NK)$. When both N and K are in the tens of millions, traditional approaches are impracticably slow. Driven by the need to cluster huge datasets in the era of big data, most work has focused on reducing the proportionality constant.

One example is the widely used canopy clustering algorithm [25]. The items are first preclustered into overlapping sets (*canopies*) based on a fast approximate similarity measure. Canopies could be biological sequences sharing the same k-mer or documents sharing a keyword. Some traditional clustering algorithm is run on all items, but with the restriction that slow, exact similarities are only computed between items belonging to the same canopy. Similar to the k-mer prefilter used in CD-HIT, UCLUST, kclust and MMseqs [8, 11, 26, 27], the preclustering reduces the number of comparisons by a large factor F using the slow, exact measure, but the time complexity of the exact distance calculation $O(N^2/F)$ is still quadratic in N. Linear-time clustering algorithms, using for instance hashing techniques, have been proposed [28, 29]. But like the preclustering step in canopy clustering or Linclust's prefilter to find k-mer matches, these algorithms are only approximate. If falsely clustered pairs are costly (e.g. for redundancy filtering), pairwise links need to be verified with the exact similarity measure, which still requires quadratic time complexity. In contrast, Linclust's linear time complexity of $O(mN)$ includes verification of all edges between items using the exact distance measure.

Linclust can be trivially generalized to cluster any items for which a set of m keys per item can be defined such that (1) items belonging to a cluster are likely to share at least one of their keys and (2) items not belonging to a cluster are unlikely to share a key (see Methods, Optimal k-mer length). For clustering documents the keys could be all $m = \binom{n}{k}$ subsets of the n keywords of size k, for example [28]. To achieve a high sensitivity, we could select as centre of the group of items sharing a key the member with the largest sum of sizes of groups it belongs to. In this way, the centre items are able to pull together into the same cluster many items from different groups.

We perform the clustering in step 5 of Figure 1 with the greedy incremental clustering, because it always chooses the longest sequence as the cluster representative. It ensures that the representative sequences, being the longest sequence in each cluster, are likely to contain all protein domains of all cluster members. Our rule in step 2 to choose the longest protein sequence per k-mer group as its centre is well-suited to achieve large clusters, because the longest sequences tend to be selected as centres of most of the k-mer groups they belong to, and these long sequences therefore have edges to most sequences they share k-mers with.

As far as we know, Linclust is the only algorithm that could run on datasets of billions of items resulting in billions of clusters, overcoming the time and memory bottlenecks of existing clustering algorithms. Linclust could therefore be useful for many other applications. We have recently extended Linclust to nucleotide sequences. We are also working on a version to cluster D-dimensional vectors, which could be used, for instance, for metagenomic binning to cluster contigs by their coverage profiles across D metagenomic samples [30].

In summary, we hope the Linclust algorithm will prove helpful to exploit the tremendous value in publicly available metagenomic and metatranscriptomic datasets. Linclust should lead to considerable savings in computing resources in current applications. Most importantly, it will enable previously infeasible large-scale analyses.

METHODS

The Linclust algorithm consists of the following steps (Figure 1):

(1) Generating the table of k-mers. We transform the sequence set 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 (see subsection ”Reduced amino acid alphabet”). The k-mer length is chosen as described in subsection ”Optimal k-mer length” and is typically between 10 and 14.

For each sequence, we extract m k-mers, as described in "Selection of k-mers". Increasing m from its default value of 20 (option `-kmer-per-seq`) increases the sensitivity at the cost of a moderately decreasing speed (Supplemental Fig. S4). 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 a table with mN lines. Therefore, Linclust has a memory footprint of $mN \times 16$ bytes.

(2) Finding exact k-mer matches. We sort this table by the k-mer index using the in-place 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 together sequences into blocks of lines that contain the same k-mer. For each such k-mer group we select the longest sequence as its centre sequence. 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. We further overwrite the k-mer index by the centre sequence identifier and resort the mN lines of the table by the centre sequence identifier. The k-mer match stage results file has one entry for each centre sequence identifier containing the list of identifiers of sequences that share a k-mer with the centre sequence. If a sequence shares multiple k-mer matches with a centre sequence, we keep only the entry with the lowest diagonal $i-j$.

(3a) 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 step 3b and are added to the cluster of their centre sequence after step 5.

(3b) 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 [31].

(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 [32]. 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) Greedy incremental clustering. This algorithm was already implemented for MMseqs [11]. Briefly, the file with the validated directed edges from centre sequences to member sequences is read in and all reverse edges are added. The list of input sequences is sorted by decreasing length. While the list is not yet empty, the top sequence is removed from the list, together with all sequences still in the list that share an edge with it. These sequences form a new cluster with the top sequence as its representative.

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, $\mathrm{MI} = \sum_{x,y=1}^{A} p(x,y)\log_2\left(p(x,y)/p(x)/p(y)\right)$. 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 $A=13$, which performed well over all tested clustering sequence identities from 50% to 100%, merges (L,M), (I,V), (K,R), (E, Q), (A,S,T), (N, D) and (F,Y).

Optimal k-mer length. For optimal results and efficiency, the majority of the sequences in k-mer groups should be homologous to their centre sequence. In other words, the k-mers have to be specific enough for the size of the database, with larger databases requiring larger k. To automatically set a good value of k, a very conservative condition is to limit to 1 the expectation value E_{FP} of the number of sequences per k-mer group that are not homologous to their centre sequence. E_{FP} is equal to the number mN of k-mers selected in the entire sequence set times the probability p_{match} for one of those k-mers to match the k-mer of the k-mer group by chance. If the k-mers were not preselected by their hash function values, this probability would be approximately $1/A_{\mathrm{eff}}^k$, where $1/A_{\mathrm{eff}} = \sum_{a=1}^{A} p_a^2$ is the probability for two letters from the reduced alphabet of size A to match by chance (1/8.7 for $A=13$) and p_a is the frequency of letter a in the database. Due to the preselection, only a fraction $\sim m/L$ of the entire set of k-mers is used, where L is the average sequence length. Therefore, the probability of two selected k-mers to match by chance is $L/(mA_{\mathrm{eff}}^k)$. The condition for the k-mer specificity is $1 \geq E_{\mathrm{FP}} = mNL/(mA_{\mathrm{eff}}^k) = NL/A_{\mathrm{eff}}^k$, and hence we demand $k \geq \lfloor\log(NL)/\log(A_{\mathrm{eff}})\rfloor =: k_{\mathrm{spec}}$. In Linclust, we set $k=\max\{k_{\mathrm{spec}}, k_{\mathrm{seqid}}\}$, with $k_{\mathrm{seqid}}=14$ for a

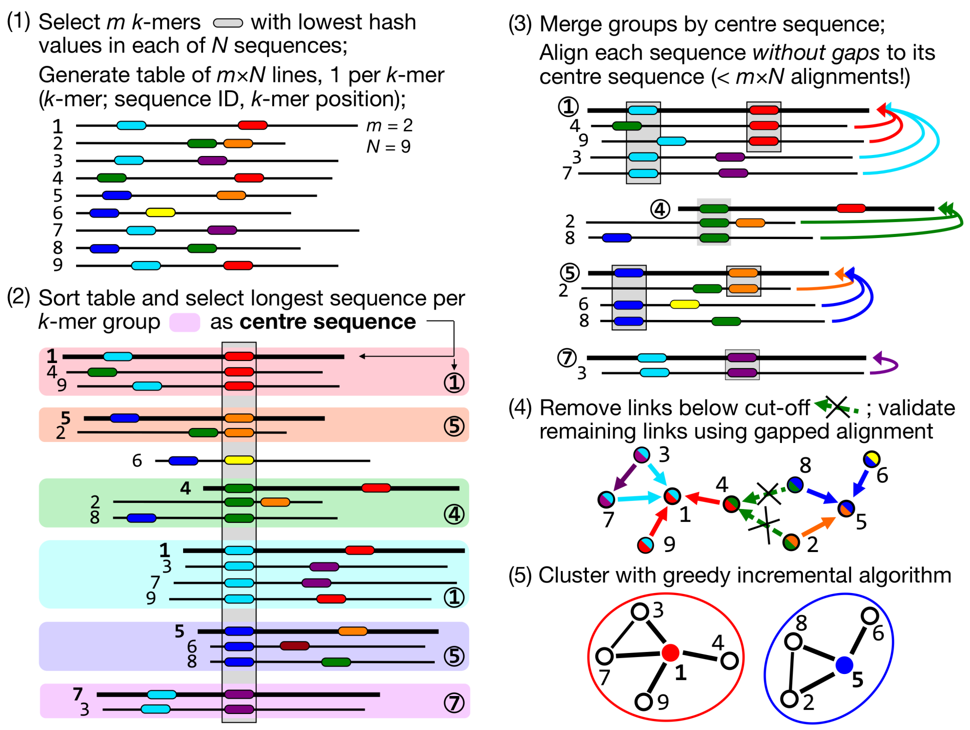

(1) Select *m* *k*-mers with lowest hash values in each of *N* sequences;
Generate table of *m*×*N* lines, 1 per *k*-mer (*k*-mer; sequence ID, *k*-mer position);
m = 2
N = 9
(2) Sort table and select longest sequence per *k*-mer group as **centre sequence**
(3) Merge groups by centre sequence;
Align each sequence *without gaps* to its centre sequence (< *m*×*N* alignments!)
(4) Remove links below cut-off ; validate remaining links using gapped alignment
(5) Cluster with greedy incremental algorithm

FIG. 5. **Linear-time clustering algorithm.** Steps 1 and 2 find exact k-mer matches between the N input sequences that are extended in step 3 and 4. (1) Linclust selects in each sequence the m (default: 20) k-mers with the lowest hash function values, as this tends to select the same k-mers across homologous sequences. It uses a reduced alphabet of 13 letters for the k-mers and sets k between 10 and 14 depending on the sequence set size and the sequence identity threshold. It generates a table in which each of the mN lines consists of the k-mer, the sequence identifier, and the position of the k-mer in the sequence. (2) Linclust sorts the table by k-mer in quasi-linear time, which identifies groups of sequences sharing the same k-mer (large shaded boxes). For each k-mer group, it selects the longest sequence as centre. It thereby tends to select the same sequences as centre among groups sharing sequences. (3) It merges k-mer groups with the same centre sequence together (①: red + cyan and ⑤: orange + blue) and compares each group member to the centre sequence in two steps: by global Hamming distance and by gapless local alignment extending the k-mer match. (4) Sequences above a score cut-off in step 3 are aligned to their centre sequence using gapped local sequence alignment. Sequence pairs that satisfy the clustering criteria (e.g. on the E-value, sequence similarity, and sequence coverage) are linked by an edge. (5) The greedy incremental algorithm finds a clustering such that each input sequence has an edge to its cluster's representative sequence. Note that the number of sequence pairs compared in steps 3 and 4 is less than mN, resulting in a linear time complexity.

sequence identity clustering threshold $\geq 90\%$ and $k_{\text{seqid}} = 10$ otherwise to ensures slightly higher efficiency for high sequence identities, for which longer k-mers are sufficiently sensitive.

Selection of k-mers. To be able to cluster two 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 k-mers such that the same k-mers tend to be extracted from homologous 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. Note that we cannot simply store a subset of $A^k\,m/L$ k-mers to be selected due to its sheer size.

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 their argument, 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. S5).

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.

Clustering datasets that don't fit into main memory. Linclust needs $m \times 16$ bytes of memory per sequence. If the computer's main memory is too small, Linclust automatically splits the k-mer array into C equal-sized chunks small enough to fit each into main memory (Supplemental Fig. S1). For each chunk index $c \in \{0,\ldots,C-1\}$ we run Linclust steps 1 and 2 (Figure 5) normally but extract only k-mers whose numerical index modulo C yields a rest c. This way each of the C runs builds up a k-mer table with only about mN/C lines instead of mN, and hence each run needs C times less memory. Each run writes out a file with all found k-mer groups, and afterwards all C files are merged into a single file such that k-mer groups are sorted by ascending centre IDs. Finally, Linclust steps 3 to 5 are performed as usual.

Parallelization and supported platforms. We used OpenMP to parallelize all stages except the fast step 5 and SIMD instructions to parallelize step 3 and step 4. Linclust supports Linux and Windows, Mac OS X and CPUs with AVX2 or SSE4.1 instructions.

Clustering criteria. Linclust/MMseqs2 and Linclust has three main criteria to link two sequences by an edge: (1) a maximum E-value threshold (option `-e [0,∞[`) computed according to the gap-corrected Karlin-Altschul statistics using the ALP library [33]; (2) a minimum coverage (option `-c [0,1]`, which is defined by the number of aligned residue pairs divided by either the maximum of the length of query/centre and target/non-centre sequences (default mode, `--cov-mode 0`), or by the length of the target/non-centre sequence (`--cov-mode 1`), or by the length of the query/centre (`--cov-mode 2`); (3) a minimum sequence identity (`--min-seq-id [0,1]`) with option `--alignment-mode 3` defined as the number of identical aligned residues divided by the number of aligned columns including internal gap columns, or, by default, defined by a highly correlated measure, the equivalent similarity score of the local alignment (including gap penalties) divided by the maximum of the lengths of the two locally aligned sequence segments. The score per residue *equivalent* to a certain sequence identity is obtained by a linear regression using thousands of local alignments as training set (Fig. S2 in [27]).

The sequence identity in UCLUST is defined as number of identical residues in the pairwise global alignment divided by the number of aligned columns including internal gaps. Due to the global alignment, no explicit coverage threshold is needed. CD-HIT defines sequence identity as the number of identical residues in the local alignment divided by the length of the shorter sequence. Therefore, sequence coverage of the shorter sequence must be at least as large as the sequence identity threshold.

Tools and options for benchmark comparison. Linclust and Linclust/MMseqs2 (commit 5e21868) used the commands `mmseqs linclust --cov-mode 1 -c 0.9 --min-seq-id 0.9` and `mmseqs cluster --cov-mode 1 -c 0.9 --min-seq-id 0.9` for 90%, respectively, and `--min-seq-id 0.7` or `--min-seq-id 0.5` for 70% and 50%. The minimum coverage of 90 % of the shorter sequence was chosen to enforce global similarity, similar to UCLUST and CD-HIT. CD-HIT 4.6 was run 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 (7.0.1090) was run with `--id 0.9, 0.7, 0.5`, for RAPsearch2 (2.23) we used `-z 16`, for DIAMOND (v0.8.36.98) option `--id 0.5`, and for MASH (v2.0) `-s 20 -a -i -p 16`. 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 three tests: one based on (1) experimentally validated GO annotations, (2) general functional GO annotations (mostly inferred from homologous proteins) and (3) Pfam annotations. 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 [16]. 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.

GO annotations often annotate the whole sequence. We used the Pfam annotations of the UniProt to check local consistence of clusters (Fig. 3c). We compared the Pfam domain annotation of the representative sequence against all cluster members. If the member had the exact same domain annotation as the representative sequence we counted it as correct (value=1) and otherwise as false (value=0).

Clustering. We downloaded ∼1800 metagenomic and ∼400 metatranscriptomic datasets with assembled contigs from the Joint Genome institute's IMG/M archive [3] and NCBI's Sequence Read Archive [20] (`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 [19] resulting in 1,595,926,152 proteins.

We clustered the 1.59 million sequence fragments with Linclust using the following acceptance criteria: (1) the minimum sequence identity is 50%, using the score-per-column similarity measure described in Clustering criteria, (2) the shorter of the two sequences has at least 90% of its residues aligned, and (3) the maximum E-value is 10^{-3} (default) (Linclust options: `--min-seq-id 0.5 --cov-mode 1 -c 0.9 --cluster-mode 2`). The clustering step found 424 million cluster within 10 hours on a server with two 14-core Intel Xeon E5-2680 v4 CPUs (2.4 GHz) and 762 GB RAM.

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`.

ACKNOWLEDGEMENTS

We are grateful to Cedric Notredame and Chaok Seok for hosting MS at the CRG in Barcelona and at Seoul National University for 12 and 30 months, respectively. Thanks to Milot Mirdita and Clovis Galiez for discussions and to 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).

AUTHOR CONTRIBUTIONS

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

COMPETING INTERESTS

The authors declare no competing financial interests.

[1] M. S. Rappe and S. J. Giovannoni, "The uncultured microbial majority," *Ann. Rev. Microbiol.*, vol. 57, no. 1, pp. 369–394, (2003).

[2] A. Wilke, J. Bischof, W. Gerlach, E. Glass, T. Harrison, K. P. Keegan, T. Paczian, W. L. Trimble, S. Bagchi, A. Grama, *et al.*, "The MG-RAST metagenomics database and portal in 2015," *Nucleic Acids Res.*, vol. 44, no. D1, pp. D590–D594, (2016).

[3] V. M. Markowitz, I.-M. A. Chen, K. Chu, E. Szeto, K. Palaniappan, M. Pillay, A. Ratner, J. Huang, I. Pagani, S. Tringe, *et al.*, "IMG/M 4 version of the integrated metagenome comparative analysis system," *Nucleic Acids Res.*, vol. 42, no. D1, pp. D568–D573, (2014).

[4] 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.*, vol. 23, no. 1, pp. 9–15, (2012).

[5] N. Desai, D. Antonopoulos, J. A. Gilbert, E. M. Glass, and F. Meyer, "From genomics to metagenomics," *Curr. Opin. Biotechnol.*, vol. 23, no. 1, pp. 72–76, (2012).

[6] T. Prakash and T. D. Taylor, "Functional assignment of metagenomic data: challenges and applications," *Brief. Bioinformatics*, vol. 13, no. 6, pp. 711–727, (2012).

[7] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "CD-HIT: accelerated for clustering the next-generation sequencing data.," *Bioinformatics*, vol. 28, pp. 3150–3152, (2012).

[8] R. C. Edgar, "Search and clustering orders of magnitude faster than BLAST.," *Bioinformatics*, vol. 26, pp. 2460–2461, (2010).

[9] W. Li, L. Fu, B. Niu, S. Wu, and J. Wooley, "Ultrafast clustering algorithms for metagenomic sequence analysis.," *Brief. Bioinform.*, vol. 13, pp. 656–668, (2012).

[10] M. Steinegger and J. Söding, "MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets," *Nature Biotechnol.*, vol. 35, pp. 1026–1028, (2017).

[11] M. Hauser, M. Steinegger, and J. Söding, "MMseqs software suite for fast and deep clustering and searching of large protein sequence sets," *Bioinformatics*, vol. 32, no. 9, pp. 1323–1330, (2016).

[12] B. Buchfink, C. Xie, and D. H. Huson, "Fast and sensitive protein alignment using diamond," *Nature Methods*, vol. 12, no. 1, pp. 59–60, (2015).

[13] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: fast genome and metagenome distance estimation using minhash," *Genome Biology*, vol. 17, p. 132, (2016).

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

[15] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool.," *J. Mol. Biol.*, vol. 215, pp. 403–410, (1990).

[16] M. Mirdita, L. von den Driesch, C. Galiez, M. J. Martin, J. Söding, and M. Steinegger, "Uniclust databases of clustered and deeply annotated protein sequences and alignments," *Nucleic Acids Res.*, vol. 45, no. D1, pp. D170–D176, (2017).

[17] Gene Ontology Consortium, "Gene Ontology Consortium: going forward.," *Nucleic Acids Res.*, vol. 43, no. D1, pp. D1049–D1056, (2015).

[18] R. D. Finn, P. Coggill, R. Y. Eberhardt, S. R. Eddy, J. Mistry, A. L. Mitchell, S. C. Potter, M. Punta, M. Qureshi, A. Sangrador-Vegas, *et al.*, "The pfam protein families database: towards a more sustainable future," *Nucleic Acids Res.*, vol. 44, no. D1, pp. D279–D285, (2016).

[19] D. Hyatt, G.-L. Chen, P. F. LoCascio, M. L. Land, F. W. Larimer, and L. J. Hauser, "Prodigal: prokaryotic gene recognition and translation initiation site identification," *BMC Bioinformatics*, vol. 11, no. 1, p. 119, (2010).

[20] Y. Kodama, M. Shumway, and R. Leinonen, "The sequence read archive: explosive growth of sequencing data," *Nucleic Acids Res.*, vol. 40, no. D1, pp. D54–D56, (2012).

[21] 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*, vol. 348, no. 6237, pp. 1261359–1–9, (2015).

[22] S. Ovchinnikov, H. Park, N. Varghese, P.-S. Huang, G. A. Pavlopoulos, D. E. Kim, Kamisetty, Hetunandan, N. C. Kyrpides, and D. Baker, "Protein structure determination using metagenome sequence data," *Science*, vol. 355, pp. 294–298, (2017).

[23] T. A. Hopf, J. B. Ingraham, F. J. Poelwijk, C. P. Schärfe, M. Springer, C. Sander, and D. S. Marks, "Mutation effects predicted from sequence co-variation," *Nature Biotechnol.*, vol. 35, pp. 128–135, (2017).

[24] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *J. Classification*, vol. 1, no. 1, pp. 7–24, (1984).

[25] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," *Proc. sixth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pp. 169–178, (2000).

[26] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences.," *Bioinformatics*, vol. 22, pp. 1658–1659, (2006).

[27] M. Hauser, C. Mayer, and J. Soding, "kClust: fast and sensitive clustering of large protein sequence databases," *BMC Bioinformatics*, vol. 14, pp. 248+, (2013).

[28] J. A. Marshall and L. C. Rafsky, "Exact clustering in linear time," *Preprint at https://arxiv.org/abs/1702.05425*, (2017).

[29] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *Preprint at https://arxiv.org/abs/1408.2927*, (2014).

[30] M. Albertsen, P. Hugenholtz, A. Skarshewski, K. L. Nielsen, G. W. Tyson, and P. H. Nielsen, "Genome sequences of rare, uncultured bacteria obtained by differential coverage binning of multiple metagenomes," *Nature Biotechnol.*, vol. 31, no. 6, pp. 533–538, (2013).

[31] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh, "The Universal Protein Resource (UniProt)," *Nucleic Acids Res.*, vol. 33, no. Suppl. 1, pp. D154–D159, (2005).

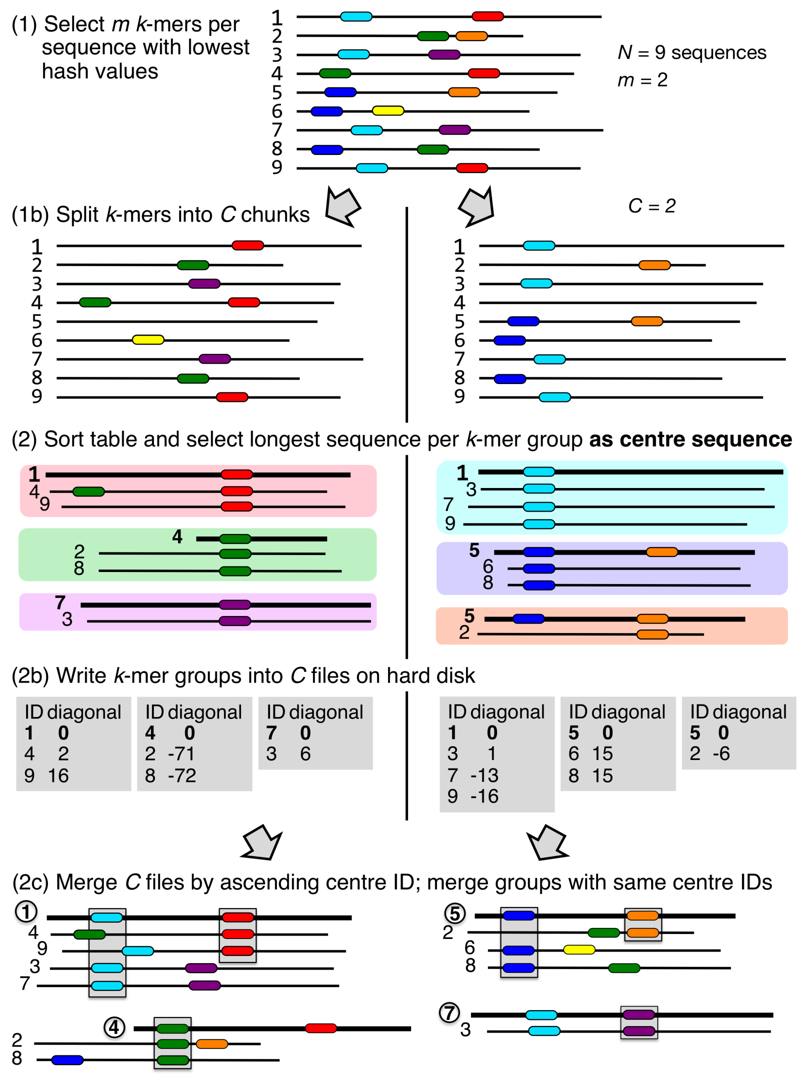
[32] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth, "SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications," *PLoS One*, vol. 8, (2013).

[33] S. Sheetlin, Y. Park, M. C. Frith, and J. L. Spouge, "ALP & FALP: C++ libraries for pairwise local alignment E-values," *Bioinformatics*, vol. 32, no. 2, pp. 304–305, (2015).

Clustering huge protein sequence sets in linear time

Martin Steinegger[1,2,3] **& 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, Department of Chemistry, [3]Seoul National University, Seoul, Korea


(1) Select *m* *k*-mers per sequence with lowest hash values
1
2
3
4
5
6
7
8
9
N = 9 sequences
m = 2
(1b) Split *k*-mers into *C* chunks
C = 2
(2) Sort table and select longest sequence per *k*-mer group **as centre sequence**
(2b) Write *k*-mer groups into *C* files on hard disk
(2c) Merge *C* files by ascending centre ID; merge groups with same centre IDs
①
④
⑤
⑦

ID	diagonal
1	**0**
4	2
9	16

ID	diagonal
4	**0**
2	-71
8	-72

ID	diagonal
7	**0**
3	6

ID	diagonal
1	**0**
3	1
7	-13
9	-16

ID	diagonal
5	**0**
6	15
8	15

ID	diagonal
5	**0**
2	-6

Figure S 1. Splitting the database into chunks For each chunk index $c \in \{0, \ldots, C-1\}$ we run Linclust steps 1 and 2 (Figure 1) normally but extract only k-mers whose numerical index modulo C yields a rest c. This way each of the C runs builds up a k-mer table with only about mN/C lines instead of mN, and hence each run needs C times less memory. Each run writes out a file with all found k-mer groups (sequence ID, diagonal). The diagonal is $i-j$ of the k-mer match, where i is the position of the groups k-mer in the centre sequence and j the position in the other sequence. Afterwards all C files are merged into a single file such that k-mer groups are sorted by ascending centre IDs. Finally, Linclust steps 3 to 5 are performed as usual.

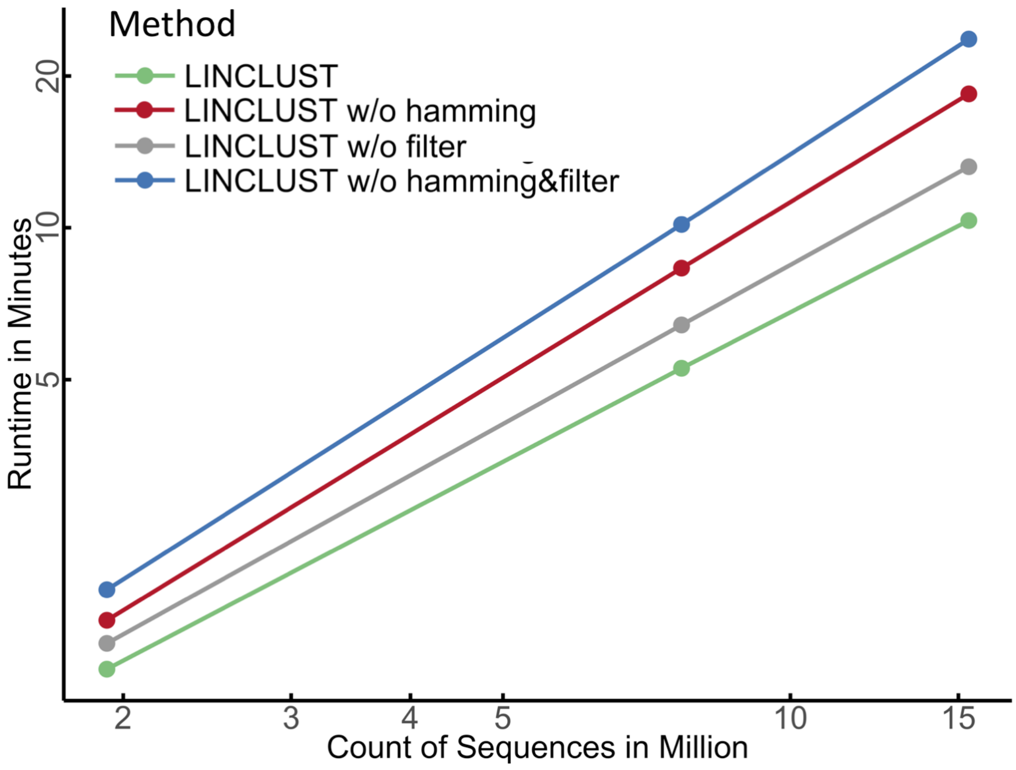


Figure S 2. **Contribution of Hamming distance pre-clustering and gapless local alignment filter steps to the decrease of the Linclust runtime** Double-logarithmic plot of runtimes versus sequence set size illustrating the contribution of runtime decrease for the Hamming distance pre-clustering and ungapped alignment filtering stage.

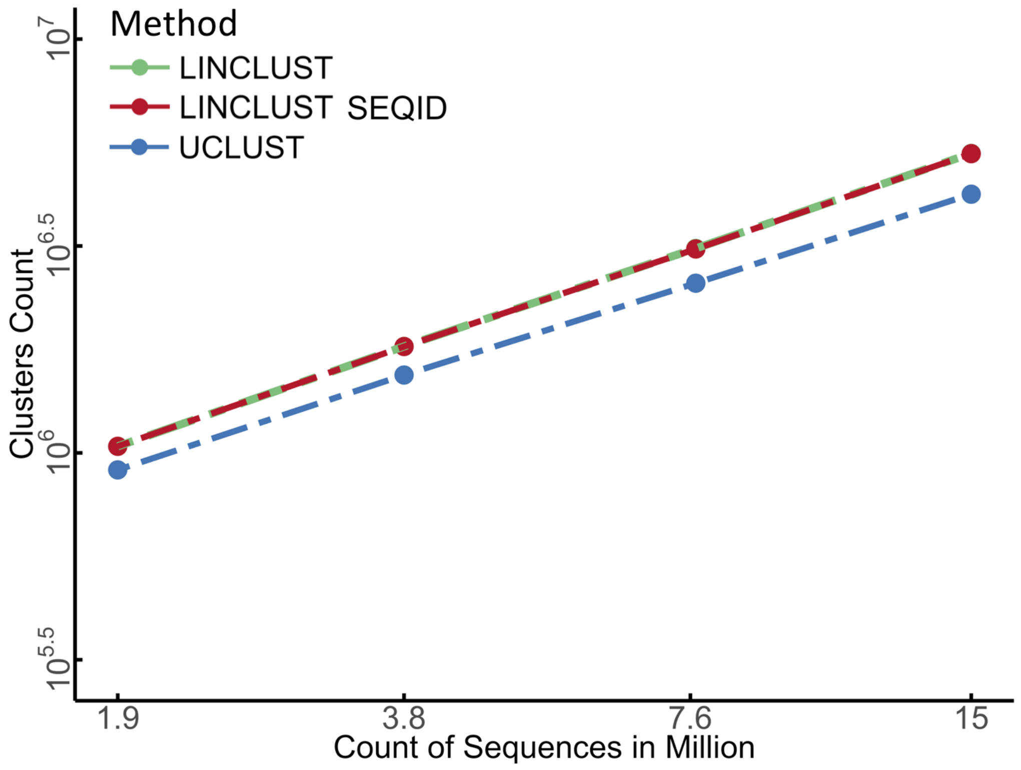


Figure S 3. Equivalence of two sequence identity measures. Number of clusters obtained at 50% sequence identity for Linclust with two different sequence identity definitions: (1) the default definition ("Linclust"), based on the local alignment score divided by the maximum length of the two aligned sequence segments; and (2) the fraction of identical residues in the alignment relative to the number of aligned columns including gaps ("Linclust seqid"). Both measures produce very similar numbers of clusters.

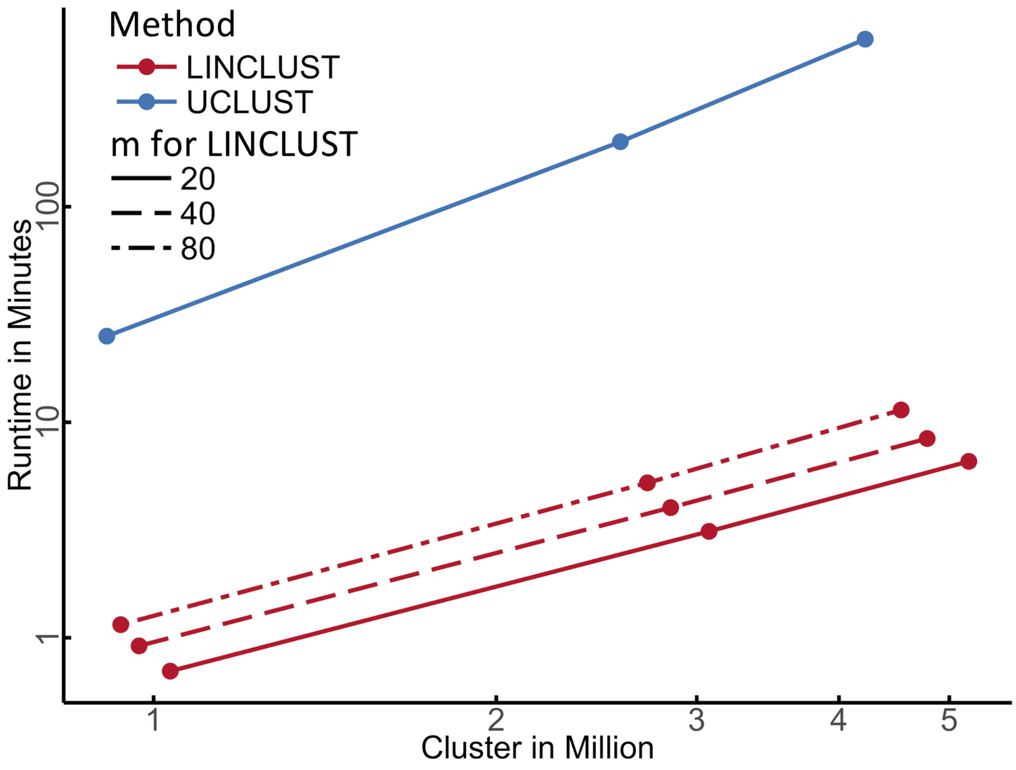


Figure S 4. **Influence of the number m of k-mers extracted per sequence.** Double-logarithmic plot of runtimes versus cluster size. Through the parameter m, the number of k-mers selected per sequence, the user can set the trade-off between sensitivity and speed. At $m = 80$ the runtime of Linclust increases by a factor of 1.66 over the default setting $m = 20$ while producing 8% fewer clusters. At $m = 80$, Linclust generates only 7% more clusters than UCLUST while still being much faster.

```c
// (c) 2017 Johannes Soeding & Martin Steinegger, Gnu Public License version 3
// Rotate left macro: left circular shift by numbits within 16 bits

#define RoL(val, numbits) (val << numbits) ^ (val >> (16 - numbits))

// Transform each letter x[i] to a fixed random number RAND[x[i]]
// to ensure instantaneous mixing into the 16 bits
// Do XOR with RAND[x[i]] and 5-bit rotate left for each i from 1 to k
unsigned circ_hash(const int * x, unsigned length){
 short unsigned RAND[21] = {0x4567, 0x23c6, 0x9869, 0x4873, 0xdc51, 0x5cff, 0x944a, 0x58ec,
                            0x1f29, 0x7ccd, 0x58ba, 0xd7ab, 0x41f2, 0x1efb, 0xa9e3, 0xe146,
                            0x007c, 0x62c2, 0x0854, 0x27f8, 0x231b}; // 16 bit random numbers
 short unsigned h = 0x0;
 h = h^ RAND[x[0]];                  // XOR h and ki
 for (int i = 1; i < length; ++i){
   h = RoL(h, 5);
   h ^= RAND[x[i]];                  // XOR h and ki
 }
 return h;
}

// Rolling hash variant for previous hash function:
// Computes hash value for next key x[0:length-1] from previous hash value
// hash( x[-1:length-2] ) and x_first = x[-1]
unsigned circ_hash_next(const int * x, unsigned length, int x_first, short unsigned h){
 short unsigned RAND[21] = {0x4567, 0x23c6, 0x9869, 0x4873, 0xdc51, 0x5cff, 0x944a, 0x58ec,
                            0x1f29, 0x7ccd, 0x58ba, 0xd7ab, 0x41f2, 0x1efb, 0xa9e3, 0xe146,
                            0x007c, 0x62c2, 0x0854, 0x27f8, 0x231b}; // 16 bit random numbers
 // undo INITIAL_VALUE and first letter x[0] of old key
 h ^= RoL(RAND[x_first], (5*(length-1)) % 16);
 // circularly permute all letters x[1:length-1] to 5 positions to left
 h =  RoL(h, 5);
 // add new, last letter of new key x[1:length]
 h ^= RAND[x[length-1]];
 return h;
}
```

Figure S 5. k-mer hashing function implemented in C.

Chapter 5

Uniclust databases of clustered and deeply annotated protein sequences and alignments

5.1 Introduction

Uniclust is a collection of three clustered and multiple sequence alignment databases to be designed as a resource for highly sensitive protein sequence analysis. The first three, Uniclust90, Uniclust50, Uniclust30, are generated from the UniProt sequence database by clustering sequences together that have more than 90%, 50% and 30% maximum pairwise sequence identity, respectively. We compared the clustering qualities of our databases with those of Uniref90 and Uniref50 (Suzek et al. 2007) using three scores, which measure the coherence of keyword annotations, GO annotations (Gene Ontology Consortium 2015) and protein names among the sequence members of clusters. The Uniclust90 and Uniclust50 create more consistent clusters compared to the Uniref databases. The improved quality of clustering over the Uniref (Suzek et al. 2007) is owed to a new clustering tool, MMseqs2 (Steinegger and Söding 2017), which is much more sensitive and faster than CD-HIT (Li and Godzik 2006), the tool used for clustering the UniRef database.

The sequences in Uniclust are annotated with our highly sensitive HHblits homology search tool for matches to Pfam (Finn et al. 2016) and SCOP (Murzin et al. 1995) domains and to protein sequences from the PDB (Kouranov et al. 2006) database. The high sensitivity of HHblits (Remmert et al. 2011) affords a deeper annotation than seen in UniProt, which uses the less sensitive HMMER3 (Eddy 2009) software for annotation with PFAM, SUPERFAMILY (Wilson et al. 2007) and other databases. Whereas in Uniprot only 73.4% and 51.1% of sequences are annotated as containing a match to Pfam or CATH (Orengo et al. 1997), in Uniclust 82.1%, 74.6% and 66.2% of sequences are annotated by a match to the Pfam, PDB, and SCOP database, respectively.

Our server also offers the Uniboost30, Uniboost20 and Uniboost10 databases of multiple sequence alignments, which are obtained by adding local sequence matches from profile searches to the multiple sequence alignments of each Uniclust30 cluster. These databases are supported by the new release of HH-suite3 and will allow us to improve the sensitivity of HHblits searches by between 20% and 50% on a standard SCOP20 benchmark set (to be published). These alignment databases might also become very useful as a resource for (deep) machine-learning applications that benefit from training on massive amounts of labeled / annotated sequence profiles. HHblits can detect 90% of all Uniprot Pfam annotation on a clan level and 3.5 times more novel annotations.

The Uniclust server provides access to an interactive cluster visualizations. It provides a rich interface for browsing alignments and summaries of taxonomic composition, keywords and protein existence evidence. All databases (Uniclust 90/50/30), Uniboost 30/20/10) and the visualisation can be accessed at `http://uniclust.mmseqs.com/`

Lars von den Driesch performed the evaluation. Milot Mirdita developed the webserver, Lars von den Driesch, Milot Mirdita, Clovis Galiez, Maria Martin, Martin Steinegger and Johannes Söding jointly designed the research. Lars von den Driesch, Milot Mirdita, Clovis Galiez, Martin Steinegger and Johannes Söding wrote the manuscript.

5.2 References

Eddy, S. R. (2009). “A new generation of homology search tools based on probabilistic inference.” In: *Genome informatics. International Conference on Genome Informatics* 23.1, pp. 205–11.

Finn, R. D. et al. (2016). “The Pfam protein families database: towards a more sustainable future”. In: *Nucleic Acids Res.* 44.D1, pp. D279–D285.

Gene Ontology Consortium (2015). “Gene Ontology Consortium: going forward.” In: *Nucleic Acids Res.* 43.D1, pp. D1049–D1056. DOI: 10.1093/nar/gku1179.

Kouranov, A. et al. (2006). “The RCSB PDB information portal for structural genomics.” In: *Nucleic acids research* 34.Database issue, pp. D302–5. DOI: 10.1093/nar/gkj120.

Li, Weizhong and Adam Godzik (2006). “Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences.” In: *Bioinformatics* 22.13, pp. 1658–1659. DOI: 10.1093/bioinformatics/btl158.

Murzin, A. G. et al. (1995). “SCOP: A structural classification of proteins database for the investigation of sequences and structures”. In: *J. Mol. Biol.* 247.4, pp. 536–540. DOI: http://dx.doi.org/10.1016/S0022-2836(05)80134-2.

Orengo, C A et al. (1997). “CATH–a hierarchic classification of protein domain structures.” In: *Structure (London, England : 1993)* 5.8, pp. 1093–1108. DOI: 10.1016/S0969-2126(97)00260-8.

Remmert, M. et al. (2011). “HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment”. en. In: *Nature Methods* 9.2, pp. 173–175. DOI: 10.1038/nmeth.1818.

Steinegger, M. and J. Söding (2017). "MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets". In: *Nature Biotechnology* 35, 1026 EP -. DOI: 10.1038/nbt.3988; https://www.nature.com/articles/nbt.3988#supplementary-information.

Suzek, B. et al. (2007). "UniRef: comprehensive and non-redundant UniProt reference clusters." In: *Bioinformatics* 23.10, pp. 1282–1288. DOI: 10.1093/bioinformatics/btm098.

Wilson, D. et al. (2007). "The SUPERFAMILY database in 2007: families and functions." In: *Nucleic acids research* 35.Database issue, pp. D308–13. DOI: 10.1093/nar/gkl910.

5.3 Journal article

D170–D176 *Nucleic Acids Research, 2017, Vol. 45, Database issue* *Published online 29 November 2016*
doi: 10.1093/nar/gkw1081

Uniclust databases of clustered and deeply annotated protein sequences and alignments

Milot Mirdita[1,†], Lars von den Driesch[1,2,†], Clovis Galiez[1], Maria J. Martin[2], Johannes Söding[1,*] and Martin Steinegger[1,3,4,*]

[1]Quantitative and Computational Biology Group, Max Planck Institute for Biophysical Chemistry, Göttingen, Germany, [2]European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Cambridge, UK, [3]Department for Bioinformatics and Computational Biology, Technische Universität München, Munich, Germany and [4]Department of Chemistry, Seoul National University, Seoul, Korea

Received August 15, 2016; Revised October 14, 2016; Editorial Decision October 24, 2016; Accepted November 01, 2016

ABSTRACT

We present three clustered protein sequence databases, Uniclust90, Uniclust50, Uniclust30 and three databases of multiple sequence alignments (MSAs), Uniboost10, Uniboost20 and Uniboost30, as a resource for protein sequence analysis, function prediction and sequence searches. The Uniclust databases cluster UniProtKB sequences at the level of 90%, 50% and 30% pairwise sequence identity. Uniclust90 and Uniclust50 clusters showed better consistency of functional annotation than those of UniRef90 and UniRef50, owing to an optimised clustering pipeline that runs with our MMseqs2 software for fast and sensitive protein sequence searching and clustering. Uniclust sequences are annotated with matches to Pfam, SCOP domains, and proteins in the PDB, using our HHblits homology detection tool. Due to its high sensitivity, Uniclust contains 17% more Pfam domain annotations than UniProt. Uniboost MSAs of three diversities are built by enriching the Uniclust30 MSAs with local sequence matches from MMseqs2 profile searches through Uniclust30. All databases can be downloaded from the Uniclust server at uniclust.mmseqs.com. Users can search clusters by keywords and explore their MSAs, taxonomic representation, and annotations. Uniclust is updated every two months with the new UniProt release.

INTRODUCTION

The number of protein sequences in public databases such as UniProt (1) or GenBank (2) is growing fast, in part due to various large-scale genomics projects (3–5). The rapid growth makes it attractive for many applications to work with representative subsets, in which the representatives are computed by clustering similar sequences together and choosing only a single representative per cluster. Apart from saving computational resources, the more even coverage of sequence space of such clustered databases can improve the sensitivity of sequence similarity searches (6–8).

The popular UniProt Reference Clusters (UniRef) (9) consist of three databases that are generated by clustering the UniProtKB sequences in three steps using the CD-HIT software (10): UniRef100 combines identical UniProtKB sequences and fragments with 100% sequence identity into common entries. UniRef90 sequences are obtained by clustering UniRef100 sequences together that have at least 90% sequence identity and 80% sequence length overlap, and UniRef50 clusters together UniRef90 sequences with at least 50% sequence identity and 80% sequence length overlap.

Here, we introduce the Uniclust sequence databases which, like UniRef, are clustered, representative sets of UniProtKB sequences at three different clustering levels. But whereas UniRef relies on the CD-HIT software for the clustering, we use our software suite MMseqs2 (github.com/soedinglab/mmseqs2, Steinegger & Söding, to be published). The following characteristics make Uniclust databases unique and useful: First, the sensitivity of MMseqs2 for distantly homologous sequences allows us to cluster the UniProtKB down to 30% sequence identity. Second, we have developed a cascaded clustering workflow within MMseqs2 in order to produce sequence clusters that are as compact and functionally homogeneous as possible. As a result, Uniclust90 and Uniclust50 clusters show higher functional consistency scores than UniRef90 and UniRef50 at similar clustering depths, respectively. Third, we provide deep annotation of Uniclust sequences with Pfam (11) and SCOP (12) domains, and matches to PDB sequences (13)

*To whom correspondence should be addressed. Email: martin.steinegger@mpibpc.mpg.de
Correspondence may also be addressed to Johannes Söding. Tel: +49 551 201 2890; Email: soeding@mpibpc.mpg.de
†These authors contributed equally to the paper as first authors.

using HH-suite, our remote homology detection software suite. The sensitivity of HH-suite allows us to annotate 17% more Pfam domains than UniProt, which uses InterPro and HMMER3 for these annotations. Fourth, we provide the MSAs of all Uniclust clusters as well as the three Uniboost databases with MSAs of different diversity levels that are obtained by enriching Uniclust30 clusters with local sequence matches.

MATERIALS AND METHODS

We developed an open-source bash pipeline (github.com/soedinglab/uniclust-pipeline) to generate all data described here: the Uniclust clusterings, cluster summary headers, domain annotations for sequences, and the Uniboost databases of multiple sequence alignments. We provide the pipeline scripts as a supplementary archive file to avoid cluttering the descriptions here with command line options and other details irrelevant for the understanding.

Uniclust clustering pipeline

The Uniclust clusters contain all sequences in the UniProt knowledge base (UniProtKB), the union of the Swiss-Prot and TrEMBL databases. Sequences longer than 14 000 amino acid residues are split into multiple individual entries to limit memory usage and improve compatibility with other tools. (This affects 352 sequences in the 2016_03 release.) Once a year we will cluster these sequences from scratch as described in the following.

In order to cluster together sequences of $\geq$30% pairwise sequence identity, we need high sensitivity, yet the enormous number of pairwise comparisons (on the order of $(10^7)^2$) requires very high speed at the same time. We developed a cascaded clustering workflow in MMseqs (14) that uses three clustering steps with progressively increasing sensitivity and decreasing speed.

The first step consists of an extremely fast redundancy filtering that can cluster sequences of identical length and 100% overlap (‘mmseqs clusthash’). It reduces each sequence to a five-letter alphabet, computes a 64 bit CRC32 hash value for the full-length sequences, and places sequences with identical hash code that satisfy the sequence identity threshold into the same cluster. This step is run with a threshold of 90% and reduces the 61 million sequences of UniProtKB 2016_03 down to 40 million clusters in ~20 min on a single 16-core node.

Similar to the UniRef100 clustering, we cluster fragments of sequences together with their full-length sequences. We add sequences to a cluster if they have at least 90% sequence identity to the representative sequence and are also covered by at least 95% of their length, without regard to the E-value.

In the first cascaded clustering step, in which we generate the Uniclust90 sequence set, we use the simple greedy clustering strategy of CD-HIT (10) that was already part of MMseqs. We assign a sequence to a cluster if it has at least 90% sequence identity with the representative sequence of the cluster and a sequence length overlap of 90% of the shorter of the two sequences. Similar to the UniRef100 clustering, to cluster fragments of sequences together with their full-length sequences we also add sequences to a cluster if they have at least 90% sequence identity to the representative sequence and are also covered by at least 95% of their length, without regard to the *E*-value.

In the third step, we generate the Uniclust50 and Uniclust30 clustering both directly from the sequences in Uniclust90, using a 50% or 30% sequence identity threshold, respectively, and a minimum sequence length overlap of 80%. A high minimum overlap ensures that all proteins within one cluster have the same or a very similar domain structure and is also an effective criterion to achieve functional homogeneity (15). We avoided the cascaded clustering approach of generating Uniclust30 from Uniclust50 because we found this resulted in slightly inferior clustering quality to the direct approach.

In addition to the simple greedy clustering, we implemented affinity propagation, depth-*n* single linkage clustering, and the classic greedy set-cover algorithm in MMseqs2 and compared the clustering qualities. We found that the cluster compactness for all algorithms could be further improved by passing over all sequences after the clustering and reassigning each to the cluster whose representative sequence is most similar to it. The greedy set-cover algorithm with sequence reassignment gave best results and is therefore used in the final clustering step. The three-step clustering took 5 days on 10 nodes with two Intel Xeon E5-2640 v3 CPUs and 128GB main memory each.

Updating Uniclust. We will update the Uniclust databases every two months following the new UniProt release. To keep the cluster identifiers stable between updates, we do not recluster from scratch but instead update the clustering incrementally, add new sequences to existing clusters, create new clusters, and remove deprecated sequences (14). We employ the updating workflow ‘mmseqs clusterupdate’ in the MMseqs2 package for that purpose, which has the added advantage of running in linear time instead of quadratic in the number of sequences. To avoid excessive computational demands, we recompute the MSAs and sequence annotations only during the reclustering step once per year and for major UniProt releases.

Consensus sequences and representative sequences. We provide two FASTA-formatted files for each of the Uniclust databases (see section Files for Download). One contains the representative sequences and the other the consensus sequences of each cluster. Consensus sequences are computed by running ‘mmseqs result2profile’. The headers of the consensus sequence summarize the annotations of the cluster’s member sequences with the top five non-redundant descriptions, giving precedence to Swiss-Prot over TrEMBL annotations and a low rank to descriptions containing *hypothetical, unknown* etc.

Uniboost MSAs

For many applications such as secondary structure prediction, more diverse MSAs produce more accurate results. Due to the stringent sequence length overlap criterion that ensures functional homogeneity of the Uniclust30 clusters, they contain only 6 sequences on average. We therefore enrich the Uniclust30 MSAs with local sequence matches to boost their diversity. We add local alignment matches through highly sensitive iterative profile-sequence homology searches using four iterations of MMseqs2 through the database of Uniclust30 consensus sequences.

The resulting MSAs are filtered to adjust the diversity: Sequences with a BLOSUM62 score per aligned residue to the consensus sequence of less than 0.0, 0.5 and 1.1 are removed from the MSAs of the Uniboost10, Uniboost20, and Uniboost30 databases, respectively. These values were heuristically found to correspond to 10%, 20% and 30% sequence identity.

Deep domain annotations

We first annotate Uniclust30 MSAs with matches to Pfam-A, SCOP domains, and to the PDB structure database, using our remote homology detection software HHblits, which is based on pairwise comparison of profile hidden Markov models (HMMs). Hence, we compute HMMs for Uniclust30 clusters from the corresponding Uniboost10 MSAs and search the Pfam-A, SCOP, and pdb70 databases of the HH-suite (16). These profile HMM databases are automatically kept up to date by our HH-suite server (e.g. weekly for the PDB).

To avoid multiple annotations of a region with matches to the same database, the pipeline processes matches in the order of increasing *E*-value. It accepts matches as annotation if their *E*-value is <0.01 and the database match overlaps by <10% of its aligned residues with already annotated regions.

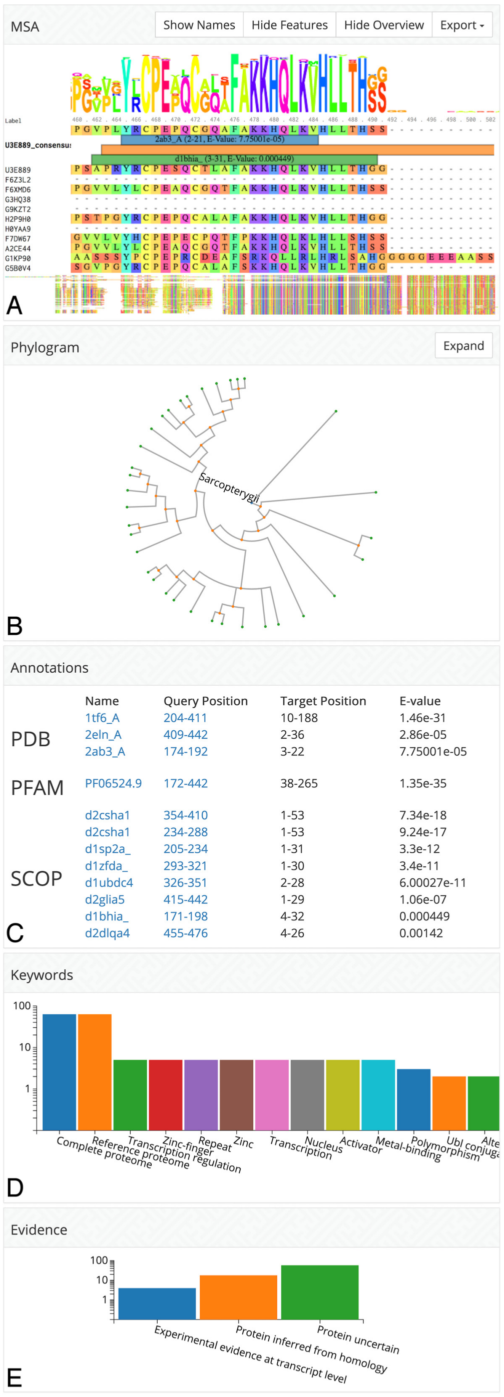
The pipeline annotates UniProt sequences by transferring annotations of Uniclust30 MSAs to their member sequences. We need to ensure that the annotation refers to a region of the member sequence that is homologous to the annotated consensus sequence of the cluster. We therefore only transfer the cluster annotation to the member sequence if the *E*-value for the subalignment E_{subali} is less than 0.01: $E_{\text{subali}} = E_{\text{domain}} + K * \text{length\_consensus} * e^{\lambda s_{\text{subali}}} < 0.01$. Here E_{domain} is the HHblits *E*-value of the domain match, s_{subali} is the BLOSUM62 score of the pairwise subalignment between the consensus sequence and the member sequence overlapping the database match, and the term including s_{subali} is an *E*-value computed with the Karlin-Altschul statistic (17).

Webserver

To investigate specific clusters and get familiar with the information contained in the Uniclust and Uniboost databases, we have set up a web server that offers interactive features using modern web standards and framework such as the D3.js visualization toolkit (18).

The server can perform a full-text search for keywords and sequence identifiers of over a hundred biological databases linked to UniProt entries. Searches will give a list of clusters as result, each linking to a cluster page.

The cluster page shows (Figure 1): (i) an interactive BioJS alignment viewer (19), which displays our Pfam, SCOP, and PDB annotations as colored bars on top; (ii) an expandable taxonomic tree (20) of the species represented in the



	Name	Query Position	Target Position	E-value
PDB	1tf6_A	204-411	10-188	1.46e-31
	2eln_A	409-442	2-36	2.86e-05
	2ab3_A	174-192	3-22	7.75001e-05
PFAM	PF06524.9	172-442	38-265	1.35e-35
SCOP	d2csha1	354-410	1-53	7.34e-18
	d2csha1	234-288	1-53	9.24e-17
	d1sp2a_	205-234	1-31	3.3e-12
	d1zfda_	293-321	1-30	3.4e-11
	d1ubdc4	326-351	2-28	6.00027e-11
	d2glia5	415-442	1-29	1.06e-07
	d1bhia_	171-198	4-32	0.000449
	d2dlqa4	455-476	4-26	0.00142

Figure 1. Visualization of a cluster with the multiple sequence alignment including domain annotations, the taxonomic tree for the species of the cluster's member sequences, domain annotations, summary of sequence annotation keywords, and protein evidence values.

cluster, in which the user can select sequences in the alignment viewer above; (iii) a list of annotations with links to the matched PDB, SCOP and Pfam entries; (iv) keywords occurring most frequently in the annotations; (v) a summary of protein evidence codes. Once a cluster is accessed, the URL will be stable and permanently available.

In the example of Figure 1, the keyword summary indicates that many member sequence are annotated as zinc fingers involved in transcription regulation. By following the links of the Pfam and SCOP domains they are revealed to be zinc finger domains.

Cluster evaluation

In order to compare the functional homogeneity of the sequences within the same cluster, we developed scores that assess the consistency of Gene Ontology terms, keyword annotations, and protein names within the clusters. For each of these three annotation types, we defined a 'worst' and a 'mean' annotation consistency score. These are, respectively, the minimum and the mean of all pairwise annotation similarities between the representative sequence and any other sequence in the cluster. (We checked that the same results are obtained if we compare with a randomly picked sequence per cluster instead of the representative one.) This gives us 2 × 3 scores. We also compute the *average* score over the three annotation types (Figure 2B).

We now explain how the evaluation procedure computes the annotation similarities between two sequences for the three annotation types. Since there are often several Gene Ontology and keyword annotations per protein, we need similarity scores that compare the lists of annotations of two proteins.

Gene Ontology score. The Gene Ontology (21) is a widely used system to describe the functions of genes. It consists of three parts, to classify biological processes, cellular components in which a protein occurs, and their molecular functions. For each of these three categories, the GO annotations are organised into a hierarchical, multi-branch tree. The similarity between two Gene Ontology terms a and b is computed as proposed in (22): $\mathrm{sim}(a, b) = 2\log P(\mathrm{LCA}(a, b))/(\log P(a) + \log P(b))$, where $P(a)$ is the probability of a protein to be annotated with a and $\mathrm{LCA}(a, b)$ (LCA for Last Common Ancestor) is the most specific annotation node in the ontology tree that contains both a and b as child nodes.

Many proteins have multiple GO-term annotations. To obtain a GO annotation similarity value $\mathrm{sim}(x, y)$ between two proteins x and y with lists of GO annotations $A(x)$ and $A(y)$, we follow (23) and define the similarity between an annotation a of one protein with the annotation A_y of another, $\mathrm{sim}(a, A_y) := \max\{\mathrm{sim}(a, b){:}\ b \in A_y\}$ and using it, we define the annotation similarity between proteins x and y: $\mathrm{sim}(x, y) = (\sum_{a\in A_x} \mathrm{sim}(a, A_y) + \sum_{b\in A_y} \mathrm{sim}(A_x, b))/(|A_x| + |A_y|)$. Note that this similarity takes values between 0 and 1 and equals 1 if and only if $A_x = A_y$.

Keyword score. Most keywords with which UniProt proteins are annotated were originally defined manually by database curators. They are automatically transferred to

Table 1. Statistics of Uniclust databases

Database	Clusters	Singletons	Average cluster size
Uniclust90	30.9 M	23.8 M	2.0 (5.4)
Uniclust50	13.5 M	9.6 M	4.6 (13.4)
Uniclust30	9.7 M	7.0 M	6.3 (19.8)

Average cluster sizes are for all clusters and, in parentheses, for non-singleton clusters.

homologous proteins according to various rules developed within UniProt (1). The keyword annotation similarity between two proteins x, y with keyword lists K_x and K_y is defined in the exact same way as the GO annotation similarity while defining $\mathrm{sim}(a, b) = I(a = b)$, with indicator function $I(\cdot)$. This yields $\mathrm{sim}(x, y) = 2|K_x \cap K_y|/(|K_x| + |K_y|)$. The keywords in the UniProt knowledge base describe functional features in categories such as molecular function, domain, biological process, ligand and cellular component. We ignore keyword categories technical term and coding sequence diversity, and keywords provided by the UniProt automatic annotation team that do not describe biological functions.

Protein name score. We compute the Levenshtein string edit distance between the protein name from the 'recommended name' section and normalise by the length of the longer protein name to get a similarity between 0 and 1. The calculation ignores protein name entries starting with the words uncharacterized, putative, potential, probable, inactive, likely, and unknown. Additionally, we remove the uninformative word 'protein' from the names.

RESULTS AND DISCUSSION

Statistics

Table 1 shows statistics for the release 2016_03 of the Uniclust databases, which is based on the UniProt 2016_03 with 61 522 041 sequences of 325 amino acids average length.

Clustering quality

To assess the functional homogeneity of the clusters we evaluated the mean and worst sequence identities over all clusters as measures of cluster compactness. We computed those through Clustal Omega distance matrices by running 'clustalo –distmat-out=distance-matrix –percent-id –full –full-iter' on all clusters. If a cluster contains more than ten sequences we sample ten random sequences for the distance matrix. Figure 2A shows these mean and worst cluster compactness values. Despite the UniRef using sequence identity and Uniclust using score-per-aligned-residue pair as similarity criterion during clustering, the Uniclust clusters have higher mean and minimum sequence identities.

Additionally to the cluster compactness we computed the annotation consistency for all clusters with respect to the Gene Ontology annotation of member sequences, the keyword consistency and the protein name consistency of each cluster's member sequences (Materials and Methods). For each annotation type we analysed the worst and the average

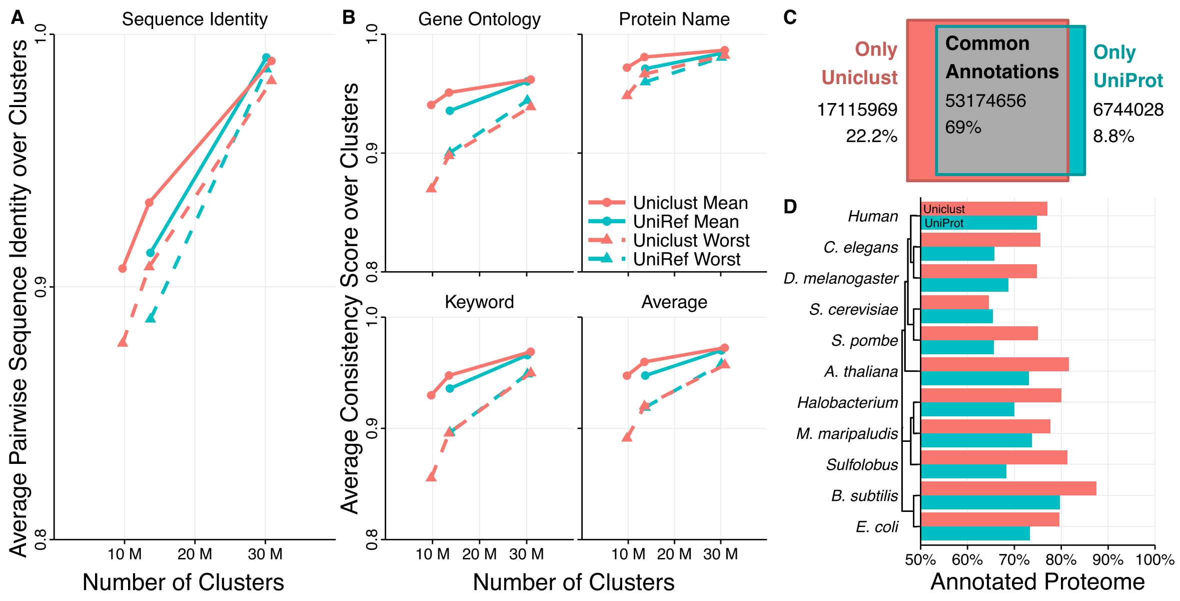


Figure 2. (**A**) Sequence identities averaged over all clusters of Uniclust30, Uniclust50, Uniclust90, UniRef50 and UniRef90. We compute the mean and worst sequence identity between all possible pairs of sequences in a cluster. If a cluster contains more than ten sequences we sample ten sequences to compute the sequence identities. (**B**) Annotation consistency scores averaged over all clusters of Uniclust30, Uniclust50, Uniclust90, UniRef50 and UniRef90. We compute the mean and worst annotation consistency between the representative sequence and all other cluster members for Gene Ontology annotations (top-left), protein names (top-right) keywords (bottom-left), and the average of the former three (bottom-right). (**C**) Total Pfam annotation count difference between Uniclust and UniProt. (**D**) Comparison of the fraction of proteins in ten model organisms with Pfam annotations in Uniclust and in UniProt.

annotation similarity between the reference sequence and the other cluster members. This analysis was performed on the Uniclust 2016_03 and UniRef 2016_03 releases based on the same version of UniProt with $N = 61\,522\,041$ sequences of 325 amino acids average length.

The y-axis in Figure 2B shows the consistency scores averaged over all clusters versus the number of clusters for Uniclust30, Uniclust50, Uniclust90 (red, left to right) and UniRef50, and Uniref90 (blue, left to right). Unsurprisingly, the lower the sequence identity threshold and the deeper the clustering, the fewer clusters are produced and the lower the annotation consistency scores get.

The mean scores of all annotation types show that the annotation consistencies of Uniclust90 and Uniclust50 clusters are markedly superior on average than to those of the corresponding UniRef databases.

The 'worst' annotation similarity per cluster is sensitive to the inclusion of even very few bad, functionally divergent sequences in the clusters. These 'worst' consistency scores are still quite high even for the Uniclust30, showing that the clustering produces highly pure clusters.

Note that an annotation similarity <1 between two sequences does not exclude the two sequences to have identical molecular functions but could simply be a consequence of one of the sequences being better annotated than the other. In this light, the cluster consistency scores are quite satisfactory. On the other hand, though, it is clear that many automatic annotations have been transferred on the basis of sequence similarity, which means that functional homogeneity might also be overestimated. However, such effects affect all clusterings in the same way and should therefore not invalidate the benchmark comparison. We further discuss in the supplementary material the evaluation using only GO EXP_F annotations, whose sparsity leads to a weak evaluation of the cluster consistencies.

Annotation depth

Figure 2C compares the number of annotations of Uniclust and UniProt. Uniclust sequences contain 70 290 625 Pfam annotations, whereas UniProt sequences are annotated with 59 918 684 Pfam domains. We analysed the overlap of Uniclust and Pfam annotations by counting how many of the overlapping Uniclust and UniProt Pfam domain annotations belonged to the same Pfam family clan. On a clan level Uniclust and UniProt share 53 174 656 common annotations, while Uniclust contains 17 115 969 sequence annotations not shared by UniProt, and UniProt sequences have 6 744 028 annotations not present in Uniclust sequences.

This greater annotation depth of Uniclust is reflected in the fraction of genes with at least one Pfam domain annotation in the proteomes of various model organisms (Figure 2D). For every model organism except for *Saccharomyces cerevisiae*, Uniclust can annotate a higher percentage of the proteome.

Availability of data

In the following we use the generic form **uniclust##_yyyy_mm.tar.gz** as placeholders for files such as **uniclust30_2016_03.tar.gz**. All downloads are available under a Creative Commons Attribution-ShareAlike 4.0 International license. We provide the following gzipped tar files for download:

- **uniclust##_yyyy_mm.tar.gz**: This archive contains three files, which will be updated every two months:
  - **uniclust##_yyyy_mm_seed.fasta**: representative (=seed) sequences of every cluster in FASTA format
  - **uniclust##_yyyy_mm_consensus.fasta**: consensus sequences of every cluster in FASTA. The sequence header starts with the Uniclust cluster identifier uc##-yymm-⟨*number*⟩, the UniProt accession code of the representative sequence, the size of the cluster, the up to five best functional annotations from cluster members, and UniProt identifiers of all cluster members.
  - **uniclust##_yyyy_mm_cluster_mapping.tsv**: tab-separated list with two columns of UniProt accession codes, the first for the representative sequence of the cluster, and the second for the member sequence.
- **uniboost##_yyyy_mm.tar.gz**: Uniboost database files in compressed A3M alignment format, with additional support files for HH-suite version 3.
- **uniclust30_yyyy_mm_hhsuite.tar.gz**: archive containing Uniclust multiple sequence alignments for all clusters in a3m format, generated with Clustal Omega (24), and additional support files for use with legacy HH-suite version 2 and current version 3.
- **uniclust_yyyy_mm_annotation.tar.gz**: archive containing three files with Pfam, SCOP, and PDB annotations, each formatted as tab-separated lists with nine columns: (1,2) identifiers for query and target, (3-5, 6-8) domain start and end-position and total sequence length for both UniProt and database sequence, (9) HHblits *E*-value.

CONCLUSION AND OUTLOOK

The Uniclust databases provide functionally homogeneous clusters of sequences at three clustering depths (90%, 50% and 30% sequence identity), sets of representative sequences, MSAs of clusters, and annotations of all sequences with Pfam, SCOP, and PDB matches. The Uniclust and Uniboost MSAs are also offered as databases for HHblits, the most sensitive method for remote protein homology detection, and the provision of regular updates to these databases resolves a sore deficiency of HHblits, which was limited by very irregular and rare database updates. The MSAs in Uniboost might also prove to be a useful resource for (deep) machine-learning applications, which benefit from training on massive amounts of labeled and annotated sequence profiles.

The clustering with our MMseqs2 software currently takes around five days on 10×16 cores, which is sustainable for the next five to ten years due to the near-ideal scalability of MMseqs2. (The Söding lab's cluster has 640 cores at this time.) But we are also actively developing both MMseqs2 and HHblits to achieve even higher speeds and sensitivities. We expect considerable improvements in the near future in the sensitivity with which we will detect and annotate structural domains in Uniclust/UniProtKB sequences using HHblits. Similarly, extending MMseqs2 to profile-profile searches will improve the sensitivity for building the Uniboost MSAs, which again will impact the sensitivity of the domain annotations.

The Uniclust server facilitates profiting from the Uniclust databases and deep HHblits domain annotations. We hope that they will become a widely used resource for protein sequence analysis.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

We are grateful to Markus Meier (Göttingen) for development of the Uniboost alignment database and to Borisas Bursteinas (EBI) for fruitful discussions. LvdD thanks the UniProt development team at the EBI for their hospitality.

FUNDING

European Research Council in the framework of its Horizon 2020 Framework Programme for Research and Innovation ['Virus-X', project no. 685778]; German Federal Ministry of Education and Research (BMBF) within the frameworks of e:Med and e:Bio [e:AtheroSysMed 01ZX1313D, SysCore 0316176A]. Funding for open access charge: Max Planck Society.

Conflict of interest statement. None declared.

REFERENCES

1. The UniProt Consortium. (2015) UniProt: a hub for protein information. *Nucleic Acids Res.*, **43**, D204–D212.
2. Benson,D.A., Cavanaugh,M., Clark,K., Karsch-Mizrachi,I., Lipman,D.J., Ostell,J. and Sayers,E.W. (2013) GenBank. *Nucleic Acids Res.*, **41**, D36–D42.
3. Koepfli,K.-P., Paten,B., O'Brien,S.J. and The Genome 10K Community of Scientists. (2015) The genome 10K project: A way forward. *Annu. Rev. Anim. Biosci.*, **3**, 57–111.
4. Zhang,G., Li,C., Li,Q., Li,B., Larkin,D.M., Lee,C., Storz,J.F., Antunes,A., Greenwold,M.J., Meredith,R.W. *et al.* (2014) Comparative genomics reveals insights into avian genome evolution and adaptation. *Science*, **346**, 1311–1320.
5. Grigoriev,I.V., Nikitin,R., Haridas,S., Kuo,A., Ohm,R., Otillar,R., Riley,R., Salamov,A., Zhao,X., Korzeniewski,F. *et al.* (2014) MycoCosm portal: gearing up for 1000 fungal genomes. *Nucleic Acids Res.*, **42**, D699–D704.
6. Park,J., Holm,L., Heger,A. and Chothia,C. (2000) RSDB: representative protein sequence databases have high information content. *Bioinformatics*, **16**, 458–464.
7. Li,W., Jaroszewski,L. and Godzik,A. (2002) Sequence clustering strategies improve remote homology recognitions while reducing search times. *Protein Eng.*, **15**, 643–649.
8. Chubb,D., Jefferys,B.R., Sternberg,M.J.E. and Kelley,L.A. (2010) Sequencing delivers diminishing returns for homology detection: Implications for mapping the protein universe. *Bioinformatics*, **26**, 2664–2671.
9. Suzek,B.E., Wang,Y., Huang,H., McGarvey,P.B. and Wu,C.H. (2015) UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, **31**, 926–932.
10. Li,W. and Godzik,A. (2006) Cd-hit: A fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**, 1658–1659.
11. Finn,R.D., Coggill,P., Eberhardt,R.Y., Eddy,S.R., Mistry,J., Mitchell,A.L., Potter,S.C., Punta,M., Qureshi,M., Sangrador-Vegas,A. *et al.* (2015) The Pfam protein families database: towards a more sustainable future. *Nucleic Acids Res.*, **44**, D279–D285.

12. Hubbard,T. J.P., Ailey,B., Brenner,S.E., Murzin,A.G. and Chothia,C. (1999) SCOP: A structural classification of proteins database. *Nucleic Acids Res.*, **27**, 254–256.
13. Berman,H., Henrick,K. and Nakamura,H. (2003) Announcing the worldwide Protein Data Bank. *Nat. Struct. Biol.*, **10**, 980.
14. Hauser,M., Steinegger,M. and Söding,J. (2016) MMseqs software suite for fast and deep clustering and searching of large protein sequence sets. *Bioinformatics*, **32**, 1323–1330.
15. Sjölander,K., Datta,R.S., Shen,Y. and Shoffner,G.M. (2011) Ortholog identification in the presence of domain architecture rearrangement. *Brief. Bioinform.*, **12**, 413–422.
16. Remmert,M., Biegert,A., Hauser,A. and Söding,J. (2012) HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nat. Methods*, **9**, 173–175.
17. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
18. Bostock,M., Ogievetsky,V. and Heer,J. (2011) D3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, **17**, 2301–2309.
19. Yachdav,G., Wilzbach,S., Rauscher,B., Sheridan,R., Sillitoe,I., Procter,J., Lewis,S., Rost,B. and Goldberg,T. (2016) MSAViewer: interactive JavaScript visualization of multiple sequence alignments. *Bioinformatics*, doi:10.1093/bioinformatics/btw474.
20. Huerta-Cepas,J., Serra,F. and Bork,P. (2016) ETE 3: reconstruction, analysis and visualization of phylogenomic data. *Mol. Biol. Evol.*, **33**, 1635–1638.
21. Gene Ontology Consortium. (2015) Gene Ontology Consortium: going forward. *Nucleic Acids Res.*, **43**, D1049–D1056.
22. Lin,D. (1998) An information-theoretic definition of similarity. In: *Proc. Fifteenth Int. Conf. Mach. Learn*. pp. 296–304.
23. Azuaje,F., Wang,H. and Bodenreider,O. (2005) Ontology-driven similarity approaches to supporting gene functional assessment. In: *Proc. ISMB'2005 SIG Meet. Bio-ontologies*. pp. 9–10.
24. Sievers,F., Wilm,A., Dineen,D., Gibson,T.J., Karplus,K., Li,W., Lopez,R., McWilliam,H., Remmert,M., Söding,J. *et al.* (2011) Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Mol. Syst. Biol.*, **7**, 539.

Supplemental: Uniclust - clustered and deeply annotated protein sequence databases

Milot Mirdita, Lars von den Driesch, Clovis Galiez,
Maria J. Martin, Johannes Söding, and Martin Steinegger

October 15th, 2016

1 Evaluation with experimentally validated annotations

As discussed in the main text, the validation may suffer from circularity, as UniProt annotations are dominantly transferred on the basis of sequence similarity. Thereby wrong homology based annotations can overestimate cluster consistency. The evaluation scores we produce can not and should not be interpreted as an absolute value of the quality of clusters, as annotations can suffer from errors and incompleteness. However the consistency scores can be used to compare different clustering algorithms or parameters of an algorithm. In our case it showed a higher consistency of Uniclust then UniRef.

Furthermore GO term associated to Uniprot entries are divided in three groups of evidence codes: experimental evidence codes and computational analysis evidence codes, which are both assigned by curators (denoted EXP_F in the sequel, with _F consisting only of functional GO annotations), and the automatically assigned evidence codes inferred from electronic annotation (IEA). In the main paper, we performed the evaluation with all evidence codes together, as the EXP_F coverage is very low and if we discard the annotations transferred by homology, there are significant disparities in the annotations depending on the type of experiment that has been carried out. As an example, Q5F3B5 and P50148 have 98.6% of sequence identity but they share a GO score of 0.0.

For sake of completeness, we evaluated the clusters of Uniclust and UniRef using only the EXP_F annotations. To cope with the disparity of the EXP_F annotations we use a procedure slightly different from the evaluation on all proteins. Specifically, to avoid strong bias depending on the choice of the representative sequence of a cluster, we average in each cluster all against all GO score comparisons of the protein annotations falling into the EXP_F category (this also includes self-scores as the clusters have very few proteins with an EXP_F annotation, in order to be consistent with the extreme situation where one has only one EXP_F protein per cluster).

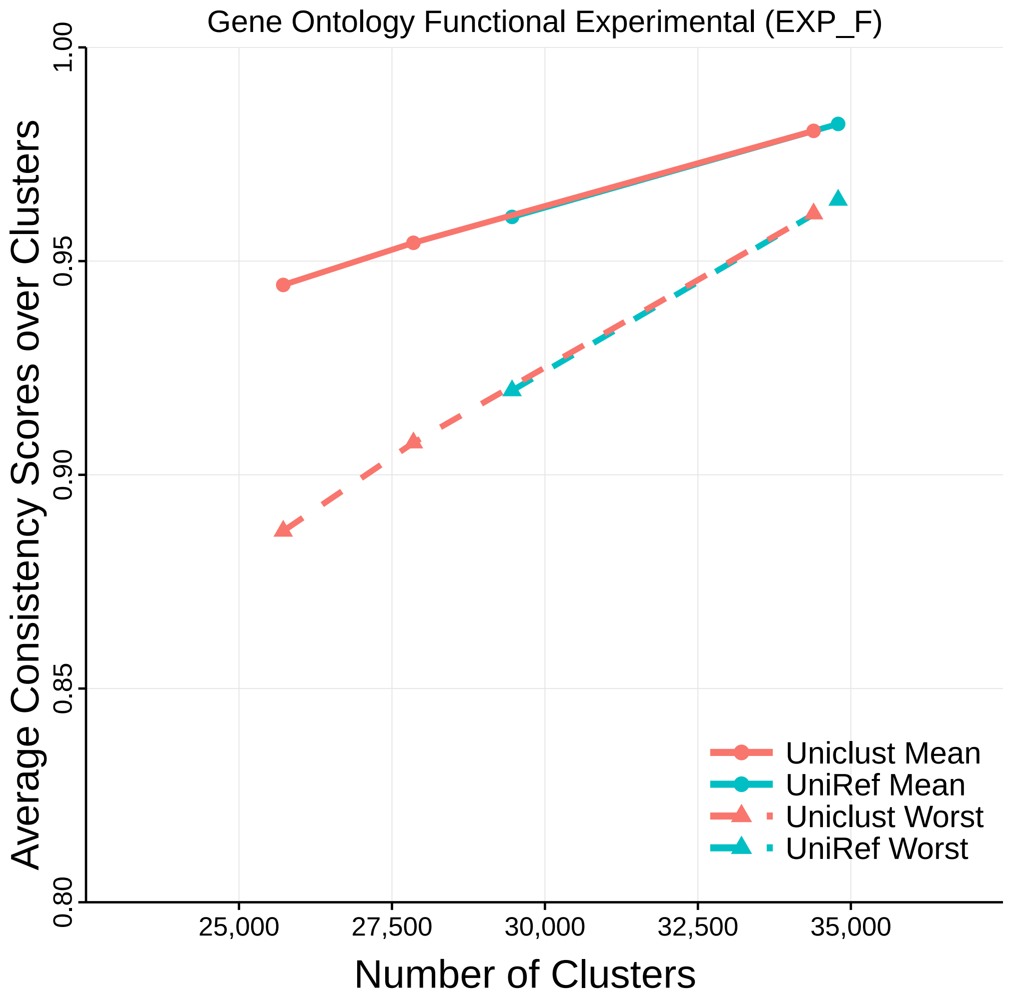


Figure 1: Evaluation with experimentally validated annotations

Note that the validation with only experimental annotation suffers from a very small coverage of clusters: only 3867 clusters in Uniclust30 include a protein with a GO EXP_F annotation.

2

Chapter 6

HFSP: High speed homology-driven function annotation of proteins

6.1 Introduction

Homology-derived Functional Similarity of Proteins (HFSP) (Mahlich et al. 2018) is a functional annotation measurement based on sequence identity and alignment length. It builds upon the previously published HSSP (Rost 2002; Rost 1999; Sander and Schneider 1991) measure. HFSP can be used in two ways: (1) as a functional similarity measure, which can be used to define functional distance between two proteins, or (2) to transfer functional annotations based on a sensitivity threshold.

The study reveals an annotation bias towards well studied proteins. Even highly similar proteins, based on sequence identity, could reside on different Enzyme Commission (Bairoch 2000) (EC) levels. This indicates that the granularity of the EC annotation is not well balanced and that different evaluation measures are needed.

The performance was evaluated by two benchmark sets based on entries of the manually curated Swiss-Prot (Bairoch and Apweiler 1999) database, published between the year 2002 and 2017. The MMseqs2 (Steinegger and Söding 2017) profile sequence search was used to annotate enzymes. The new proposed measure gains a speedup of a factor of 40 over PSI-BLAST (Altschul et al. 1997), and yields an accuracy of 83% at the fourth EC classification level when selecting the highest scoring HFSP pair per protein.

HFSP is a fast and accurate method to measure functional distance between two proteins. It will enable large scale functional annotation analysis and help to narrow the search space of proteins to experimentally analyse.

Yannick Mahlich performed the research, Martin Steinegger adapted MMseqs2. Yannick Mahlich, Burkhard Rost and Yana Bromberg jointly designed the research and wrote the manuscript.

6.2 References

Altschul, S. F. et al. (1997). “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.” In: *Nucleic acids research* 25.17, pp. 3389–402.

Bairoch, A. (2000). “The ENZYME database in 2000.” In: *Nucleic acids research* 28.1, pp. 304–5.

Bairoch, A. and R. Apweiler (1999). “The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999”. In: *Nucleic Acids Res* 27.1, pp. 49–54.

Mahlich, Yannick et al. (2018). “HFSP: high speed homology-driven function annotation of proteins”. In: *Bioinformatics* 34.13, pp. i304–i312.

Rost, B. (1999). “Twilight zone of protein sequence alignments.” In: *Protein engineering* 12.2, pp. 85–94.

— (2002). “Enzyme Function Less Conserved than Anticipated”. In: *Journal of Molecular Biology* 318.2, pp. 595–608. DOI: 10.1016/S0022-2836(02)00016-5.

Sander, C. and R. Schneider (1991). “Database of homology-derived protein structures and the structural meaning of sequence alignment”. In: *Proteins: Structure, Function, and Genetics* 9.1, pp. 56–68. DOI: 10.1002/prot.340090107.

Steinegger, M. and J. Söding (2017). “MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets”. In: *Nature Biotechnology* 35, 1026 EP -. DOI: 10.1038/nbt.3988;https://www.nature.com/articles/nbt.3988#supplementary-information.

6.3 Journal article

6. HFSP*: High speed homology-driven function annotation of proteins*

Bioinformatics, YYYY, 0–0
doi: 10.1093/bioinformatics/xxxxx
Advance Access Publication Date: DD Month YYYY
Manuscript Category

Subject Section

HFSP: High speed homology-driven function annotation of proteins

Yannick Mahlich[1,2,3*], Martin Steinegger[2,4,5], Burkhard Rost[2,3,6,7,8] and Yana Bromberg[1,3,9*]

[1]Department of Biochemistry and Microbiology, Rutgers University, 76 Lipman Dr, New Brunswick, NJ 08873, USA, [2]Computational Biology & Bioinformatics - i12 Informatics, Technical University of Munich (TUM) Boltzmannstrasse 3 85748 Garching/Munich Germany, [3]Institute for Advanced Study, Technische Universität München, Lichtenbergstrasse 2 a, D-85748 Garching, Germany, [4]Quantitative and Computational Biology group, Max-Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077 Göttingen, Germany, [5]Department of Chemistry, Seoul National University, Seoul, Korea, [6]TUM School of Life Sciences Weihenstephan (WZW), Alte Akademie 8, Freising, Germany, [7]Columbia University, Department of Biochemistry and Molecular Biophysics, 701 West, 168th Street, New York, NY 10032, USA, [8]New York Consortium on Membrane Protein Structure (NYCOMPS), 701 West, 168th Street, New York, NY 10032, USA, [9]Department of Genetics, Rutgers University, Human Genetics Institute, Life Sciences Building, 145 Bevier Road, Piscataway, NJ 08854, USA

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: The rapid drop in sequencing costs has produced many more (predicted) protein sequences than can feasibly be functionally annotated with wet-lab experiments. Thus, many computational methods have been developed for this purpose. Most of these methods employ homology-based inference, approximated via sequence alignments, to transfer functional annotations between proteins. The increase in the number of available sequences, however, has drastically increased the search space, thus significantly slowing down alignment methods.

Results: Here we describe HFSP, a novel computational method that uses results of a high-speed alignment algorithm, MMseqs2, to infer functional similarity of proteins on the basis of their alignment length and sequence identity. We show that our method is accurate (83% accuracy) and fast (more than 40-fold speed increase over state-of-the-art). HFSP can help correct at least a 20% error in legacy curations, even for a resource of as high quality as Swiss-Prot. These findings suggest HFSP as an ideal resource for large-scale functional annotation efforts.

Contact: ymahlich@bromberglab.org, yanab@rci.rutgers.edu

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

1 Introduction

The recent rapid drop in the cost of DNA-sequencing has produced a large number of fully sequenced genomes. For prokaryotes, for example, this represents a more than six-fold growth (1,400 to 9,000 in Genbank (Benson, et al., 2013)) in the last 5 years alone. While this increase in data enables many types of research, experimental annotation lags far behind. In particular, the speed (or lack thereof) of experimental evaluation and validation of protein molecular functionality clearly necessitates computational approaches. In fact, many methods (Jiang, et al., 2016; Radivojac, et al., 2013) have already been developed for this purpose, the vast majority of which rely on transfer of functional annotation by homology (Loewenstein, et al., 2009). Mistakes in available annotations (Schnoes, et al., 2009), inconsistencies in experiments, as well as simply missing or yet unknown functions make these sequence similarity-based methods error-

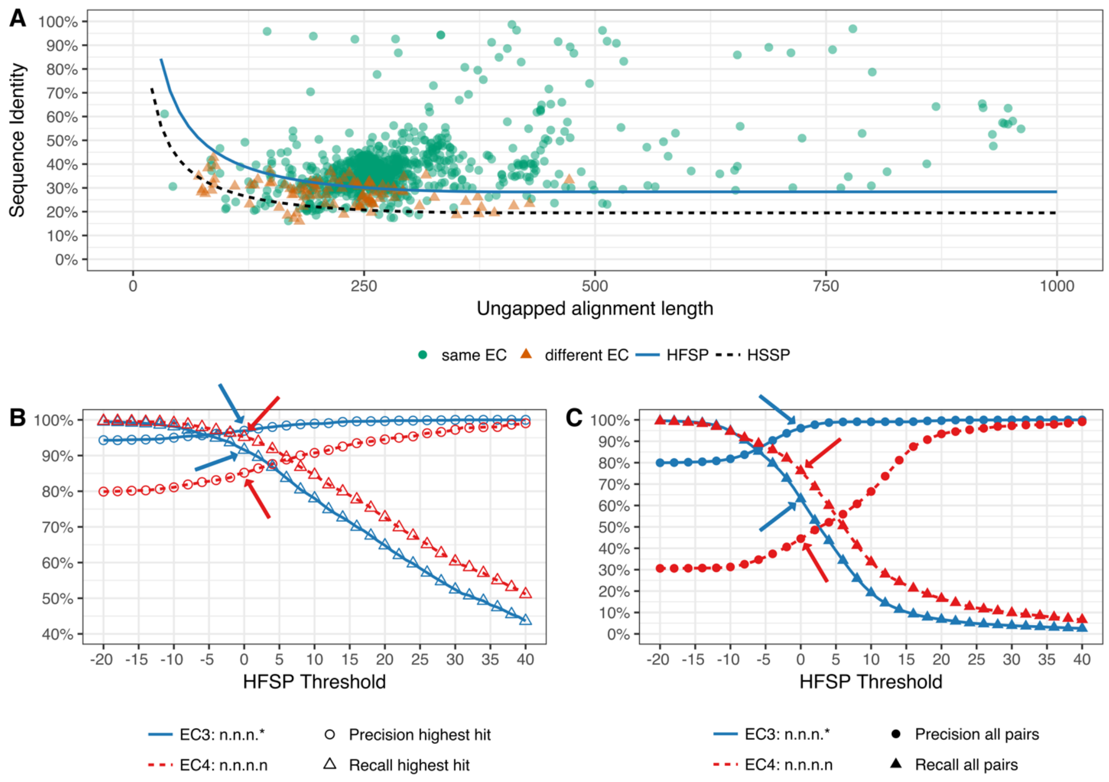


Fig. 1: HFSP precisely predicts functional identity. All Swiss-Prot 2002 protein pairwise alignments were mapped into the sequence identity vs. ungapped alignment length space. In (A) protein pairs were differentiated according to identity of their EC level 3 (same EC annotation are green circles; different annotations are red triangles). The HFSP curve (HFSP=0, light blue solid line) is shown relative to the HSSP curve (black dashed line). Protein pairs above the curve are predicted to be of same function, pairs below the curve of different function. In (B, C) precision (circles) and recall (triangles) in predicting functional identity, at 3rd (blue, solid curve) and 4th (red, dashed curve) EC level for Swiss-Prot 2002. Arrows indicate performance at default cutoff of HFSP = 0. In (B) prediction was done using the highest HFSP scoring alignment per protein. In (C) all alignments were used, resulting in significantly worse performance.

prone (Clark and Radivojac, 2011). Furthermore, organism-focused research interests result in more detailed annotations for a non-random subset of proteins, where homologous proteins of identical functionality in another species are often annotated significantly less thoroughly. Evaluating the performance of computational annotation methods is complicated by the absence of large, well curated, and 'evenly' functionally annotated protein sets, representing the entire breadth of available biomolecular functionality.

Protein sets that are used as benchmarks of prediction employ annotation ontologies, *i.e.* standardized terms and their relationships. One such benchmark set is enzymes with Enzyme Commission (Bairoch, 2000) (EC) numbers. EC numbers reflect a four level hierarchy, where each consecutive level is a more precise specification of the annotation on the previous level. For example, enzymes classified as EC:1.1.1.- are oxidoreductases (1st level), acting on the CH-OH group of electron donors (2nd level), with NAD^+ or $NADP^+$ as an electron acceptor (3rd level). The fourth and most specific level might then annotate an enzyme as alcohol dehydrogenase (EC:1.1.1.1), *i.e.* reducing the aldehyde group of the molecule. Note that dashes ("-") in EC numbers indicate lack of specificity of functional annotation at that level. While EC numbers facilitate comparison of functions across enzymes, the annotation specificity at the same EC level varies; *e.g.* the class of serine/threonine protein kinases (EC:2.7.11.-) contains a category EC:2.7.11.1 (4th level annotation = 1) that collects all kinases that are non-specific or whose specificity hasn't been analyzed to date. On the other hand, serine/threonine protein kinases with the 4th level annotations between 2 and 32 are very specifically annotated, with each category limited to proteins that act on a particular substrate. Using EC annotations as a benchmark, thus, comes at the expense of variability in annotations even at the same level of the hierarchy. This, in turn, complicates establishing functional similarity of two proteins in a precise and balanced manner across the entire enzymatic activity spectrum.

By definition, using EC annotations also means missing out on non-enzymatic functionality. Other ontologies, like the molecular function branch of Gene Ontology (Ashburner, et al., 2000) (GO) do not have this limitation. GO, however, employs a different, even more detailed, strategy in defining function than EC. The number of GO annotation levels varies by ontology sub-branch. Moreover, one protein can (and likely does) have multiple functional GO terms assigned to it (*e.g.* both copper ion binding

Article short title

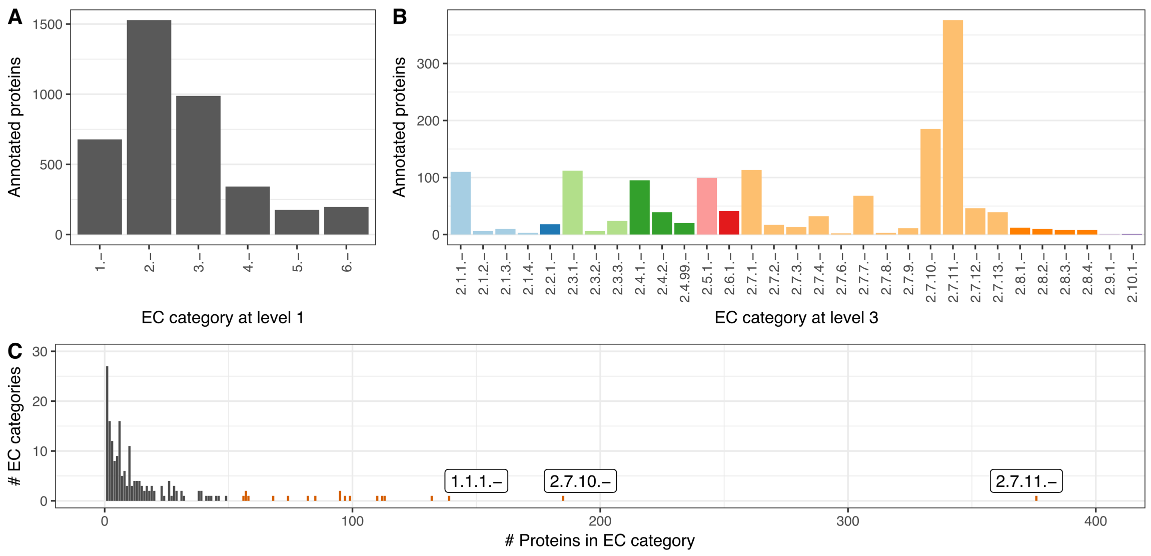


Fig. 2: Strong bias in EC distribution. Different EC categories contain different numbers of proteins with both general (A) EC level 1 and (B) more specific EC annotations. (C) This bias is particularly obvious for 3rd level EC categories, with 2.7.11.-, 2.7.10.- and 1.1.1.- being the most prominent (first three bars from right; all ECs with more than 50 proteins are red).

and DNA binding terms describe the function of P53; AmiGo 2.4.6; PMID:15358771, PMID:7824276). Thus, comparing GO annotations may lead to much stronger distortions of similarity than skewed or even incomplete EC numbers. Note that moonlighting (Khan, et al., 2014) proteins, *i.e.* proteins that can be assigned multiple specific functions, further confuse functional similarity metrics.

As a consequence of the drastic increase in genomic and protein sequences in need of annotation, the search space for all computational function assignment methods has also increased. A centerpiece of much of sequence analysis efforts is the BLAST (Altschul, et al., 1990; Altschul, et al., 1997) family. We note that with the quasi exponential growth in search space, while PSI-BLAST (Altschul, et al., 1997) may still remain viable for the analysis of a single protein, large scale evaluations are not time-feasible. Many methods that reduce runtime while retaining or increasing alignment accuracy have been developed over the last years, including caBLASTp (Daniels, et al., 2013), HHblits (Remmert, et al., 2011), and MMseqs2 (Steinegger and Soding, 2017). However, replacing (PSI-) BLAST in any bioinformatics pipeline with another alignment method requires parameter re-optimization or even a complete method overhaul.

Existing function prediction methods are very sophisticated, using a variety of inputs (*e.g.* structure and literature mining) and computational techniques (*e.g.* machine learning). However, here we focused on HSSP (Rost, 1999; Rost, 2002; Sander and Schneider, 1991) – a simple distance metric that infers protein function and structure similarity from sequence identity and alignment length. We optimized HSSP parameters to classify protein pairs as functionally identical or different using the results of MMseqs2, a lightning-fast alignment method. We found that our newly developed HFSP (Homology-derived Functional Similarity of Proteins) method is 40-fold faster than HSSP, while retaining HSSP precision in annotating enzymatic functionality of proteins (83% accuracy; Figure 1).

Analyzing existing protein databases with our method, we showed that currently available computationally determined annotations in even the manually curated Swiss-Prot (The UniProt, 2017) database are incorrect for at least a fifth of the cases. We suggest that these errors are likely due to loosely defined rules of homology-based propagation of functional annotations. With the number of protein sequences in public databases bordering on 100 million and growing, HFSP is well suited to help improve the quality of existing and newly assigned functional annotations.

2 Methods

Extraction of datasets. We extracted a set of reviewed proteins from Swiss-Prot with only one, EC (Bairoch, 2000) annotation per protein (complete at all four levels; 214,000 proteins; *Swiss-Prot set*). The 2002 (latest) formula for computing the HSSP (Rost, 1999; Rost, 2002) distances was developed on a combined set of Swiss-Prot (The UniProt, 2017) and PDB (Berman, et al., 2002) proteins. To validate the performance of HSSP reported in (Rost, 1999; Rost, 2002), we extracted proteins from the Swiss-Prot set that had experimental evidence of protein existence (*e.g.* crystal structure, protein detection by antibodies, etc.) and an EC annotation in BRENDA (Placzek, et al., 2017). The resulting proteins (*Swiss-Prot 2017* set; 7,022 proteins) were further filtered to retain entries appearing in the database before January 2002 (*Swiss-Prot 2002*, 3,908 proteins). Both *Swiss-Prot 2017* and *2002* datasets were extracted in October 2017 (Uniprot release 2017_09) and redundancy reduced to 98% sequence similarity and 98% target sequence coverage with CD-HIT (Fu, et al., 2012; Li and Godzik, 2006). *Swiss-Prot 2002* contained 3,801 proteins with 1,481 unique EC annotations and *Swiss-Prot 2017* containing 6,835 proteins with 2,552 unique EC annotations (SOM Data 1).

Swiss-Prot 2017 was further split into sets containing only prokaryotic (*Swiss-Prot$_{pro}$ 2017*, 2,572 proteins) or eukaryotic (*Swiss-Prot$_{euk}$ 2017,* 4,263 proteins) proteins. Finally, we extracted two more Swiss-Prot subsets from: (1) proteins that did not have an EC annotation (293,058 proteins) and (2) proteins with incomplete or multiple EC annotations (48,536 proteins).

Aligning proteins. To augment the homology profiles used in alignments (by both PSI-BLAST (Altschul, et al., 1997) and MMseqs2), we computed alignments of all proteins in our datasets (Swiss-Prot 2002, Swiss-Prot 2017, Swiss-Prot$_{pro}$ 2017 & Swiss-Prot$_{euk}$ 2017) against proteins in the full (non-reduced) Swiss-Prot (Uniprot release 2017_09). For each specific dataset, we then extracted only those alignments, where both proteins were present in that set (*e.g. both query and target protein in Swiss-Prot 2002*).

PSI-BLAST alignments where created with NCBI-BLAST version 2.2.29+. We ran three iterations of PSI-BLAST (-num_iterations 3). In each iteration, top 500 hits (E-value 10^{-10}, -inclusion_ethresh 1e-10) were included into the profile. After the third round all alignments that satisfied the E-value ≤ 10^{-3} threshold (-evalue 1e-3) were considered for evaluation of performance.

MMseqs2 (Steinegger and Soding, 2017) parameters were chosen to mirror the PSI-BLAST runs. The alignment-mode (--alignment-mode 3) was set to calculate sequence identity between query and target was over the full alignment length, i.e. analogous to BLAST. We ran three iterations (--num-iterations 3) of alignments including hits with an E-value ≤ 10^{-10} into the generated profile (--e-profile 1e-10). Only alignments of protein pairs with and E-value ≤ 10^{-3} were reported in the final result (-e 1e-3). The sensitivity (-s) cut-off for MMseqs2 prefiltering step was set to 5.6 (default value).

It had taken MMseqs2 1,228 CPU hours to complete the alignment of our Swiss-Prot enzyme set (214,000 proteins) to the full (non-reduced) Swiss-Prot (555,594 proteins). Although MMSeqs2 was exceedingly fast for this set, note that it has been optimized to deal with much larger databases and, thus, it did not reach its full potential in speed. In earlier testing (Zhu, et al., 2015; Zhu, et al., 2018) with a dataset of ~4.2 million proteins, the all-to-all protein alignment time for the MMseqs2 was ~30 thousand CPU hours (4.2e6 x 4.2e6 =~ 1.8e13 comparisons in roughly 4 days on 12 compute nodes with 24 CPUs each). In comparison, creating the same PSI-BLAST alignments took ~1.3 million CPU hours (~3 months on 78 compute nodes with 8 CPUs each). From these numbers, the HFSP speed-up (using MMseqs2) over HSSP (using PSI-BLAST) was estimated at over 40-fold and expected to grow significantly with database size.

Defining functional identity. Proteins sharing the same EC annotation at chosen (3rd or 4th level) were assigned functional identity. For example, L-lactate dehydrogenase and D-lactate dehydrogenase have EC assignments 1.1.1.27 and 1.1.1.28 respectively. Thus, at EC level 4, the proteins are different, but at EC level 3 they are the same, 1.1.1.

Retraining HSSP curve with MMseqs2. We used the *Swiss-Prot 2002* proteins and their 3rd EC level annotations to develop the HFSP measure. Investigating the protein distribution of EC categories at the 3rd EC level, we realized a strong distortion towards a few EC categories with exceptionally many associated proteins (Figure 2C). This is in addition to other differences between EC categories (Figure 2A,B). To compensate for this category bias, we limited the size of EC categories to no more than 50 proteins (randomly chosen for the 19 larger categories, SOM Table 1). We then extracted all MMseqs2 alignments for all *Swiss-Prot 2002* protein pairs in our set.

It has been previously shown that using class-balanced training sets is beneficial in the development of data driven classification models (Rost and Sander, 1993; Wei and Dunbrack, 2013). We therefore balanced the results in training to contain equal numbers of protein pairs with the same *vs.* different 3rd level EC annotations.

We first used cross-validation for training/testing our method; *i.e.* we split the data into ten sets such that no sequence in one set shared more than 40% identity with a sequence in another set (CD-HIT clusters). In each of in ten rounds of training, one set was retained for testing and the other nine were used for training. Note that in each round of cross-validation, we reintroduced into the testing set those proteins, which were originally removed for class balancing purposes. We optimized the parameters (originally factor=480 and exponent=-0.32; Eqn. 1, SOM Table 2) of the 2002 HSSP formula (Rost, 2002) to fit a new curve separating protein pairs of identical function from those of different functions in the two-dimensional space of sequence identity (y-axis) and ungapped alignment length (alignment length – number of gaps; x-axis). Pairs of same function proteins (identical annotation for EC) and a given threshold distance away from the curve along the y-axis were true positives (TP). Pairs that did not have the same function but were also above the threshold were false positives (FP). False negatives (FN) were pairs of same function but scoring below the threshold. We optimized for F_1 score (Eqn. 3) using R's implementation of the Nelder-Mead method (Nelder and Mead, 1965), searching for a local optimal F_1 score, using combinations of exponents from -0.3 to -0.9 in steps of 0.05, and factor from 300 to 1,500 in steps of 50.

$$HSSP = PIDE - \begin{cases} 100, & for\ L \leq 11 \\ 480 \cdot L^{-0.32 \cdot \left(1+e^{-\frac{L}{1000}}\right)}, & for\ 11 < L \leq 450 \\ 19.5, & for\ L > 450 \end{cases} \quad (1)$$

$PIDE = Percent\ sequence\ identity\ of\ the\ alignment$

$L = ungapped\ alignment\ length$

$$precision = \frac{TP}{TP+FP}, recall = \frac{TP}{TP+FN} \quad (2)$$

$$F_1\ score = 2 \cdot \frac{precision \cdot recall}{precision+recall} \quad (3)$$

HFSP values for protein pairs were calculated using MMseqs2 results; Pearson correlation coefficient of HFSP to the HSSP values computed using PSI-BLAST results for same pairs. For each dataset, we calculated precision (*i.e.* how often a prediction of identical function is correct), recall (*i.e.* how many identical function pairs were correctly identified), and the F_1 score (Eqn. 2,3) using HSSP and HFSP distance thresholds to determine true/false positives/negatives.

After evaluation was completed, we retrained as described above, but without testing, one HFSP curve on the complete balanced set of *Swiss-Prot 2002* protein pairs for all further use.

Using HFSP to make function predictions. We used the 6,835 experimentally annotated proteins with 2,552 unique EC annotations of *Swiss-Prot 2017* as the reference database for all further function predictions. For every protein, only the highest HFSP-scoring protein match (≥0; excluding self-matches) was used to annotate function. We thus predicted functions of proteins in the complete Swiss-Prot set of enzymes. Curiously, some EC numbers used in Swiss-Prot protein annotation did not have any members in the experimentally annotated *Swiss-Prot* 2017 reference set. The proteins annotated with these EC numbers (32,201 proteins at 4th and 381 proteins at 3rd EC level, respectively) were considered false positives by default. Note that we are still unclear about the origins and experimental support of these annotations. Additionally, some proteins did not produce any alignments, and for others the highest hits did not reach our HFSP cutoff=0. For these, no functional assignment could be made.

3 Results

3.1 HFSP scores correlate with HSSP, but are produced more than 40-fold faster

We trained, evaluated, and defined the HFSP (Homology derived functional similarity of proteins; Eqn. 4) as described in Methods.

$$HFSP = PIDE - \begin{cases} 100, & for\ L \le 11 \\ 770 \cdot L^{-0.33 \cdot \left(1+e^{-\frac{L}{1000}}\right)}, & for\ 11 < L \le 450 \\ 28.4, & for\ L > 450 \end{cases} \quad (4)$$

HFSP uses MMseqs2 iterative profiles as they have three major advantages over PSI-BLAST: (1) compositional bias correction to suppress high scoring non-homologous alignments, (2) profile computation by only considering the 1000 most diverse sequences (PSI-BLAST uses the *n* BEST scoring hits) (3) and realignment to reduce over-extension (Frith, et al., 2008); over-extension includes sequences into the profile at the edges of the alignment threshold in consecutive iterations. Thus, MMseqs2 alignments of smaller and more distant proteins tend to be more compact, favoring higher sequence identity, and thus leading to slightly higher HSSP scores calculated using the original equation (Eqn. 1). These differences in alignment methods, however, do not significantly affect the HSSP scores across the entire spectrum, especially for high sequence identity alignments (Pearson correlation coefficient between BLAST-based and MMseqs2-based HSSP scores =0.95; Figure 3).

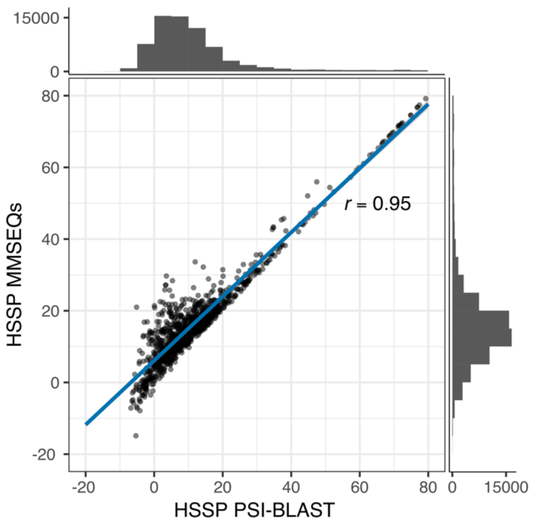


Fig. 3: HSSP scores derived from MMSeqs2 and PSI-BLAST alignments strongly correlate. HSSP scores derived from PSI-BLAST alignments (x-axis) and MMSeqs2 (y-axis) respectively. The histograms display the number of protein pairs in the respective ranges of HSSP scores. HSSP scores for both methods highly correlate (Pearson-correlation coefficient = 0.95).

3.2 HFSP precisely identifies the 3rd, but not 4th, level of EC annotations

In identifying pairs of proteins sharing the same function at the 4th level of EC (Methods), HFSP attained precision of 44.1% ± 3.6 at HFSP 0 and recall of 71.5% ± 1.6 (in cross-validation). This disappointing performance suggests that the increasing resolution/fine-tuning of experimental molecular function annotation is prohibitive for large-scale computational analyses of proteins; *i.e.* for any given alignment scoring HFSP≥0, it is more likely that the proteins in the alignment are not functionally identical.

In exploring this problem, we found that many highly sequence similar protein pairs of different EC annotations contained homologous proteins that were assigned slightly different functionality in different organisms. For example proteins from the squalene cyclase family (Interpro: IPR018333, Pfam: PF13243 & PF13249) were annotated with different ECs; *e.g.* GERS_RHISY, a germanicol synthase in the red mangrove, is assigned EC:5.4.99.34 and has 93% sequence identity (alignment length=758) to BAS_BRUGY, a Beta-amyrin synthase of the burma mangrove, which is annotated as EC:5.4.99.39. This combination of sequence identity and alignment length produces an HFSP score of 64.6. At this HFSP level protein pairs are predicted to share the same EC annotation at 4th EC level with a precision of >99%. Note that GERS_RHISY is the only EC 5.4.99.34 protein to date. The publication describing its catalytic activity (Basyuni, et al., 2007), suggests that GERS_RHISY activity warrants a brand new EC number (germanicol synthase), because it primarily catalyzes germanicol synthesis. From our perspective, GERS_RHISY should additionally carry the beta-amyrin synthase annotation, since beta-amyrin (and lupeol) are synthesized in addition to germanicol albeit at a lower rate. Note that this example also recalls the problem of moonlighting proteins.

The above example reflects the general problem of unbalanced annotation detail of different EC categories at the same level of annotation. For example, EC:5.4.99.- is by choice of the Enzyme Commission meant to “house” temporarily a collection of enzyme reactions that have yet to be more thoroughly categorized. Many members of EC:5.4.99.- fall into the same PFAM families, while catalyzing the conversion of the same reactant into similar chemical compounds; *i.e.* the 4th level EC annotations of these proteins convey only a small amount of functional difference. However, 5.4.99.- also contains significantly different proteins catalyzing different reactions, where fourth level annotations convey very large differences. Note that in this scheme, automated protein function annotation is significantly limited by lack of awareness of what individual EC numbers represent; *i.e.* it is incorrect to assume that the fourth, most precise, level EC annotations, across the entire EC system, are similarly defined in terms of depth of functional understanding and/or functional distances between proteins of the same third level EC. Note, however, that increasing the HFSP threshold for calling protein functions identical leads to significantly improved precision (if at significant cost to recall). For example, at HFSP cutoff = 20, 93% of the protein pairs are correctly annotated to share functionality. In other words, protein pairs with higher HFSP score represent more reliable predictions. This improvement is unsurprising as it is due in large part to increasing sequence identity and is very likely reflective of closer evolutionary relationships between proteins.

In identifying pairs of proteins sharing the same function at the 3rd level of EC, we found that performance improved drastically at the default HFSP cutoff = 0. Here, our method attained precision of 96% ± 1.2 at HFSP 0 and recall of 64% ± 1.6 (in cross-validation, Figure 1). These results suggest that in the absence of additional knowledge about an aligned protein pair, it is prudent to only accept higher scoring HFSP alignments (for 4th digit annotations) or to move up in the required resolution of functional annotation (i.e. to 3rd EC level).

Finally, we tested HFSP precision and recall on proteins in *Swiss-Prot 2017* that were NOT in *Swiss-Prot 2002* (which was used for training of the HFSP curve), *i.e.* proteins that were added to Swiss-Prot after January

2002. We found that performance for this subset was similar to the expected performance at both the 3rd and 4th EC levels (Figure 4), suggesting that our measure remains applicable for newly added proteins AND enzyme classes (EC numbers).

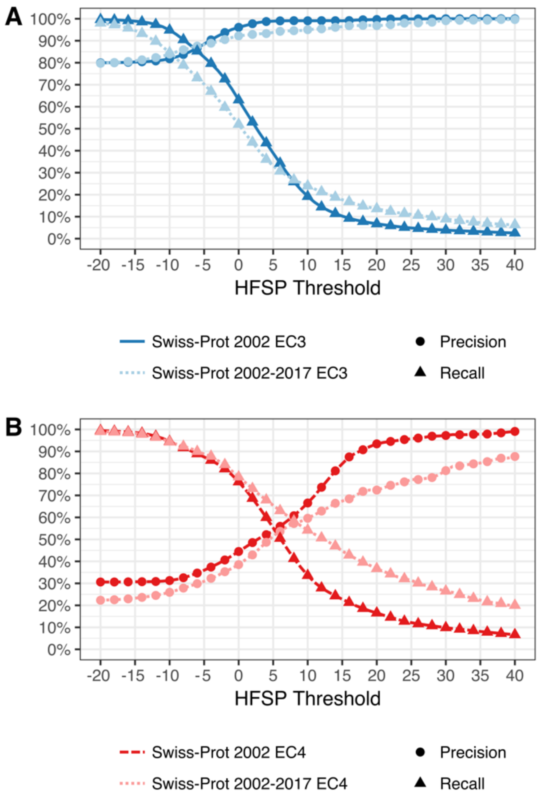


Fig. 4: HFSP performs as expected on newly added proteins. Precision and recall of function prediction at (A) 3rd (dark blue) EC level of proteins in Swiss-Prot 2002 and of those added since 2002 (Swiss-Prot 2002-2017; light blue) are similar. However, for the 4th EC level, the (B) performance on newly added proteins (dark red) is worse than for older ones (light red).

3.3 HFSP performance differs in annotating prokaryotic vs. eukaryotic proteins

We additionally evaluated the HFSP performance in annotating the eukaryotic vs. prokaryotic proteins of the entire *Swiss-Prot 2017 set* (Methods, Figure 5A) at the 3rd EC level. At our default cutoff of HFSP = 0, eukaryotic protein pairs were assigned functional similarity correctly more often than prokaryotic ones (precision/recall 96%± 1.5%/62% vs. 91%±1.5/47%, respectively). Note that there were more eukaryotic proteins in our data than prokaryotic ones, which may have contributed to this disparity during HFSP curve optimization. This larger number of proteins can be explained by the eukaryotes (1) trending towards bigger proteomes and, perhaps more importantly, (2) making up a bigger fraction of model organisms, which are better studied. Curiously, at the 4th EC level this trend was reversed, *i.e.* precision was better for prokaryotes than for eukaryotes (precision/recall 62%/55% vs. 42%/79%, respectively, Figure 5B). This observation may potentially be due to a smaller number of homology-confusing multi-domain proteins in prokaryotes. It may also reflect a lower enzymatic diversity of prokaryotic proteins in our set: 1,522 distinct EC annotations in eukaryotes *vs*. 1,403 in prokaryotes. Whether this difference is due to actual diversity or a result of experimental bias remains unclear.

3.4 HFSP accurately predicts unknown protein function at all EC levels

There is a conceptual difference between annotating functionality of an unknown protein and measuring functional similarity of two proteins. That is, in assigning ONE specific protein function to a newly obtained amino acid sequence is not the same as relying on homology to identify proteins sharing the similar functionality in a particular database. To use HFSP as a method of function prediction we proposed simply relying on the “highest hit”; we have previously shown that this approach is best for transferring functional annotations with HSSP (Zhu, et al., 2018) and suggest that similar logic should apply here.

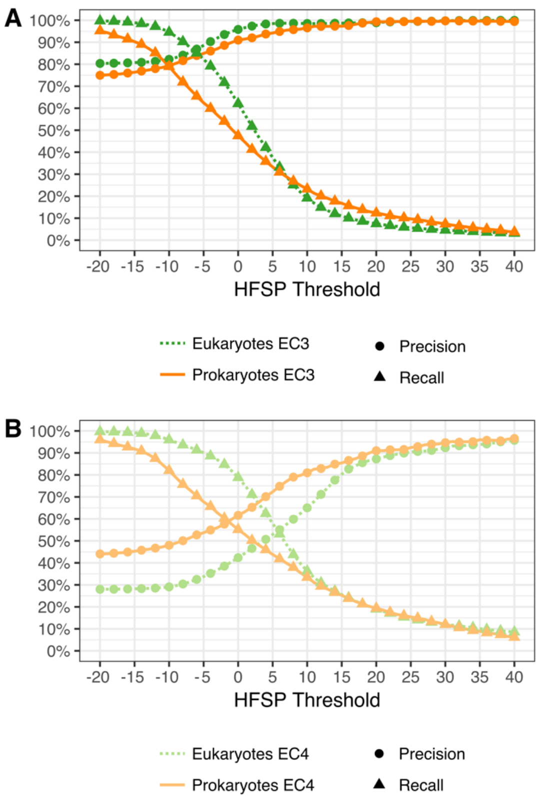


Fig. 5: Differing annotation performance for prokaryotic and eukaryotic proteins at 3rd and 4th EC level. (A) For the 3rd EC level at default cutoff of HFSP = 0, eukaryotic protein pairs are assigned functional similarity correctly more often than prokaryotic ones. However, for high thresholds, *i.e.* higher precision at the expense of recovered protein pairs, performance is similar. **(B)** Performance is better for prokaryotes than eukaryotes at the 4th EC level.

By mapping the highest HFSP match (at cutoff = 0 and excluding self-hits) for the experimentally annotated proteins of the *Swiss-Prot 2017* set, we were able to correctly identify the 4th level EC function of 4,668 (~83% of 5,647) proteins. As expected, the numbers were higher for the 3rd level EC (5,425 of 5,647 proteins, 96%). Note that this performance is the upper limit of actual HFSP performance, as *Swiss-Prot 2002*, on which our method was developed, is a subset of *Swiss-Prot 2017*. Also note that (1) 625 proteins in our *Swiss-Prot 2017* set did not reach our HFSP cutoff=0 and (2) 563 proteins did not align to any others in our set. Of these, 645 proteins (291 and 354, respectively) proteins were unique in our set; *i.e.* there was no other protein with the same EC number at 4th EC level. Thus, 1,188 proteins in our set (~17% of 6,835 in the set) could not be assigned function at all – ~8% due to HFSP limitations and ~9% due to the absence of homologs.

3.5 Functional annotations even in manually curated databases are often incorrect

We applied the highest HFSP hit measure to evaluate EC annotations in the entire Swiss-Prot set (Methods) on the basis of their alignment to our experimentally annotated *Swiss-prot 2017* set. We estimate that 142,831 of the 214,000 Swiss-Prot enzymes (67%) are correctly annotated at the 4th level of EC (Figure 6). Curiously, 32,201 (15%) of the enzymes in Swiss-Prot had no corresponding 4th level ECs (381 3rd level ECs) in *Swiss-Prot 2017,* raising questions as to the accuracy of these annotations. Another 4,937 are deemed wrongly annotated (highest hit at HFSP≥0 has a different EC number). While these proteins may indeed be assigned wrong functionality, this may also be due to error in HFSP assignments at this level (17% false positives at this cutoff, as described above for the *Swiss-Prot 2017* experimentally-annotated set). A more interesting finding, however, is that 34,031 (19%) of the proteins in this set could not be annotated at all by HFSP, whether due to lack of alignments (17,519 proteins) or HFSP highest hits unable to reach the cutoff (16,512 proteins). These 19% of proteins that could not be annotated represent a more than two-fold higher number than expected (~8% as described above for the *Swiss-Prot 2017* set). We, thus, suggest that the Swiss-Prot EC annotations of many of these 34,031 proteins, a sixth of the total number of annotations, are incorrect.

3.6 Identifying proteins of new functionality is simplified with HFSP

One problem of function transfer by homology methods is their inability to identify proteins of completely novel, *i.e.* not found in the reference database, functionality. Note that sequence similar proteins are also likely functionally *similar*, but are clearly not necessarily functionally *identical*. To evaluate how HFSP deals with proteins of novel functionality, we extracted a set of proteins from *Swiss-Prot 2017*, where no other protein in our set had the same 4th EC level annotation ('unknown' functionality). These 'unknown' proteins, *i.e.* assigned to a 4th EC level category, which appear just once in our set, are a minority (19%; 1,317 of 6,835 proteins), albeit a significant one. We asked if we could in advance identify these 'unknown' proteins, for which prediction of function could not be made, rather than making incorrect predictions.

We separated function predictions for the 6,835 proteins in Swiss-Prot 2017 into three subsets (1) 'unknown', as described above, and (2) correctly and (3) incorrectly predicted 'known', *i.e.* proteins with 4th EC level annotations containing more than one protein. We then compared the highest hit HFSP score distributions for all three sets (Figure 7). HFSP

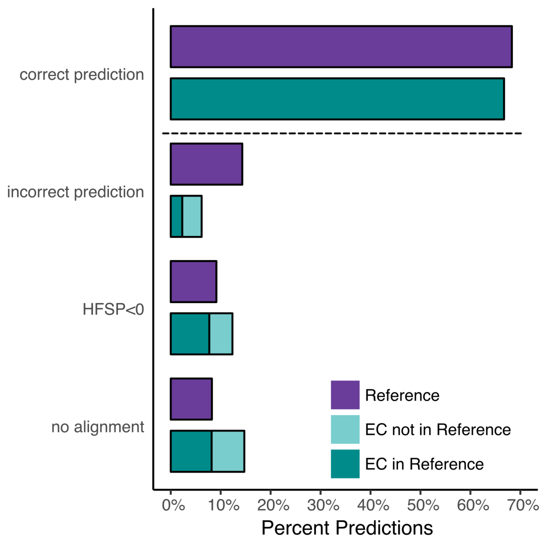


Fig. 6: More proteins in Swiss-Prot enzyme could not be assigned to function than expected. Function predictions for proteins in Swiss-Prot 2017 (Reference, purple) and all proteins in Swiss-Prot with EC annotation complete on all four levels that either share an EC with proteins in Reference (light teal) or not (dark teal).

scores for correctly annotated proteins with known functionality appear to come from a mixture of two distributions. These are likely to be evolutionarily distant (peak of the distribution at HFSP = ~20) *vs.* close (peak at HFSP = ~65) homologs. The peak of the distribution of 'unknown' protein scores is obviously different from either (HFSP = ~2). However, the distribution of incorrect predictions for 'known' proteins closely follows the 'unknowns' (Fig. 7A and SOM Fig. 2A, B). Combined, 'known incorrect' and 'unknown', make up less than 10% of all predictions at HFSP≥14 (false discover rate, FDR=9.6%), whereas between the default cut-off and HFSP=14 (0≤HFSP<14) this fraction is nearly 40%. Despite the fact, that at this threshold only ~6% of all predictions are of 'unknown' origin, these are still 30% of all 'unknown' proteins; similarly ~3% of all predictions, but 29% of all 'known incorrect' proteins are at HFSP≥14. These observations suggest that while we can not differentiate incorrect predictions from missing-reference ones, HFSP handles new protein function, as well as that which it has already seen, with higher scores indicating more reliable/correct annotations.

Given the vast number of proteins that yet have to be functionally annotated (*e.g.* TrEMBL is currently approaching 109 million proteins), the number of potential EC functionalities missing from our reference set, as well as the understanding that the total number of enzymes among the un-annotated proteins may not mirror the Swiss-Prot distribution (where ~47% of all proteins are annotated enzymes including those with incomplete and multiple EC annotations), we suspect that accurately estimating the HFSP cutoff at which the FDR would fall below some threshold, *e.g.* 5% (currently at HFSP≥28), is not possible. For example, given the current distribution of scores, 29% of 1,384 'unknowns' and incorrect 'knowns' present at HFSP≥14 make up only 407 proteins. If we were annotating tens of millions of proteins, however, this error rate can be expected to produce hundreds of thousands of annotations. On the other hand, given the limited size of our reference database, we can not necessarily expect that the true positive findings would grow accordingly.

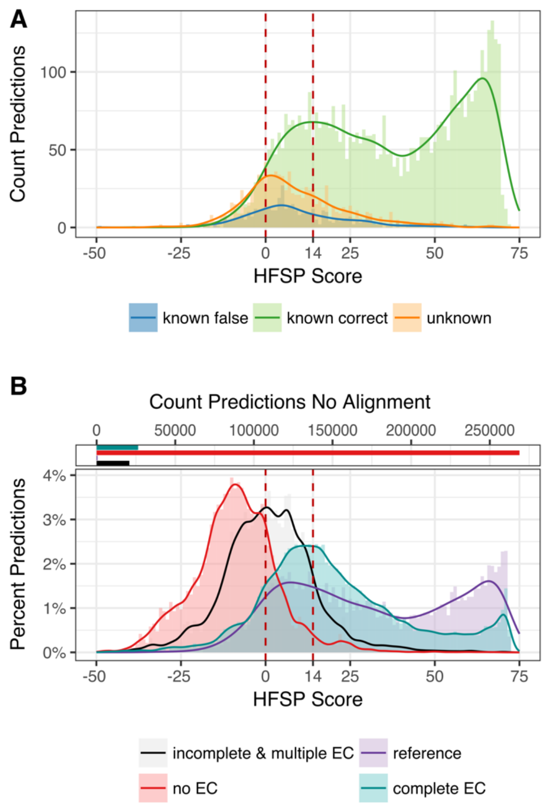


Fig. 7: HFSP is robust to previously unseen enzymatic functionality. (A) Proteins with no known homologs – approximated by investigating experimentally annotated proteins which fall into a EC category unique to the protein (orange) – show on average smaller highest scoring HFSP hits than proteins with existing homologs (green – correct predictions, blue – incorrect predictions). Of all predictions at HFSP score ≥ 14, less than 10% of proteins with "unknown" and "known" but falsely predicted function where observed **(B, bottom panel):** Highest HFSP score predictions for different protein subsets of the non-reduced Swiss-Prot: (i) experimentally verified enzymes (reference - purple), (ii) enzymes with EC annotation complete on all 4 levels that are not experimentally verified (complete EC - teal), (iii) enzymes with incomplete EC annotation or multiple EC annotations (incomplete & multiple EC – black) and (iv) proteins that are not annotated as enzymes (no EC – red); note that for most proteins with no EC annotation there were no matched to the reference database (268,857 proteins, 91%; **B, top panel**).

We further predicted EC annotation for all Swiss-Prot (555,594 proteins in October 2017, Figure 7B). Importantly, the majority (91%) of the non-enzymes (no EC annotations; 293,058 proteins) did not generate any matches to our reference database. Of the remaining non-enzymes, 21% (4,987 proteins) scored at HFSP ≥0, making up 3% of all predictions (false positives, 1% for all predictions at HFSP ≥14). Predictions could be made for 57% of the enzymes with multiple or incomplete EC annotations (27,717 of 48,536 proteins); 53% (14,668 proteins) of these scored at HFSP ≥0 and 13% above HFSP ≥14 (3,653 proteins). If these proteins were like our 'unknowns', we would expect at least twice as many with a match at HFSP ≥14. Thus, we suspect, that the enzymes in this set are not especially novel and can likely be annotated using HFSP and our reference dataset. This further suggests that at least 73% (43% no hits and 30% below HFSP=0) of proteins with incomplete or multiple EC annotations could be proteins with no homologous sequence in our reference database.

In light of our findings, we note that without further experimental work to elaborate on the functions of the yet-unannotated proteins, even the best function prediction methods will soon reach their limits. We suggest that using HFSP cutoffs can help in both more accurately annotating protein function and, arguably even more importantly, in identifying new frontiers of molecular function exploration.

4 Conclusion

While experimental function annotation of proteins is more accurate, computational methods are more readily available for the vast amount of sequences currently in our databases. Here we demonstrated that our newly developed HFSP (homology-derived functional similarity of proteins) is a fast an accurate method applicable to this task. Applying HFSP to evaluate existing annotations we also highlighted inconsistencies in existing annotations of enzymatic activity reported in Swiss-Prot. We thus suggest that HFSP provides both a way to (1) enrich functional annotation analysis on a large scale, as well as to (2) narrow down the space of proteins of interest for further experimental analysis.

Acknowledgements

We thank Drs. Chengsheng Zhu and Max Miller (both Rutgers), for all help with interpreting our data and providing feedback on the manuscript. We also want to extend our deepest gratitude to Dr. Peter Kahn (Rutgers) for his valuable feedback and help in finalizing the manuscript. Big thanks to Dr. Predrag Redivojac, Jiang Yuxiang (both Indiana U) and Yanran Wang (Rutgers) for valuable discussions. Last but not least, we thank all those who deposit their experimental data in public databases and those who maintain these databases.

Funding

YB was supported by the NSF CAREER Award 1553289, NIH U01 GM115486, and USDA-NIFA 1015:0228906; YB and YM by the TU Munich (TUM) – Institute for Advanced Study Hans Fischer Fellowship (TUM-IAS), funded by the German Excellence Initiative and the EU Seventh Framework Programme, grant agreement 291763.

Conflict of Interest: none declared.

References

Altschul, S.F., *et al.* Basic local alignment search tool. *J Mol Biol* 1990;215(3):403-410.

Altschul, S.F., *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* 1997;25(17):3389-3402.

Ashburner, M., *et al.* Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet* 2000;25(1):25-29.

Bairoch, A. The ENZYME database in 2000. *Nucleic Acids Res* 2000;28(1):304-305.

Basyuni, M., *et al.* Triterpene synthases from the Okinawan mangrove tribe, Rhizophoraceae. *FEBS J* 2007;274(19):5028-5042.

Benson, D.A., *et al.* GenBank. *Nucleic Acids Research* 2013;41(D1):D36-D42.

Berman, H.M., *et al.* The Protein Data Bank. *Acta Crystallogr D Biol Crystallogr* 2002;58(Pt 6 No 1):899-907.

6. HFSP*: High speed homology-driven function annotation of proteins*

Article short title

Clark, W.T. and Radivojac, P. Analysis of protein function and its prediction from amino acid sequence. *Proteins* 2011;79(7):2086-2096.

Daniels, N.M.*, et al.* Compressive genomics for protein databases. *Bioinformatics* 2013;29(13):i283-290.

Frith, M.C.*, et al.* The whole alignment and nothing but the alignment: the problem of spurious alignment flanks. *Nucleic Acids Res* 2008;36(18):5863-5871.

Fu, L.*, et al.* CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics* 2012;28(23):3150-3152.

Jiang, Y.*, et al.* An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome Biol* 2016;17(1):184.

Khan, I.*, et al.* Genome-scale identification and characterization of moonlighting proteins. *Biol Direct* 2014;9:30.

Li, W. and Godzik, A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* 2006;22(13):1658-1659.

Loewenstein, Y.*, et al.* Protein function annotation by homology-based inference. *Genome Biol* 2009;10(2):207.

Nelder, J.A. and Mead, R. A Simplex Method for Function Minimization. *The Computer Journal* 1965;7(4):308-313.

Placzek, S.*, et al.* BRENDA in 2017: new perspectives and new tools in BRENDA. *Nucleic Acids Res* 2017;45(D1):D380-D388.

Radivojac, P.*, et al.* A large-scale evaluation of computational protein function prediction. *Nat Methods* 2013;10(3):221-227.

Remmert, M.*, et al.* HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nat Methods* 2011;9(2):173-175.

Rost, B. Twilight zone of protein sequence alignments. *Protein Eng* 1999;12(2):85-94.

Rost, B. Enzyme function less conserved than anticipated. *J Mol Biol* 2002;318(2):595-608.

Rost, B. and Sander, C. Prediction of protein secondary structure at better than 70% accuracy. *J Mol Biol* 1993;232(2):584-599.

Sander, C. and Schneider, R. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins* 1991;9(1):56-68.

Schnoes, A.M.*, et al.* Annotation error in public databases: misannotation of molecular function in enzyme superfamilies. *PLoS Comput Biol* 2009;5(12):e1000605.

Steinegger, M. and Soding, J. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat Biotechnol* 2017;35(11):1026-1028.

The UniProt, C. UniProt: the universal protein knowledgebase. *Nucleic Acids Res* 2017;45(D1):D158-D169.

Wei, Q. and Dunbrack, R.L., Jr. The role of balanced training and testing data sets for binary classifiers in bioinformatics. *PLoS One* 2013;8(7):e67863.

Zhu, C.*, et al.* Functional Basis of Microorganism Classification. *PLoS Comput Biol* 2015;11(8):e1004472.

Zhu, C.*, et al.* fusionDB: assessing microbial diversity and environmental preferences via functional similarity networks. *Nucleic Acids Res* 2018;46(D1):D535-D541.

Chapter 7

Conclusions

Over the course of the last decade, the amount sequencing data has rapidly increased due to advances in high-throughput sequencing technology. Through these advances, metagenomic studies have flourished. Mining the hidden sequence treasures in complex metagenomic samples became a computationally challenging task. In this thesis I presented methods to close the cost gap between data generation and their processing through sequence search and clustering.

MMseqs introduced cascaded clustering and the greedy set cover algorithm to protein clustering. It made it feasible to cluster huge protein sets down to 30% sequence identity, while being fast and more sensitive than state of the art methods. The fast prefilter and the clustering based on similarity graphs could in the future not only be used for protein sequences, but also for sequential or vector data. Especially interesting is the possibility to cluster longitudinal behavioural data, created from animal experiments or tracking data from humans collected through smart devices. Clustering of similar behaviour could be linked to diseases, which could be used to create early detection and prevention systems. Currently, we are working on an extension to cluster gene expression profiles.

MMseqs2: Up until now, methods for homology inference had to trade sensitivity for speed and no method came close to the sensitivity of BLAST or PSI-BLAST in our benchmark. MMseqs2 now finally closes this speed-sensitivity gap. The MMseqs2 algorithm could also easily be extended to match nucleotide sequences. The prefilter could be adapted to produce nucleotide k-mers matching the error model of DNA sequencers instead of protein substitution matrices. This would enable a fast and sensitive error aware matching of nucleotide reads. Another application would be a general purpose search engine for the detection of similar text fragments containing spelling mistakes. The only changes necessary would be an increased alphabet size and a spelling error models instead of a protein substitution matrices. Currently, I am working on extensions to support profile-profile searches by using a 32-state alphabet representing profile columns. This will increase the sensitivity and speed of searches.

Linclust proposes the first $O(N)$ protein sequence clustering algorithm and implementation. It can process billions of sequences in a single day on a single server. The approach can be easily extended to a general purpose clustering algorithm for D-dimensional vector data. Currently, I am working to extend the algorithm into two areas: (1) the support of nucleotide sequences and (2) to developing an overlap graph based protein level assembler (PLASS). The Linclust algorithm enables PLASS to compute the overlap assembly graph in linear time, instead of near quadratic time.

The new era of data generation calls for fast algorithms. The novel algorithms proposed in this work will not only help to organise our protein sequence space, but enable previously infeasible large-scale analyses.

Appendices

Appendix A

User Guide

A.1 Introduction

The user guide for MMseqs2 and Linclust follows. It contains a detailed documentation of this software suite.

Martin Steinegger, Maria Hauser, Milot Mirdita, Lars von den Driesch, Clovis Galiez, Eli Levy Karin and Johannes Söding wrote the user guide.

A.2 Userguide

MMseqs2 (Many-against-Many searching) is a software suite to search and cluster huge sequence sets. MMseqs2 is open source GPL-licensed software implemented in C++ for Linux, Mac OS and Windows. The software is designed to run on multiple cores and servers and exhibits very good scalability. MMseqs2 reaches the same sensitivity as BLAST magnitude faster and which can also perform profile searches like PSI-BLAST but also 400 times faster.

At the core of MMseqs2 are two modules for the comparison of two sequence sets with each other - the prefiltering and the alignment modules. The first, prefiltering module computes the similarities between all sequences in one query database with all sequences a target database based on a very fast and sensitive k-mer matching stage followed by an ungapped alignment. The alignment module implements an vectorized Smith-Waterman alignment of all sequences that pass a cut-off for the ungapped alignment score in the first module. Both modules are parallelized to use all cores of a computer to full capacity. Due to its unparalleled combination of speed and sensitivity, searches of all predicted ORFs in large metagenomics data sets through the entire UniProtKB or NCBI-NR databases are feasible. This allows for assigning to functional clusters and taxonomic clades many reads that are too diverged to be mappable by current software.

MMseqs2 clustering module can cluster sequence sets efficiently into groups of similar sequences. It takes as input the similarity graph obtained from the comparison of the sequence set with itself in the prefiltering and alignment modules. MMseqs2 further supports an updating mode in which sequences can be added to an existing clustering with stable cluster identifiers and without the need to recluster the entire sequence set. We are using MMseqs2 to regularly update versions of the UniProtKB database clustered down to 30% sequence similarity threshold. This database is available at uniclust.mmseqs.com.

System Requirements

MMseqs2 runs on modern UNIX operating systems and is tested on Linux and OSX. Additionally, we are providing a preview version for Windows.

The alignment and prefiltering modules are using with SSE4.1 (or optionally AVX2) and OpenMP, i.e. MMseqs2 can take advantage of multicore computers.

When searching large databases, MMseqs2 may need a lot main memory (see section memory requirements). We offer an option for limiting the memory usage at the cost of longer runtimes. The database is split into chunks and the program only holds one chunk in memory at any time. For clustering large databases containing tens of millions of sequences, you should provide enough free disk space (~500 GB). In section Optimizing Sensitivity and Consumption of Resources, we will discuss the runtime, memory and disk space consumption of MMseqs2 and how to reduce resource requirements for large databases.

Check system requirements

To check if MMseqs2 supports your system execute the following commands, depending on your operating system: #### Linux

```
[[ $(uname -m) == "x86_64" ]] && echo "64bit Supported" || echo "64bit Unsupported"
cat /proc/cpuinfo | grep -c sse4_1 > /dev/null \
&&  echo "SSE4.1 Supported" || echo "SSE4.1 Unsupported"
cat /proc/cpuinfo | grep -c avx2 > /dev/null \
&& echo "AVX2 Supported" || echo "AVX2 Unsupported"
```

MacOS

```
[[ $(uname -m) == "x86_64" ]] && echo "64bit Supported" || echo "64bit Unsupported"
sysctl -a | grep machdep.cpu.features | grep -c SSE4.1 > /dev/null \
&& echo "SSE4.1 Supported" || echo "SSE4.1 Unsupported"
sysctl -a | grep machdep.cpu.leaf7_features | grep -c AVX2 > /dev/null \
&& echo "AVX2 Supported" || echo "AVX2 Unsupported"
```

Windows The `mmseqs.bat` script will print a message if its run on an unsupported system. On a supported system, it will execute the correct MMseqs2 version and forward all parameters.

*

Installation

MMseqs2 can be installed by downloading a statically compiled version, compiling the from source, using Homebrew or Docker.

Install static Linux version

The following command will download the lastest MMseqs2 version, extract it and set the `PATH` variable.

Linux If your computer supports AVX2 use:

```
wget https://mmseqs.com/latest/mmseqs-static_avx2.tar.gz
tar xvzf mmseqs-static_avx2.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

If your computer supports SSE4.1 use:

```
wget https://mmseqs.com/latest/mmseqs-static_sse41.tar.gz
tar xvzf mmseqs-static_sse41.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

Mac If your computer supports AVX2 use:

```
wget https://mmseqs.com/latest/mmseqs-osx-static_avx2.tar.gz
tar xvzf mmseqs-osx-static_avx2.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

If your computer supports SSE4.1 use:

```
wget https://mmseqs.com/latest/mmseqs-osx-static_sse41.tar.gz
tar xvzf mmseqs-osx-static_sse41.tar.gz
export PATH=$(pwd)/mmseqs/bin/:$PATH
```

Windows (preview) The latest version is always available on:

```
https://mmseqs.com/latest/mmseqs-win64.zip
```

Download and unzip it at a convenient location. Inside you will find the `mmseqs.bat` wrapper script, which should be used to substitute all calls to `mmseqs` in the remainder of this document, and a `bin` folder with all dependencies of the MMseqs2 Windows version. Please always keep the `mmseqs.bat` script one folder above the `bin` folder, or it will not be able to correctly identify its dependencies anymore.

The windows build also contains both the SSE4.1 and the AVX2 version. The `mmseqs.bat` script will automatically choose the correct one.

Compile from source

Compiling MMseqs2 from source has the advantage that it will be optimized to the specific system, which should improve its performance. To compile MMseqs2 `git`, `g++` (4.6 or higher) and `cmake` (3.0 or higher) are needed. Afterwards, the MMseqs2 binary will be located in `build/bin/`.

```
git clone https://github.com/soedinglab/MMseqs2.git
cd MMseqs2
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. ..
make
make install
export PATH=$(pwd)/bin/:$PATH
```

:exclamation: On MacOS, please install the `gcc@7 zlib bzip2 vim cmake` packages from Homebrew, if you want to compile MMseqs2. The default MacOS `clang` compiler does not support OpenMP and MMseqs2 will not be able to run multithreaded. Use the following cmake call:

```
CXX="$(brew --prefix)/bin/g++-7" cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. ..
```

Windows The windows build process is more involved due to MMseqs2's dependency on an installed shell. We use the Cygwin environment and Busybox to provide all necessary dependencies and bundle them all together. If you want to compile MMseqs2 on your own, install the following packages from Cygwin:

```
bash xxd cmake make gcc-g++ zlib-devel libbz2-devel busybox-standalone binutils
```

Afterwards, use a workflow similar to the `util/build_windows.sh` script to build MMseqs2 on Windows.

Install with Homebrew

You can install MMseqs2 for Mac OS through Homebrew by executing the following:

```
brew install mmseqs2
```

This will also automatically install the bash completion (you might have to execute `brew install bash-completion` first). This will also work for Linuxbrew.

Use the Docker image

You can pull the official docker image by running:

```
docker pull soedinglab/mmseqs2
```

If you want to build the docker image from the git repository, execute:

```
git clone https://github.com/soedinglab/MMseqs2.git
cd MMseqs2
docker build -t mmseqs2 .
```

Use the BASH command completion

MMseqs comes with a bash command and parameter auto completion by pressing tab. The bash completion for subcommands and parameters can be installed by adding the following lines to your $HOME/.bash_profile:

```
if [ -f /Path to MMseqs2/util/bash-completion.sh ]; then
    source /Path to MMseqs2/util/bash-completion.sh
fi
```

*

Getting Started

Here we explain how to run a search for sequences matches in the query database against a target database and how to cluster a sequence database. Test data (a query and a target database for the sequence search and a database for the clustering) are stored in the `examples` folder.

Search

Before searching, you need to convert your FASTA file containing query sequences and target sequences into a sequence DB. You can use the query database `examples/QUERY.fasta` and target database `examples/DB.fasta` to test the search workflow:

```
$ mmseqs createdb examples/QUERY.fasta queryDB
$ mmseqs createdb examples/DB.fasta targetDB
```

These calls should generates five files each, e.g. `queryDB`, `queryDB_h` and its corresponding index file `queryDB.index`, `queryDB_h.index` and `queryDB.lookup` from the FASTA `QUERY.fasta` input sequences.

The `queryDB` and `queryDB.index` files contain the amino acid sequences, while the `queryDB_h` and `queryDB_h.index` file contain the FASTA headers. The `queryDB.lookup` file contains a list of tab separated fields that map from the internal identifier to the FASTA identifiers.

Important: `createdb` splits long sequences into multiple separate entries automatically. This avoids excessive resource requirements for later steps. The default value is to split sequences after 32000 residues. The identifiers of the new entries are suffixed with `_0` to `_(n-1)` for `N` splits.

For the next step, an index file of the `targetDB` is computed for a fast read in. It is recommend to compute the index if the `targetDB` is reused for several searches.

```
$ mmseqs createindex targetDB tmp
```

This call will create a `targetDB.sk6` file. In this file extension the letter `s` indicates the use of spaced k-mers and the `k6` shows the k-mer size of 6.

Then generate a directory for temporary files. MMseqs2 can produce a high IO on the file system. It is recommended to create this temporary folder on a local drive.

```
$ mkdir tmp
```

Please ensure that in case of large input databases `tmp` provides enough free space. For the disk space requirements, see the section Disk Space.

The alignment consists of two steps the `prefilter` and `alignment`. To run the search, type:

```
$ mmseqs search queryDB targetDB resultDB tmp
```

Search as standard does compute the score only. If you need the alignment information add the option "-a".

Then, convert the result database into a BLAST tab formatted file (option `-m 8` in legacy blast, `-outfmt 6` in blast+):

```
$ mmseqs convertalis queryDB targetDB resultDB resultDB.m8
```

The file is formatted as a tab-separated list with 12 columns: (1,2) identifiers for query and target sequences/profiles, (3) sequence identity, (4) alignment length, (5) number of mismatches, (6) number of gap openings, (7-8, 9-10) domain start and end-position in query and in target, (11) E-value, and (12) bit score.

Read more about searching here.

Clustering

Before clustering, convert your FASTA database into the MMseqs database (DB) format:

```
$ mmseqs createdb examples/DB.fasta DB
```

Then, generate a directory for tmp files:

```
$ mkdir tmp
```

Please ensure that in case of large input databases `tmp` provides enough free space. For the disk space requirements, see the section Disk space.

Run the clustering of your database `DB` by executing the following command. MMseqs2 will return the result database files `DB_clu`, `DB_clu.index`:

```
$ mmseqs cluster DB DB_clu tmp
```

To generate a TSV formatted output file from the output file, type:

```
$ mmseqs createtsv DB DB DB_clu DB_clu.tsv
```

You can adjust the sequence identity threshold with `--min-seq-id` and the alignment coverage with `-c` and `--cov-mode`. MMseqs2 will set the sensitivity parameters automatic based on target sequence identity (`--min-seq-id`), if it is not already specified through the `-s` or `--k-score` parameters.

Sequence information can be added by using createseqfiledb and result2flat can produce a result.

```
$ mmseqs createseqfiledb DB DB_clu DB_clu_seq
$ mmseqs result2flat DB DB DB_clu_seq DB_clu_seq.fasta
```

Read more about clustering here.

Linclust

Linclust is a clustering in linear time. It is magnitudes faster but a bit less sensitive than clustering.

Before clustering, convert your FASTA database into the MMseqs database (DB) format:

```
$ mmseqs createdb examples/DB.fasta DB
```

Then, generate a directory for tmp files:

```
$ mkdir tmp
```

To run linclust the clustering of your database `DB` by executing the following command. The result database follows the same format as the clustering format:

```
$ mmseqs linclust DB DB_clu tmp
```

To extract the representative sequences from the clustering result call:

```
mmseqs result2repseq DB DB_clu DB_clu_rep
mmseqs result2flat DB DB DB_clu_rep DB_clu_rep.fasta  --use-fasta-header
```

Updating a clustered database

It is possible to update previous clustered databases without re-clustering everything from scratch. Here is an example of how to proceed:

Let's create an older version of the DB.fasta by trimming out some sequences:

```
$ cd examples
$ awk '/^>/{seqCount++;} {if (seqCount <= 19000) {print $0;}}' DB.fasta > DB_trimmed.fasta
```

Now we create the sequence DBs, and cluster the old (trimmed) database:

```
$ mmseqs createdb DB.fasta DB_new
$ mmseqs createdb DB_trimmed.fasta DB_trimmed
$ mmseqs cluster DB_trimmed DB_trimmed_clu tmp
```

If you want to update DB_trimmed_clu with the newer version of your database DB_new

```
$ rm tmp/*
$ mmseqs clusterupdate DB_trimmed DB_new DB_trimmed_clu DB_clusterupdate tmp
```

DB_clusterupdate contains now the fresh clustering of DB_new.

Read more about updating cluster here.

*

Overview of Folders in MMseqs

- `bin`: `mmseqs`
- `data`: BLOSUM matrices and the workflow scripts (`blastp.sh`, `blastpgp.sh`, `cascaded_clustering.sh`, `linclust.sh`, `searchtargetprofile.sh`, `clustering.sh`)
- `examples`: test data `QUERY.fasta` and `DB.fasta`
- `util`: Contains the Bash parameter completion script.

Overview of MMseqs2 Commands

MMseqs2 contains five workflows that combine the core MMseqs2 modules (prefilter, align, kmermatcher, rescorediagonal and clust) and several other smaller ones.

Workflows:

- `mmseqs search`: Compares all sequences in the query database with all sequences in the target database, using the prefiltering and alignment modules. MMseqs2 search supports sequence/sequence, profile/sequence or sequence/profile searches.

- `mmseqs cluster`: Clusters sequences by similarity. It compares all sequences in the sequence DB with each other using mmseqs search, filters alignments according to user-specified criteria (max. E-value, min. coverage,...), and runs mmseqs clust to group similar sequences together into clusters.
- `mmseqs linclust`: Clusters sequences by similarity in linear time. It clusteres magnitudes faster than `mmseqs cluster` but is less sensitive.
- `mmseqs clusterupdate`: MMseqs2 incrementally updates a clustering, given an existing clustering of a sequence database and a new version of this sequence database (with new sequences being added and others having been deleted).
- `mmseqs taxonomy` Taxonomy assignment by computing the lowest common ancestor of homologs.

And the four core modules:

- `mmseqs prefilter`: Computes k-mer similarity scores between all sequences in the query database and all sequences in the target database.
- `mmseqs kmermatcher`: finds exact k-mer matches between all input sequences in linear time.
- `mmseqs align`: Computes Smith-Waterman alignment scores between all sequences in the query database and the sequences of the target database whose prefiltering scores computed by `mmseqs prefilter` pass a minimum threshold.
- `mmseqs clust`: Computes a similarity clustering of a sequence database based on Smith Waterman alignment scores of the sequence pairs computed by `mmseqs align`.

MMseqs2 has more than 30 modules in total. We provide modules for clustering, searching, alignments, taxonomy, and data transformation. For a complete list of all available modules, execute `mmseqs` without arguments.

Description of Workflows

Batch Sequence Searching using `mmseqs search`

For searching a database, query and target database have to be converted by `createdb` in order to use them in MMseqs. The search can be executed by typing:

```
$ mmseqs search queryDB targetDB outDB tmp
```

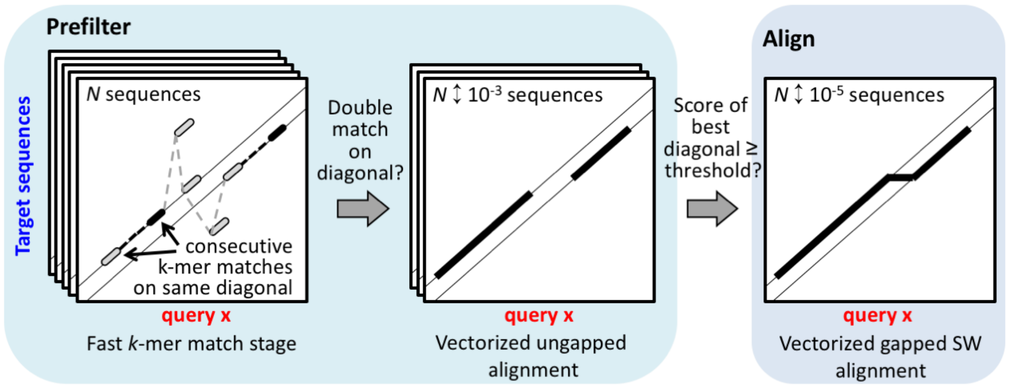

Prefilter
Target sequences
N sequences
consecutive k-mer matches on same diagonal
query x
Fast k-mer match stage
Double match on diagonal?
N ↕ 10-3 sequences
query x
Vectorized ungapped alignment
Score of best diagonal ≥ threshold?
Align
N ↕ 10-5 sequences
query x
Vectorized gapped SW alignment

Fig. A.1. Search workflow

MMseqs2 supports iterative searches which are similar to PSI-BLAST. The following program call will run two iterations through the database. In the first iteration sequences are searched against sequence and in the second one profiles are used to search against sequences.

MMseqs2 will use the output for the first iteration sequence-sequence search to computes a profile (result2profile). The profile will be used as input in the next search iteration.

```
$ mmseqs search queryDB targetDB outDB tmp --num-iterations 2
```

This workflow combines the prefiltering and alignment modules into a fast and sensitive batch protein sequence search that compares all sequences in the query database with all sequences in the target database.

Query and target databases may be identical. The program outputs for each query sequence all database sequences satisfying the search criteria (such as sensitivity).

MMseqs2 can precompute the prefilter index `createindex` to speed up subsequence prefilter index read-ins. We recommend to use an index for iterative searches or if a target database will be reused several times. However reading the index can be bottle neck when using a network file systems (NFS). It is recommended to keep the index on a local hard drive. If storing the index file on a local hard drive is not possible and the NFS is a bottleneck than do not precompute the index. MMseqs2 will compute an index on the fly which reduces the IO volume by roughly a factor of seven.

The underlying algorithm is explained in more detail in section Computation of Prefiltering Scores using mmseqs prefilter, and the important parameter list can be found in section Search Workflow.

Translated Sequence Searching

The search workflow can handle nucleotide as query or target database. It will trigger a BLASTX or TBLASTN search respectively. The search detects the open reading frames on all six frames and translates them into proteins. As default the minimum codon length of 30 (10 amino acids) is used.

To perform a search a BLASTX or TBLASTN create your database by using `createdb`. It can automatically detect if the input are amino acids or nucleotides.

```
mmseqs createdb ecoli.fna ecoli_genome --dont-split-seq-by-len
mmseqs createdb ecoli.faa ecoli_proteins
```

A BLASTX like search ca be triggered using the nucleotide database on the query database side.

```
mmseqs search ecoli_genome ecoli_proteins alnDB tmp
```

A TBLASTN like search ca be triggered using the nucleotide database on the target database side.

```
mmseqs search ecoli_proteins ecoli_genome alnDB tmp
```

It is not possible to use nucleotide databases on query and target sides (TBLASTX) of the search workflow. The following workflow can be used to perform a TBLASTX search:

```
mmseqs extractorfs genome genome_orfs --longest-orf --min-length 30 --max-length 48000
mmseqs translatenucs genome_orfs genome_orfs_aa
mmseqs translatenucs ecoli_genome ecoli_genome_aa
mmseqs search genome_orfs_aa ecoli_genome alnDB tmp
mmseqs offsetalignment genome_orfs ecoli_genome_aa alnDB alnOffsetedDB
```

All open reading frames (ORFs) from each six frames can be extracted by using `extractorf`. This ORFs are translated into proteins by `translatenucleotide`. The tool `offsetalignment` will offset the alignment position to the `orf start position + alignment start * 3`.

Clustering Databases using `mmseqs cluster`

To cluster a database, MMseqs2 needs a sequence database converted with `createdb` and an empty directory for temporary files. Then, you can run the clustering with:

```
$ mmseqs cluster inDB outDB tmp
```

and cascaded clustering with:

```
$ mmseqs cluster inDB outDB tmp --cascaded
```

The sensitivity of the clustering can be adjusted with the `-s` option. MMseqs2 will automatically adjust the sensitivity based on the `--min-seq-id` parameter, if neither `--cascaded` nor `-s` are provided.

```
$ mmseqs cluster inDB outDB tmp
```

The clustering workflow combines the prefiltering, alignment and clustering modules into either a simple clustering or a cascaded clustering of a sequence database. There are two ways to execute the clustering:

- The *Simple clustering* runs the hashclust and prefiltering, alignment and clustering modules with predefined parameters with a single iteration.
- *Cascaded clustering* clusters the sequence database using the as first step linclust and then prefiltering, alignment and clustering modules incrementally in three steps.

Cascaded Clustering The cascaded clustering workflow first runs `linclust`, our linear-time clustering module, that can produce clustering's down to 50% sequence identity in very short time.

To achieve lower sequence identities and/or to further improve the resulting clusters, we continue with three cascaded clustering steps: In the first step of the cascaded clustering the prefiltering runs with a low sensitivity of 1 and a very high result significance threshold, in order to accelerate the calculation and search only for hits with a very high sequence identity. Then alignments are calculated and the database is clustered. The second step takes the representative sequences of the first clustering step and repeats the prefiltering, alignment and clustering steps. This time, the prefiltering is executed with a higher sensitivity and a lower result significance threshold for catching sequence pairs with lower sequence identity. In the last step, the whole process is repeated again with the final target sensitivity. At last, the clustering results are merged and the resulting clustering is written to the output database.

Cascaded clustering yields more sensitive results than simple clustering. Also, it allows very large cluster sizes in the end clustering resulting from cluster merging (note that cluster size can grow exponentially in the cascaded clustering workflow), which is not possible with the simple clustering workflow because of the limited maximum number of sequences passing the prefiltering and the alignment. Therefore, we strongly recommend to use cascaded clustering especially to cluster larger databases and to obtain maximum sensitivity.

Clustering modes All clustering modes transforms the alignment results into an undirected graph. In this graph notation, the verticies represents the proteins, which are connected by an edge. An edge between proteins are introduced if the alignment criteria (e.g. `--min-seq-id`, `-c` and `-e`) are fulfilled.

Greedy Set cover (`--cluster-mode 0`) algorithm is an approximation for the NP-complete optimization problem called set cover.

Greedy set cover removes the node with most connections and all connected nodes. These forms a cluster and the procedure repeats until all nodes are in a cluster. The greedy

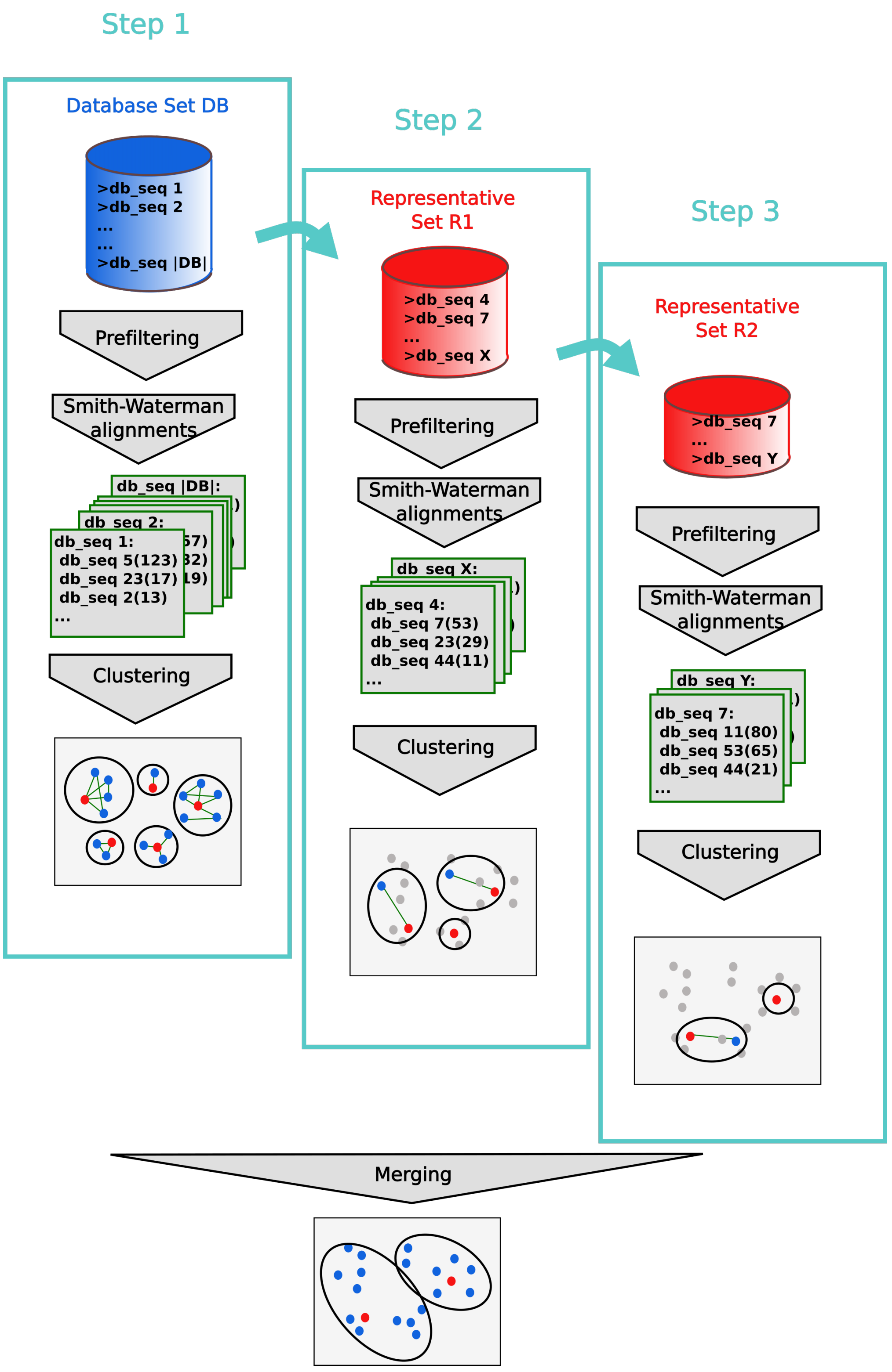


Fig. A.2. Cascaded clustering

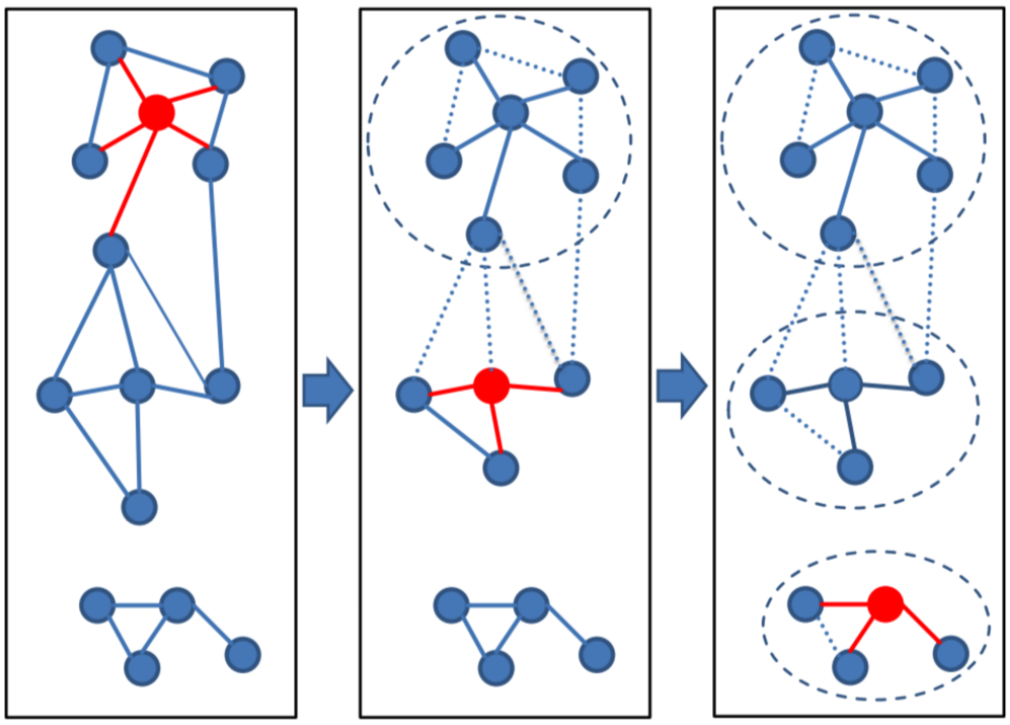


Fig. A.3. Set Cover clustering

set cover is followed by a reassignment step. Cluster member are assigned to another cluster centroid if their alignment score was higher.

Connected component (`--cluster-mode 1`) uses transitive connection to cover more remote homologs.

In connected component clustering starting at the mostly connected vertex, all vertices that are reachable in a breadth-first search are members of the cluster.

Greedy incremental (`--cluster-mode 2`) works analogous to CD-HIT clustering algorithm.

Greedy incremental clustering takes the longest sequence (indicated by the size of the node) and puts all connected sequences in that cluster, then repeatedly the longest sequence of the remaining set forms the next cluster.

Linear time clustering using `mmseqs linclust`

Linclust can cluster sequences down to 50% pairwise sequence similarity and its runtime scales linearly with the input set size.

Linear-time clustering algorithm. Steps 1 and 2 find exact k-mer matches between the N input sequences that are extended in step 3 and 4.

(1) Linclust selects in each sequence the `m` (default: 20) k-mers with the lowest hash function values, as this tends to select the same k-mers across homologous sequences. It uses a reduced alphabet of 13 letters for the k-mers and sets k=10 for sequence

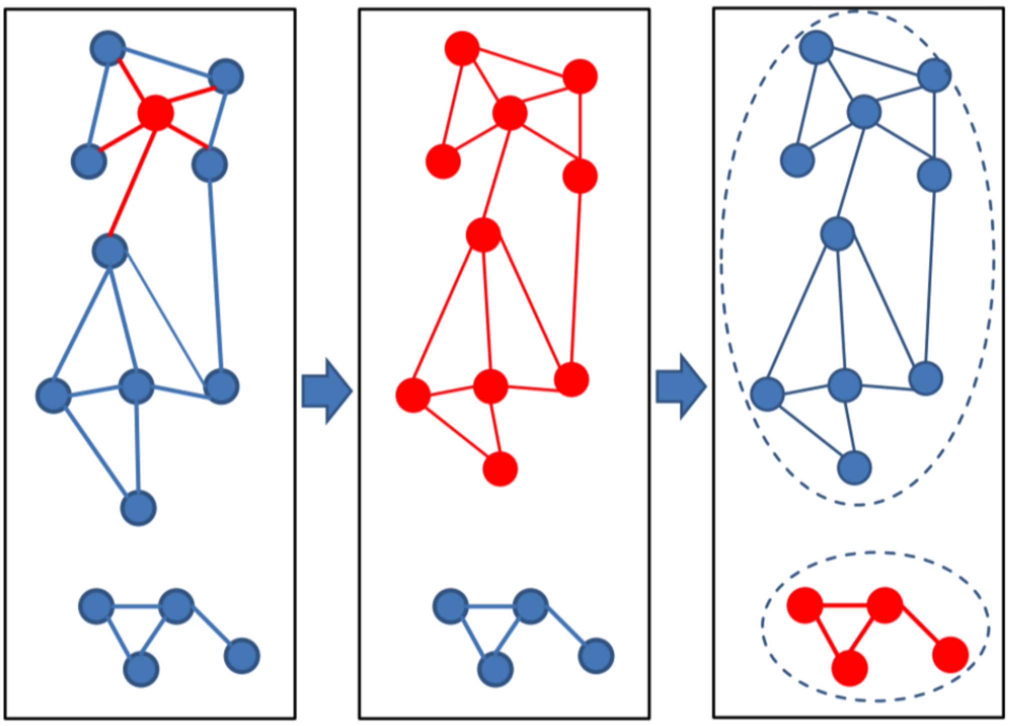


Fig. A.4. Connected component clustering

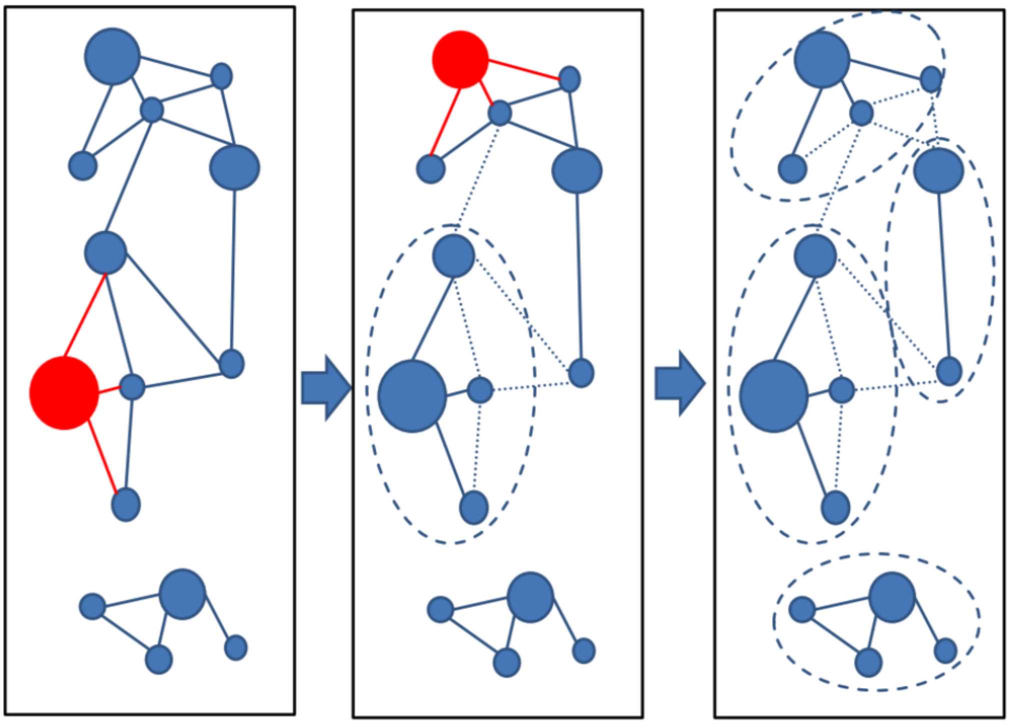


Fig. A.5. Greedy incremental clustering

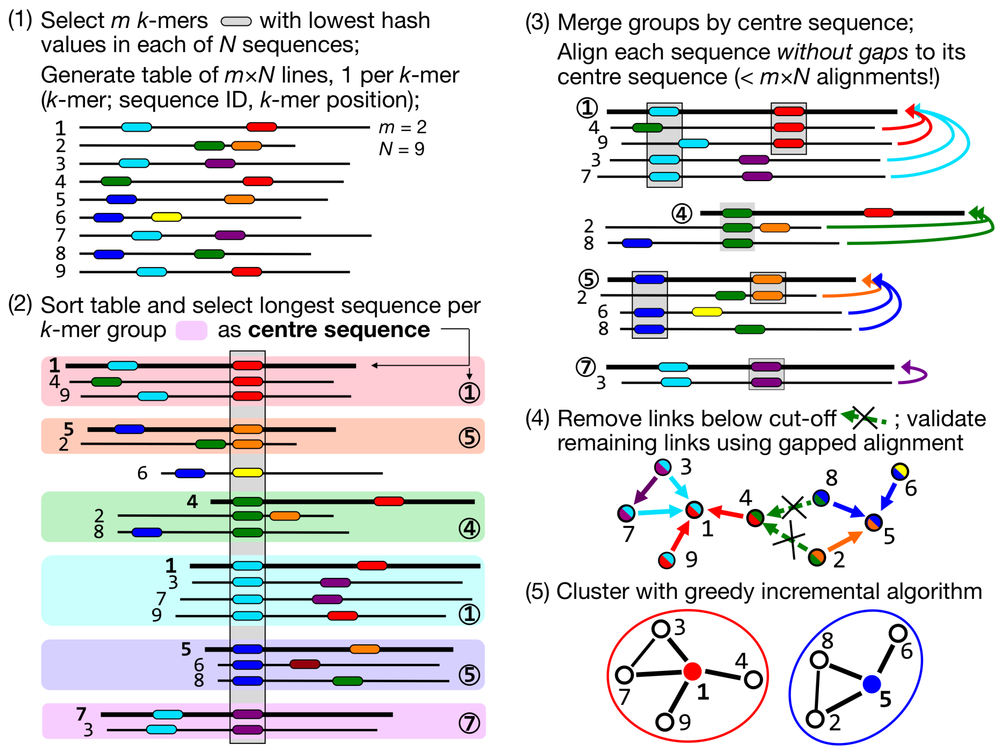

(1) Select m k-mers with lowest hash values in each of N sequences;
Generate table of m×N lines, 1 per k-mer (k-mer; sequence ID, k-mer position);
m = 2
N = 9
(2) Sort table and select longest sequence per k-mer group as centre sequence
(3) Merge groups by centre sequence; Align each sequence without gaps to its centre sequence (< m×N alignments!)
(4) Remove links below cut-off ; validate remaining links using gapped alignment
(5) Cluster with greedy incremental algorithm

Fig. A.6. Linclust algorithm

identity thresholds below 90% and k=14 above. It generates a table in which each of the mN lines consists of the *k*-mer, the sequence identifier, and the position of the *k*-mer in the sequence.

(2) Linclust sorts the table by *k*-mer in quasi-linear time, which identifies groups of sequences sharing the same k-mer (large shaded boxes). For each k-mer group, it selects the longest sequence as centre. It thereby tends to select the same sequences as centre among groups sharing sequences.

(3) It merges k-mer groups with the same centre sequence together: red + cyan and orange + blue and compares each group member to the centre sequence in two steps: by global Hamming distance and by gapless local alignment extending the k-mer match.

(4) Sequences above a score cut-off in step 3 are aligned to their centre sequence using gapped local sequence alignment. Sequence pairs that satisfy the clustering criteria (e.g. on the E-value, sequence similarity, and sequence coverage) are linked by an edge.

(5) The greedy incremental algorithm finds a clustering such that each input sequence has an edge to its cluster's representative sequence. Note that the number of sequence pairs compared in steps 3 and 4 is less than mN, resulting in a linear time complexity.

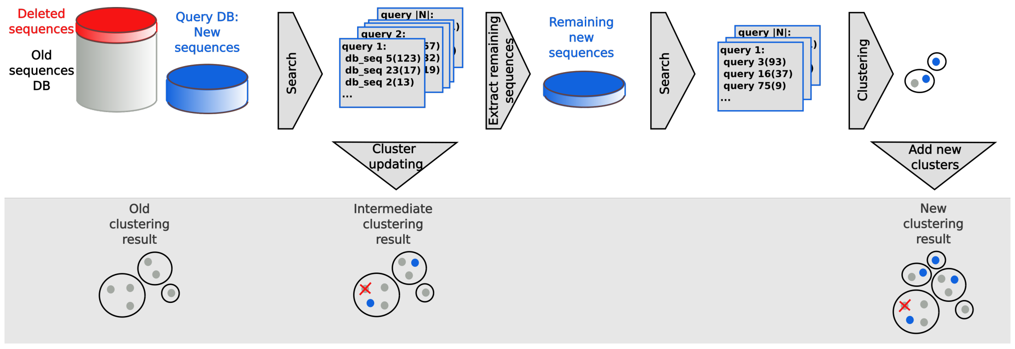


Fig. A.7. Update clustering

Run Linclust Linclust needs a sequence database created by `createdb` and an empty directory for temporary files. Then, you can run the clustering with the following command:

```
$ mmseqs linclust inDB outDB tmp
```

Increasing the k-mers selected per sequence increases the sensitivity of linclust at a moderate loss of speed. Use the paramter `--kmer-per-seq` to set the number of k-mers selected per sequence. More k-mers per sequences results in a higher sensitivity.

The output format of linclust is the same format as in `mmseqs cluster`. See section Clustering Format.

Updating a Database Clustering using `mmseqs clusterupdate`

To run the updating, you need the old and the new version of your sequence database in sequence db format, the clustering of the old database version and a directory for the temporary files:

```
$ mmseqs clusterupdate oldDB newDB oldDB_clustering outDB tmp
```

This workflow efficiently updates the clustering of a database by adding new and removing outdated sequences. It takes as input the older sequence database, the results obtained by this older database clustering, and the newer version of the sequence database. Then it adds the new sequences to the clustering and removes the sequences that were removed from the newer database. Sequences which are not similar enough to any existing cluster will be representatives of new clusters.

Taxonomy assignment using `mmseqs taxonomy`

By identifying homologs through searches with taxonomy annotated reference databases, MMseqs2 can compute the lowest common ancestor. This lowest common ancestor is a robust taxonomic label for unknown sequences.

MMseqs2 implements the 2bLCA protocol (Hingamp et. al., 2013) with `--lca-mode 2` (default) for choosing a robust LCA.

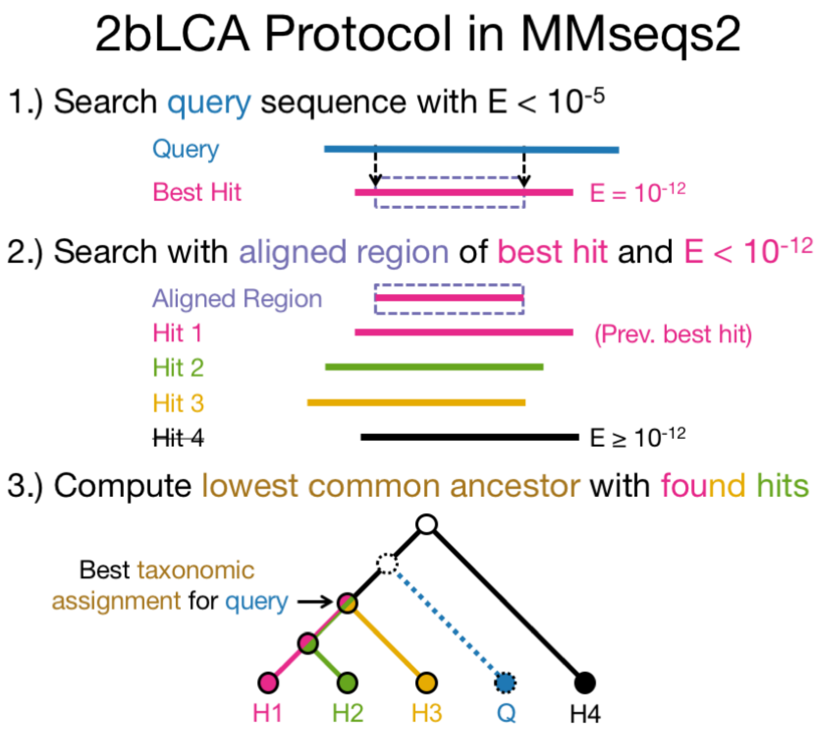

2bLCA Protocol in MMseqs2
1.) Search query sequence with E < 10-5
Query
Best Hit
E = 10-12
2.) Search with aligned region of best hit and E < 10-12
Aligned Region
Hit 1
(Prev. best hit)
Hit 2
Hit 3
~~Hit 4~~
E ≥ 10-12
3.) Compute lowest common ancestor with found hits
Best taxonomic assignment for query
H1
H2
H3
Q
H4

Fig. A.8. 2bLCA protcol

The second search can be disabled with `--lca-mode 1`. The LCA will then be only computed through the usual search workflow parameters (`--max-accept`, `-e`, etc.).

The LCA implementation is based on the Go implementation of blast2lca software on GitHub. It implements the LCA computation efficiently through *Range Minimum Queries* through an dynamic programming approach.

Prerequisites The taxonomy workflow requires the NCBI taxonomy taxdump.tar.gz. It is available on the NCBI FTP server:

```
mkdir ncbi-taxdump && cd ncbi-taxdump
wget ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz
tar xzvf taxdump.tar.gz
cd ..
```

Provide the path to the extraction location in the `mmseqs taxonomy` call as the `<i:NcbiTaxdmpDir>` parameter.

The workflow further requires a tab-separated mapping `<i:targetTaxonMapping>` with every target database identifier mapped to a NCBI taxon identifier. The `convertkb` module can generate this mapping for any database with UniProt accessions, such as the Uniclust, UniRef, and the UniProt itself:

```
# Turn the target sequences into a MMseqs2 database (this also creates targetDB.lookup)
# Skip this step if you already created a database
```

```
mmseqs createdb target.fasta targetDB

# The targetDB.lookup file should be in the following format:
# numeric-db-id tab-character UniProt-Accession (e.g. Q6GZX4)

# UniRef has a prefixed accession (e.g. UniRef100_Q6GZX4)
# Remove this prefix first:
# sed -i 's|UniRef100_||g' targetDB.lookup

# Download the latest UniProt Knowledgebase:
wget ftp://ftp.expasy.org/databases/.../uniprot_sprot.dat.gz
wget ftp://ftp.expasy.org/databases/.../uniprot_trembl.dat.gz
cat uniprot_sprot.dat.gz uniprot_trembl.dat.gz > uniprot_sprot_trembl.dat.gz

# Generate annotation mapping DB (target DB IDs to NCBI taxa, line type OX)
mmseqs convertkb uniprot_sprot_trembl.dat.gz targetDB.mapping \
--kb-columns OX --mapping-file targetDB.lookup

# Reformat targetDB.mapping_OX DB into tsv file
mmseqs prefixid targetDB.mapping_OX targetDB.mapping_OX_pref
tr -d '\000' < targetDB.mapping_OX_pref > targetDB.tsv_tmp

# Cleanup: taxon format:  "NCBI_TaxID=418404 {ECO:0000313|EMBL:AHX25609.1};"
# Only the numerical identifier "418404" is required.
awk '{match($2, /=([^ ;]+)/, a); print $1"\t"a[1]; }' targetDB.tsv_tmp > targetDB.tsv
```

The convertkb module extracts either all or the chosen UniProt Knowledgebase line types into separate databases, which are indexed by their UniProt accession. By providing a tab separated mapping file between target database identifiers and UniProt accessions, a database of UniProt Knowledgebase entries, indexed by their target database identifiers, can be created. This database is then transformed into a tsv file.

Classification Once the prerequisites are generated, the taxonomy classification can be executed:

```
mmseqs taxonomy queryDB targetDB targetDB.tsv ncbi-taxdump queryLcaDB tmp
mmseqs createtsv queryDB queryLcaDB queryLca.tsv
```

Each line of the result file queryLca.tsv will contain a tab separated list of 1) query accession, 2) LCA NCBI taxon ID, 3) LCA rank name, and 4) LCA scientific name.

The --lca-ranks parameter can be supplied with a colon (:) separated string of taxonomic ranks. For example, --lca-ranks genus:family:order:superkingdom will resolve the respective ranks of the LCA and return a colon concatenated string of taxa as the fifth column of the result file.

Description of Core Modules

For advanced users, it is possible to skip the workflows and execute the core modules for maximum flexibility. Especially for the sequence search it can be useful to adjust the prefiltering and alignment parameters according to the needs of the user. The detailed parameter lists for the modules is provided in section Detailed Parameter List.

MMseqs2 contains three core modules: prefiltering, alignment and clustering.

Computation of Prefiltering Scores using `mmseqs prefilter`

The prefiltering module computes an ungapped alignment score for all consecutive k-mer matches between all query sequences and all database sequences and returns the highest score per sequence.

If you want to *cluster* a database, or do an all-against-all search, the same database will be used on both the query and target side. the following program call does an all-against-all prefiltering:

```
$ mmseqs prefilter sequenceDB sequenceDB resultDB_pref
```

`sequenceDB` is the base name of the mmseqs databases produced from the FASTA sequence databases by `mmseqs createdb`, the prefiltering results are stored in the mmseqs database files `resultDB_pref` and`prefilterDB.index`.

For *sequence search* two different input databases are usually used: a query database `queryDB` and a target database `targetDB`, though they can again be identical. In this case, the prefiltering program call is:

```
$ mmseqs prefilter queryDB targetDB resultDB_pref
```

MMseqs2 can handle profiles or protein sequences as input for the `queryDB`.

The prefilter k-mer match stage is key to the high speed and sensitivity. It detects consecutive short words (“k-mer”) match on the same diagonal. The diagonal of a k-mer match is the difference between the positions of the two similar ‘’k”-mer 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-mer is generated (orange frame). The similarity threshold determines the list length and sets the trade-off between speed and sensitivity. The similiar k-mer list lenght can be controlled with `-s`.

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).

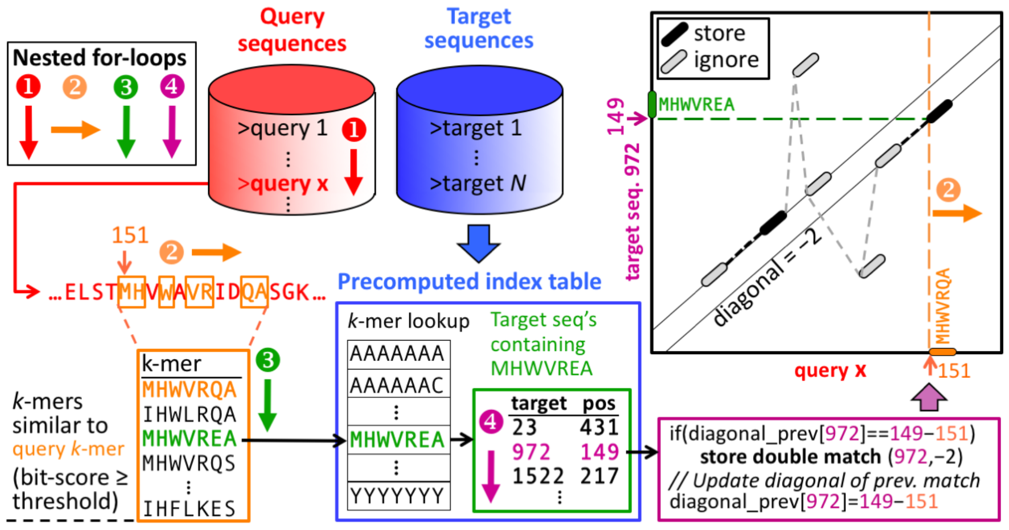


Fig. A.9. Prefilter

For each consecutive k-mer matches an ungapped alignment is computed. Only the maximal ungapped alignment score for each target is reported.

Set sensitivity `-s` parameter The sensitivity of the prefiltering can be set using the `-s` option. Internally, `-s` sets the average length of the lists of similar k-mers per query sequence position.

- *Similar k-mers list length*: Low sensitivity yields short similar k-mer lists. Therefore, the speed of the prefiltering increases, since only short k-mer lists have to be generated and less lookups in the index table are necessary. However, the sensitivity of the search decreases, since only very similar k-mers are generated and therefore, the prefiltering can not identify sequence pairs with low sequence identity.

It is possible to speed best hits searches by stepwise increasing `-s`. MMseqs2 includes a workflow for this purpose. How to find the best hit the fastest way

The following graphic shows the average AUC sensitivity versus speed-up factor relative to BLAST for 637,000 test searches. White numbers in plot symbols give number of search iterations.

It is furthermore possible to use change the k-mer lengths, which are used in the prefiltering. Longer k-mers are more sensitive, since they cause less chance matches. Though longer k-mers only pay off for larger databases, since more time is needed for the k-mer list generation, but less time for database matching. Therefore, the database matching should take most of the computation time, which is only the case for large databases. As default MMseqs try to compute the optimal k-mer length based on the target database size.

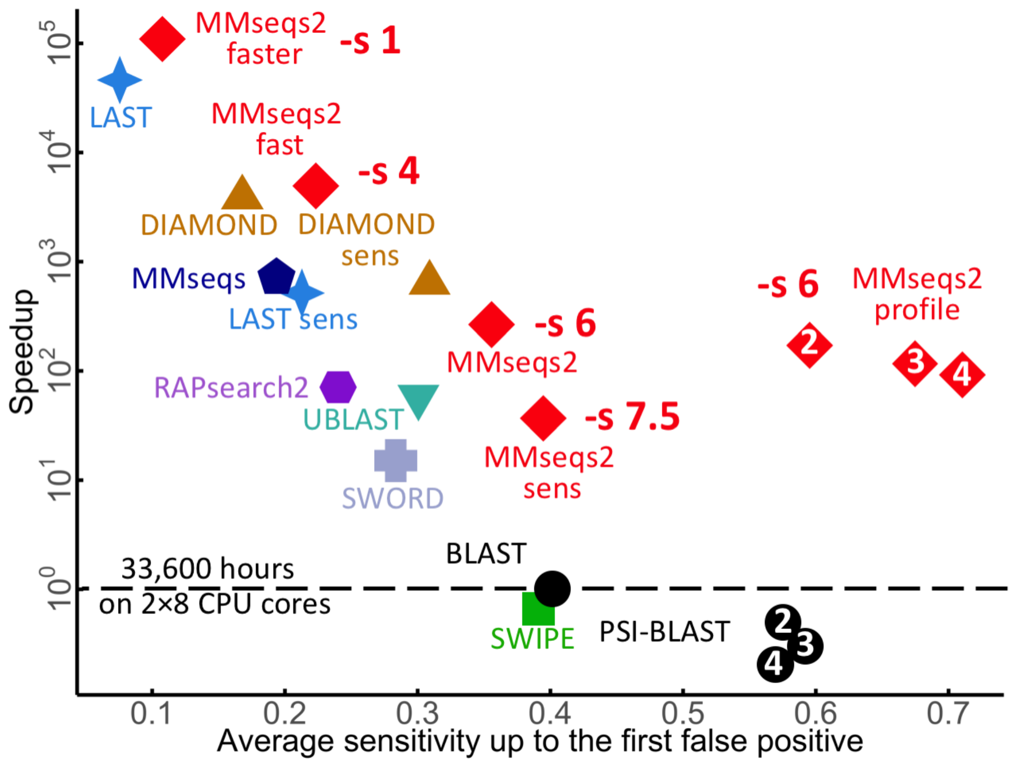


Fig. A.10. Prefilter sensitivity

Local alignment of prefiltering sequences using `mmseqs align`

In the alignment module, you can also specify either identical or different query and target databases. If you want to do a clustering in the next step, the query and target databases need to be identical:

```
$ mmseqs align sequenceDB sequenceDB resultDB_pref resultDB_aln
```

Alignment results are stored in the database files `resultDB_aln` and `resultDB_aln.index`.

Program call in case you want to do a sequence search and have different query and target databases:

```
$ mmseqs align queryDB targetDB resultDB_pref resultDB_aln
```

This module implements a SIMD accelerated Smith-Waterman-alignment (Farrar, 2007) of all sequences that pass a cut-off for the prefiltering score in the first module. It processes each sequence pair from the prefiltering results and aligns them in parallel, calculating one alignment per core at a single point of time. Additionally, the alignment calculation is vectorized using SIMD (single instruction multiple data) instructions. Eventually, the alignment module calculates alignment statistics such as sequence identity, alignment coverage and e-value of the alignment.

Clustering sequence database using `mmseqs clust`

For calling the stand-alone clustering, you need the input sequence database and a result database:

```
$ mmseqs cluster sequenceDB resultsDB_aln resultsDB_clu
```

Clustering results are stored in the MMseqs database files `resultsDB_clu` and the index `resultsDB_clu.index`. The clustering module offers the possibility to run three different clustering algorithms by altering the `--cluster-mode` parameter. A greedy set cover algorithm is the default (`--cluster-mode 0`). It tries to cover the database by as few clusters as possible. At each step, it forms a cluster containing the representative sequence with the most alignments above the special or default thresholds with other sequences of the database and these matched sequences. Then, the sequences contained in the cluster are removed and the next representative sequence is chosen.

The second clustering algorithm is a greedy clustering algorithm (`--cluster-mode 2`), as used in CD-HIT. It sorts sequences by length and in each step forms a cluster containing the longest sequence and sequences that it matches. Then, these sequences are removed and the next cluster is chosen from the remaining sequences.

The third clustering algorithm is the connected component algorithm. This algorithm uses the transitivity of the relations to form larger clusters with more remote homologies. This algorithm adds all proteins to a cluster, that are reachable in a breadth first search starting at the representative with the most connections.

Output File Formats

MMseqs Database Format

Most MMseqs2 commands use the MMseqs database format. The format is inspired by ffindex, which was developed by Andreas Hauser. It avoids drastically slowing down the file system when millions of files would need to be written or accessed, e.g. one file per query sequence in a many-to-many sequence search. MMseqs databases hide these files from the file system by storing them in a single data file. The *data file* `<name>` contains the data records, i.e. the contents of the file, concatenated and separated by \0 characters. A second, *index file* `<name>.index` contains for each numerical identifies (corresponding to the file name) the position of the corresponding data record in the data file.

Each line of the *index file* contains, separated by tabs, (1) the ID, (2) the offset in bytes of the data_record counted from the start of the data file, and (3) the size of the data record. The IDs have to be sorted numerically in ascending order, since for accessing a data record by IDs the matching IDs are found by binary search.

Here is an example for a database containing four sequences:

```
PSSLDIRL\0GTLKRLSAHYTPAW\0AEAIFIHEG\0YTHGAGFDNDI\0
```

The corresponding index file (file extension `.index`) could look like this.

```
10  0   9
11  9  15
12  24  10
13  34  12
```

The index contains four IDs, one for each data record: 10, 11, 12 and 13. The corresponding data records have offset positions 0, 9, 25, 35 and the data record sizes are 9, 15, 10, and 12 respectively.

The MMseqs2 modules `createdb` and `createfasta` do the format conversion from fasta to the MMseqs database format. `createdb` generates an MMseqs database from a FASTA sequence database. It assigns each sequence in the FASTA file sequentially a numerical id. Sequences that are longer than 32768 letters are split. `createfasta` converts an MMseqs database to a FASTA formatted text file: the sequence headers contain the DB identifiers preceded by >, and the sequence is extracted from the corresponding data record of the DB's data file.

However, for fast access in very large databases it is advisable to use the MMseqs database directly without converting it to FASTA format. We provide several tools at `http://github.com/soedinglab/ffindex_soedinglab/` (query, build and apply function on each entry) to work with MMseqs databases. The binary `ffindex_get` can be used to directly access single records stored in an MMseqs database.

Prefiltering format

Each data record consists of the prefilter results for one query sequence. The ID is the database accession code, a numerical identifier (ID) for the query that was sequentially assigned by `createdb`.

Each line in a data record reports on one matched database sequence and has the following format (white space = '\t'):

```
targetID  -log(E-value)  diagonal
```

where `targetID` is the database identifier of the matched sequence, `-log(E-value)` is the ungapped negative logarithmic E-value of the match, and `diagonal` is the diagonal i-j (i = position in query, j = position in db sequence) on which the match occurs.

Example of a database record for prefiltering:

```
2       71       0
3       35       0
5       -2       8
```

The first line describes a match with database sequence 2 on `diagonal` 0 with a `-log(e-value)` of 71 (e-value 1.46e-31).

Alignment format

Each data record consists of the alignment results for one query sequence. The ID if the queries was sequentially assigned by `createdb`.

Each line in a data record reports on match, i.e., one database sequence aligned to the query. It has the following format (white space = ’ ̂’)

```
targetID  alnScore  seqIdentity  eVal  qStart  qEnd  qLen  tStart  tEnd  tLen  [alnCigar]
```

Here, `targetID` is the database identifier of the matched sequence, `alnScore` is the bit score of the alignment in half bits, `seqIdentity` is the sequence identity [0:1], `eVal` is the e-value of the match, `qStart` is the start position of the alignment in the query, `qEnd` is the end position of the alignment in the query, `tStart` and `tEnd` are the start and end positions in the target (i.e. the database sequence), tLen`is the target sequence length, the optional`alnCigar`string encodes the alignment in compressed format and is only included in the results if option`-a‘ was used in mmseqs2 search. The numbers preceding the three letters M, I, and D give the number of match positions in a block aligned without gaps, the number of insertions and of deletions, respectively.

Example data record for alignment results:

```
2 705 1.000 8.771e-207 0  372 373 0  372 373 373M
5 367 0.595 3.319e-105 29 372 373 21 364 369 52M3I126M3D163M
3 347 0.565 2.722e-99  13 367 373 20 367 373 10M5I53M3I118M1D166M
```

The first line with targetID 2 is an identity match. The last sequence 3 has a Smith-Waterman alignment score of 347, the sequence identity 0.565 and the e-value 2.722e-99, the query start and end position is 13,367 of the total length 373, the target start and end position is 20,367 of the total length 373, the alignment string is 10M5I53M3I118M1D166M.

Clustering format

Internal cluster format Each data record consists of the IDs of the members of one cluster. The ID refers to the representative sequence of that cluster, (usually assigned by `createdb`).

Each line in a data record contains one ID of a cluster member. The first line of each data record contains the ID of the representative sequence of that cluster.

Here is an example of a cluster record with 3 cluster members:

```
2
5
3
```

The 2 is the ID of the representatives sequence while 5 and 3 are further cluster members.

Cluster TSV format The internal format can be converted to a flat tsv file:

```
$ mmseqs createtsv sequenceDB sequenceDB resultsDB_clu resultsDB_clu.tsv
```

The `resultsDB_clu.tsv` file follows the following format:

```
#cluster-representative     cluster-member
ID1 ID1
ID1 ID25
ID1 ID32
ID1 ID10
ID4 ID4
ID4 ID534
```

All members of the clustering are listed line by line. The first column always contains the representative sequence, the second contains the cluster member. For the example the cluster with the representative sequence ID1 contains four members it self and ID25, ID32, ID10. ID are parsed from the header from the input database (see id parsing from headers).

Cluster Fasta like format The internal format can be converted to a fasta a like format:

```
mmseqs createseqfiledb DB clu clu_seq
mmseqs result2flat DB DB clu_seq clu_seq.fasta
```

The resulting fasta a like format file will look like this:

```
>ID1
>ID1
MAGA....R
>ID25
MVGA....R
>ID32
MVGA....R
>ID10
MVGV....R
>ID4
>ID4
MCAT...Q
>ID534
MCAR...Q
```

A new cluster is marked by two identical name lines of the representative sequence, where the first line stands for the cluster and the second is the name line of the first cluster sequence. It is followed by the fasta formatted sequences of all its members.

Extract representative sequence To extract the representative of a clustering use the following commands:

```
mmseqs result2repseq DB clu clu_rep
mmseqs result2flat DB DB clu_rep clu_rep.fasta --use-fasta-header
```

The resulting fasta will contain all representative sequences:

```
>ID1
MAGA....R
>ID4
MCAT...Q
```

Identifier parsing

MMseqs2 parses identifier from the fasta header when transforming a result DB into a flat file by using e.g. `createtsv`, `convertalis`, ...). We support following fasta header types:

```
    Uniclust,
    Swiss-Prot,
    Trembl,
    GenBank,
    NCBI Reference Sequence,
    Brookhaven Protein Data Bank,
    GenInfo Backbone Id,
    Local Sequence identifier,
    NBRF PIR,
    Protein Research Foundation,
    General database identifier,
    Patents,
    NCBI GI
```

If none of the header supported could be detected than we extract everything from header start (excluding >) until the first whitespace.

Optimizing Sensitivity and Consumption of Resources

This section discusses how to keep the run time, memory and disk space consumption of MMseqs2 at reasonable values, while obtaining results with the highest possible sensitivity. These considerations are relevant if the size of your database exceeds several millions of sequences and are most important if the database size is in the order of tens of millions of sequences.

Prefiltering module

The prefiltering module can use a lot of resources (memory consumption, total runtime and disk space), if the parameters are not set appropriately.

Memory Consumption For maximum efficiency of the prefiltering, the entire database should be held in RAM. The major part of memory is required for the k-mer index table of the database. For a database containing `N` sequences with an average length `L`, the memory consumption of the index lists is `(N * L * 7) byte`. Note that the memory consumption grows linearly with the size of the sequence database. In addition, the index table stores the pointer array and two auxiliary arrays with the memory consumption of `a^k*8` byte, where `a` is the size of the amino acid alphabet (default a=20, does not include the unknown amino acid X) and k is the k-mer size. The overall memory consumption of the index table is

```
M = (7 * N * L + 8 a^k) byte
```

Therefore, the UniProtKB database version of April 2014 containing 55 million sequences with an average length 350 needs about 71 GB of main memory.

MMseqs2 will automatically split the target database if the computer has not enough main memory.

Runtime The prefiltering module is the most time consuming step. It can scale from minutes in runtime to days by adjusting the sensitivity setting. Searching with 637000 protein sequences against 30 Mio Uniprot seqeunces took around 12 minutes on a 16 cores.

Disk Space The prefiltering results for very large databases can grow to considerable sizes (in the order of TB) of the disk space if very long result lists are allowed and no strict ungapped score threshold is set. As an example, an all-against-all prefiltering run on the 25 Mio seqeunces with `--max-seqs` 300 yielded prefiltering list with an average length of 150 and an output file size of 78 GB. One entry needs roughly 21 byte of space. To compute the worse case hard disk space usage `S` use the following formular. `N` is the Database sequence size L is `--max-seqs`.

```
S = (21 * N * L) byte
```

Important Options for Tuning the Memory, Runtime and Disc Space Usage

- The option `-s` controls the sensitivity in the MMseqs2 prefiltering module. The lower the sensitivity, the faster the prefiltering becomes, though at the cost of search sensitivity. See Set sensitivity -s parameter.
- The option `--max-seqs`controls the maximum number of prefiltering results per query sequence. For very large databases (tens of millions of sequences), it is a good advice to keep this number at reasonable values (i.e. the default value 300). For considerably larger values of `--max-seqs`, the size of the output can be in the range of several TB of disk space for databases containing tens of millions of sequences. Changing `--max-seqs` option has no effect on the run time but can degrade the sensitivity.

Alignment Module

In the alignment module, generally only the total runtime and disk space are the critical issues.

Memory Consumption The major part of the memory is required for the three dynamic programming matrices, once per core. Since most sequences are quite short, the memory requirements of the alignment module for a typical database are in the order of a few GB.

Runtime The alignment is based on a striped vectorized algorithm which can process roughly 2 giga cell updates per second (GCUPS). The time to compute the alignment of two average sized proteins (350 residues) takes roughly 6.0625E-5 seconds on one CPU. For example computing 23 Mio. alignments on 8 cores takes 2 minutes.

If a huge amount of alignments have to be calculated, the run time of the alignment module can become a bottleneck. The run time of the alignment module depends essentially on two parameters:

- The option `--max-seqs` controls the maximum number of sequences aligned with a query sequence. By setting this parameter to a lower value, you accelerate the program, but you may also lose some meaningful results. Since the prefiltering results are always ordered by their significance, the most significant prefiltering results are always aligned first in the alignment module.

- The option `--max-accept` controls the maximum number of alignment results per query sequence.

- The option `--max-rejected` defines the maximum number of rejected sequences for a query until the calculation of alignments stops. A reject is an alignment whose statistics don't satisfy the search criteria such as coverage threshold, e-value threshold etc. Per default, `--max-rejected` is set to `INT_MAX`, i.e. all alignments until `--max-seqs` alignments are calculated.

Disk Space Since the alignment module takes the results of the prefiltering module as input, the size of the prefiltering module output is the point of reference. If alignments are calculated and written for all the prefiltering results, the disk space consumption is 1.75 times higher than the prefiltering output size.

Clustering Module

In the clustering module, only the memory consumption is a critical issue.

Memory Consumption The clustering module can need large amounts of memory. The memory consumption for a database containing `N` sequences and an average of `r` alignment results per sequence can be estimated as

```
M = (6 * N * r) byte
```

To prevent excessive memory usage for the clustering of large databases, you should use cascaded clustering (`--cascaded` option) which accumulates sequences per cluster incrementally, therefore avoiding excessive memory use.

If you run the clustering module separately, you can tune the following parameters:

- `--max-seqs` parameter which controls the maximum number of alignment results per query considered (i.e. the number of edges per node in the graph). Lower value causes lower memory usage and faster run times.
- Alternatively, `-s` parameter can be set to a higher value in order to cluster the database down to higher sequence identities. Only the alignment results above the sequence identity threshold are imported and it results in lower memory usage.

Runtime Clustering is the fastest step. It needs less than an hour for the clustering of the whole UniProtKB.

Disk Space Since only one record is written per cluster, the memory usage is a small fraction of the memory usage in the prefiltering and alignment modules.

Workflows

The search and clustering workflows offer the possibility to set the sensitivity option `-s` and the maximum sequences per query option `--max-seqs`. `--max-rejected` option is set to `INT_MAX` per default. Cascaded clustering sets all the options controlling the size of the output, speed and memory consumption, internally adjusting parameters in each cascaded clustering step.

How to run MMseqs2 on multiple servers using MPI

MMseqs2 can run on multiple cores and servers using OpenMP (OMP) and message passing interface (MPI). MPI assigns database splits to each servers and each server computes them using multiple cores (OMP). Currently `prefilter`, `align`, `result2profile`, `swapresults` can take advantage of MPI. To parallelize the time-consuming k-mer matching and gapless alignment stages prefilter 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. The number of chunks is controlled through the `--split` parameter. 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 * N * L) / #chunks + 21^k * 8) byte
```

Thus, it 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 (assume that all machines have the same amount of memory). Make sure that MMseqs2 was compiled with MPI by using the `HAVE_MPI=1` flag (`cmake -DHAVE_MPI=1 -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`). Our precomplied static version of MMseqs2 can not use MPI. To search with multiple server just call the search and add the RUNNER variable. The TMP folder has to be shared between all nodes (e.g. NFS)

```
  RUNNER="mpirun -np 42" mmseqs search queryDB targetDB resultDB tmp
```

For clustering just call the clustering. The TMP folder has to be shared between all nodes (e.g. NFS)

```
  RUNNER="mpirun -np 42" mmseqs cluster DB clu tmp
```

Frequently Asked Questions

This section describes common questions.

How to set the right alignment coverage to cluster

MMseqs has three modes to control the coverage.

(1) With `--cov-mode 0 -c [0.0,1.0]` only sequences are clustered that have a sequence length overlap greater than X% of the longer of the two sequences. This coverage mode should be used to cluster full length protein sequences. The multi domain structure of proteins will be most likely preserved when using a coverage > 80% (`-c 0.8`).

For example:

```
 q: MAVGTACRPA
 t: -AVGTAC---
```

The coverage of would be 6/10=60%

```
 q: -AVGTAC---
 t: MAVGTACRPA
```

The coverage of would be 6/10=60%

(2) With `--cov-mode 1 -c [0.0,1.0]` (target-cov mode) only sequences are clustered that have a sequence length overlap greater than X% of the target sequence. The target cov mode can be used to cluster protein fragments. To suppress fragments from becoming representative sequences, it is recommended to use `--cluster-mode 2` in conjunction with `--cov-mode 1`.

For example:

```
q: MAVGTACRPA
t: -AVGTAC---
```

The target coverage would be 6/6=100%

```
q: -AVGTAC---
t: MAVGTACRPA
```

The target coverage would be 6/10=60%

(3) With `--cov-mode 2 -c [0.0,1.0]` (query-cov mode) only sequences are clustered that have a sequence length overlap greater than X% of the query sequence. The query cov mode can be used while searching e.g. to assure a certain level of coverage.

For example:

```
q: MAVGTACRPA
t: -AVGTAC---
```

The query coverage would be 6/10=60%

```
q: -AVGTAC---
t: MAVGTACRPA
```

The query coverage would be 6/6=100%

How is MMseqs computing the sequence identity

There are two ways MMseqs can compute the sequence identity, both of which produce similar results:

(1) When using `--alignment-mode 3` mmseqs2 will compute the number of identical aligned residues divided by the number of aligned columns including columns containing a gap in either sequence.

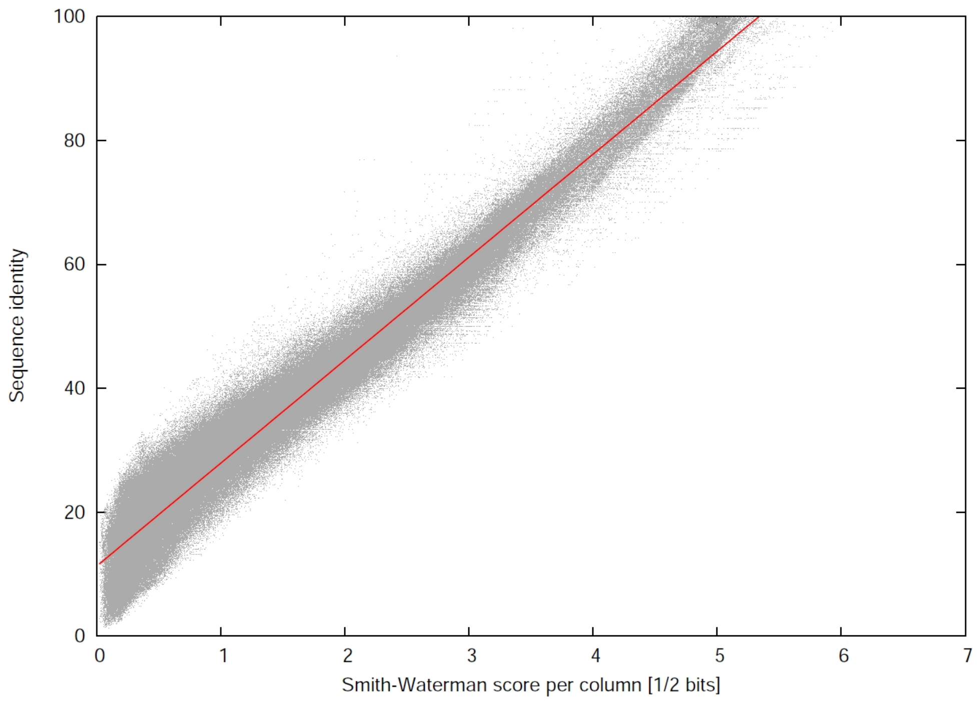


Fig. A.11. Relationship between score per column and sequence identity

(2) By default, the sequence identity is estimated from the score per column, i.e., the local alignment bit score divided by the maximum length of the two aligned sequence segments. The estimate uses the linear regression function (shown in red below) between the sequence identity computed as in (1) and the score per column in the scatter plot:

The score per column is a better measure of the degree of similarity than the actual sequence identity, because it also takes the degree of similarity between aligned amino acids and the number and length of gaps into account.

How to restart a search or clustering workflow

MMseqs checks if files are already computed in the `tmpDir` and skips already computed results. To restart delete temporary result files from the crashing step that were created by MMseqs and restart the workflow with the same program call again. You can recognise the temporary files that should be deleted by their file ending `.[0-9]+`.

If the job crashed while merging files they can be merged manually using ffindex_build (https://github.com/soedinglab/ffindex_soedinglab). For example, if the merge step of the alignment fails while using 56 threads then the result could be recovered by using the following command.

```
for i in $(seq 0 55); do ffindex_build -a aln{,.tmp.index} -d aln.$i  -i aln.index.$i ; done
LC_ALL=C sort --parallel 28 -n -k 1,1 aln.tmp.index > aln.index
```

How to find the best hit the fastest way

MMseqs2 can apply an iterative approach to speed up best-hit-searches. It will start searching with the lowest sensitivity defined with `--start-sens` and search until the target sensitivity `-s` is reached. The amount of steps to reach `-s` can be defined with `--sens-steps`.

Queries are only used again in the next iteration, if no match could be found that fulfilled the acceptance criteria in the previous iteration.

For example, the following search performs three search steps with sensitivity `-s` 1, 4 and 7.

```
mmseqs search qDB tDB rDB tmp --start-sens 1 --sens-steps 3 -s 7 --max-accept 1
```

Using this iterative approach can speed up best-hit-searches 4-10 times.

How is MMseqs handling low complexity

MMseqs uses reduces low complexity effects on the query and target database.

Query sequences are handled by an amino acid local compositional bias correction. In prefilter and alignment stages we apply a correction to substitution matrix scores assigning lower scores to the matches of amino acids that are overrepresented in the local sequence neighborhood. To switch the compositional bias correction on and off use `--comp-bias-corr`.

Target sequences low-complexity regions are masked during the prefilter stage. We use TANTAN with a threshold of 90% probability for low complexity. Masking can be controlled with `--mask`.

How to redundancy filter sequences with identical length and 100% length overlap.

To redundancy filter sequences of identical length and 100% overlap `mmseqs clusthash` can be used. It reduces each sequence to a five-letter alphabet, computes a 64 bit CRC32 hash value for the full-length sequences, and places sequences with identical hash code that satisfy the sequence identity threshold into the same cluster.

Example: cluster sequences at 90% sequence identity

```
mmseqs clusthash sequenceDB resultDB --min-seq-id 0.9
mmseqs clust sequenceDB resultDB clusterDB
```

How to add sequence identities and other alignment information to a clustering result.

We can add sequence identities and other alignment information to the clustering result `outDB` by running an additional align step:

```
$ mmseqs align sequenceDB sequenceDB resultDB alignDB -a
$ mmseqs createtsv sequenceDB sequenceDB alignDB align.tsv
```

The `-a` parameter computes the whole backtrace. `--alignment-mode 3` could be used instead if the backtrace is not needed. This would save disk space. The backtrace is however computed anyway (for the calculation of the sequence identities) and then discarded.

How to run external tools for each database entry

The`apply` module can be used to call an external tool for each entry of a MMseqs2 database. It works like the map step from the map/reduce pattern. It calls for every index entry the specified process with the passed parameters. The process reads the entry data from stdin and its stdout is written to a new entry in the result database. The tool supports OpenMP and MPI parallelization for spreading out the job over several compute nodes.

Example: An awk script which takes an alignment result entry from stdin and prints out all lines with an e-value <0.001 to stdout (Hint: the `filterdb` module can also solve this problem, but with less overhead):

```
mmseqs apply resultDB filteredResultDB -- awk '$4 < 0.001 { print; }'
```

The `apply` module exports the `MMSEQS_ENTRY_NAME` environment variable into the called processes. It contains the current database key.

How to manually cascade cluster

It is possible to cluster the representative sequences of an clustering run and merge the cluDB results with the following workflow.

```
# first clustering run
mmseqs linclust sequenceDB clu1 tmp1
# create a subset of the sequenceDB only with representative sequences
mmseqs createsubdb clu1 sequenceDB cluSequenceDB
# cluster representative sequences
mmseqs cluster "cluSequenceDB" clu2 tmp2
# merge two clusterings in to one results
mmseqs mergecluster sequenceDB final_clu clu1 clu2
```

How to create a HHblits database

One can turn the output of a search (or clustering) into a HHblits database. You need to have HH-suite properly installed with MPI support. The following procedure creates an HHblits-compatible database “searchMsa” resulting from the enrichment of sequences of “DBquery” with the sequences of “DBtarget”:

```
mmseqs search DBquery DBtarget searchOut tmp -a
mmseqs result2msa DBquery DBtarget searchOut searchMsa --compress
mpirun -np 2 cstranslate_mpi -i searchMsa -o searchMsa_cs219 -A /path/to/cs219.lib \
-D /path/to/context_data.lib -x 0.3 -c 4 -I ca3m
```

The files `/path/to/cs219.lib` and `/path/to/context_data.lib` are provided in the “data” subfolder of your HH-suite installation. The parameters `-x 0.3 -c 4` have been empirically found to perform well.

For creating an HHblits database from a clustering, the procedure is almost the same, except that you have to create symlinks to the ffindex _header and _sequence files needed by HHblits:

```
mmseqs cluster DB clu tmp
mmseqs result2msa DB DB clu cluMsa --compress
ln -s DB_h cluMsa_header.ffdata
ln -s DB_h.index cluMsa_header.ffindex
ln -s DB cluMsa_sequence.ffdata
ln -s DB.index cluMsa_sequence.ffindex
mpirun -np 2 cstranslate_mpi -i cluMsa -o cluMsa_cs219 -A /path/to/cs219.lib \
-D /path/to/context_data.lib -x 0.3 -c 4 -I ca3m
```

In the “search” case, those files are generated by MMseqs2, since it needs to merge the query and the target sequence databases. No merging is done for clustering, since both the query and target sequence database are the same.

Workflow Control Parameters

Search Workflow

Compares all sequences in the query database with all sequences in the target database.

Usage:

`mmseqs search <queryDB> <targetDB> <outDB> <tmpDir> [opts]`

Options:

`-s [float] Target sensitivity in the range [1:8.5] (default=4).`

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. 1.0 fastest - 8.5 sensitive. The sensitivty between 8 to 8.5 should be as sensitive as BLAST. For detailed explanation see section Computation of Prefiltering Scores using mmseqs prefilter.

Clustering Workflow

Calculates the clustering of the sequences in the input database.

Usage:

`mmseqs cluster <sequenceDB> <outDB> <tmpDir> [opts]`

Options:

`--cascaded Start the cascaded instead of simple clustering workflow.`

The database is clustered incrementally in three steps and improves the sensitivity of the clustering greatly compared to the general workflow. For detailed explanation, see the section Clustering sequence database using mmseqs clust.

`-s [float] Target sensitivity in the range [2:9] (default=4).`

Adjusts the sensitivity of the prefiltering and influences the prefiltering run time. For detailed explanation see section Computation of Prefiltering Scores using mmseqs prefilter.

`--min-seq-id [float] list matches above this sequence identity [0.0:1.0] (default=0.0).` Read more about how MMseqs is computing sequence identity in section How is MMseqs computing the sequence identity.

`--cov-mode [int] "0: coverage of query and target, 1: coverage of target [0:1] (default=0).` `-c [float] "list matches above this fraction of covered residues (see cov-mode) [0.0:1.0] (default=0.8).` Read more about coverage is computed at section How to set the right alignment coverage to cluster

Updating Workflow

Updates the existing clustering of the previous database version with new sequences from the current version of the same database.

Usage:

`mmseqs clusterupdate <oldDB> <newDB> <oldDB_clustering> <outDB> <tmpDir> [opts]`

Options:

`--sub-mat [file] Amino acid substitution matrix file.`

Substitution matrices for different sequence diversities in the required format can be found in the MMseqs2 `data` folder.

External Libraries used in MMseqs2

We would also like to thank the developers of the open source librarys used in MMseqs2:

- Striped Smith-Waterman Library
- ALP Library
- TANTAN
- Open MP Template Library
- kseq
- iota
- blast2lca

Developers Guide

Regression test

To run a search regression test execute the following steps:

```
# download the runner script run_codeship_pipeline.sh and set permissions
$ wget https://bitbucket.org/martin_steinegger/mmseqs-benchmark/run_codeship_pipeline.sh
$ chmod +x run_codeship_pipeline.sh

# change three varialbes in this file edit the following variables:
# If you dont have AVX2 on the machine just comment all lines containing MMSEQSAVX

BASE_DIR="$HOME/clone/regression_test"
MMSEQSSSE="$HOME/clone/build/src/mmseqs"
MMSEQSAVX="$HOME/clone/build/avx2/src/mmseqs"
```

```
# run script and set CI_COMMIT_ID to some non-empty string
# (in our CI system this is automatically set to the git commit).
$ CI_COMMIT_ID="TESTING" ./run_codeship_pipeline.sh

# The script will return an error code != 0 if there is a regression
# in sensitivity of MMseqs2. The error code can be checked with "echo $?".
$ [ $? -eq 1 ] && echo "Error"
```

It will print a report with sensitivity AUCs it achieved and then error out if it did not achieve the minimum AUCs. Currently 0.235 for normal sequence searches and 0.331 for profile searches.

You can also use our Docker images to run this benchmark:

```
cd mmseqs-folder
docker build -t mmseqs2 .
git clone https://bitbucket.org/martin_steinegger/mmseqs-benchmark.git
cd mmseqs-benchmark
docker build -t mmseqs-benchmark .
```

The regression test passed, if the second image exits cleanly.

Sanitizers

MMseqs2 can be built with ASan/MSan/UBSan/TSan support by specifying calling:

```
cmake -DHAVE_SANITIZER=1 -DCMAKE_BUILD_TYPE=ASan ...
```

Replace ASan with MSan, UBsan or TSan for the other sanitizers. CMake will error and abort if your compiler does not support the respective sanitizer.

License Terms

The software is made available under the terms of the GNU General Public License v3.0. Its contributors assume no responsibility for errors or omissions in the software.