The Journal of Open Source Software

Blini: lightweight nucleotide sequence search and dereplication

Amit Lavon [1]

1 University of California, Irvine, CA, USA

DOI: 10.21105/joss.09494

Software

- Review
- Repository
- Archive

Editor: Frederick Boehm

Reviewers:

- @snowformatics
- @telatin

Submitted: 25 June 2025
Published: 17 December 2025

Summary

Blini is a tool for quick lookup of nucleotide sequences in databases, and for quick dereplication of sequence collections. It is meant to help cleaning and characterizing large collections of sequences that would otherwise be too big to search with BLAST (Altschul et al., 1990) or too demanding for a local machine to process, for example with Sourmash (Brown & Irber, 2016) or with MMseqs (Steinegger & Söding, 2018). Blini is designed to be fast and have a small memory footprint, while allowing the user to tweak its resource consumption to improve matching resolution. Finally, Blini is delivered as a single runnable binary, with no need to install any additional software.

Statement of need

Metagenomes are collections of genetic material from various organisms, which are often not initially known. In modern metagenomics, researchers often generate hundreds of thousands of redundant metagenome-assembled genomes (MAGs) across different samples. In (Pasolli et al., 2019) for example, over 150,000 MAGs were dereplicated into about 5,000 species-level bins. Such clustering operations typically require many compute hours, which can grow quadratically with the size of the input, when using classical clustering algorithms.

Characterizing the taxonomic makeup of a sample or a MAG collection involves searching its contents in large databases in order to find which organism matches each nucleotide sequence. Assembled sequences can reach lengths of millions of bases, making alignment-based search methods too cumbersome. Such big queries are often outsourced to powerful cloud-based services such as BLAST (Altschul et al., 1990) or CZID (Simmonds et al., 2024).

In recent years, k-mer-based algorithms were introduced, which enabled efficient searching in large datasets on local machines. Mash distance (Ondov et al., 2016) introduced an alignment-free estimation formula for average nucleotide identity between sequences, making sequence comparison linear. Sourmash (Brown & Irber, 2016) uses fractional min-hashing in order to create small representations of large sequences, which allow for efficient searching and comparison. The LinClust clustering algorithm (Steinegger & Söding, 2018) uses k-mer matching to reduce the number of pairwise comparisons and achieve linear scaling with the size of the input.

This work, Blini, combines insights from Mash, Sourmash, and LinClust into a simple tool that can quickly cluster or look up big collections of sequences using estimated identity or containment, with tweakable estimation resolution (similar to Sourmash's *scale*).

Algorithm

Fingerprinting with k-mers

Blini uses constant-length subsequences (k-mers) to create fingerprints for sequences. It uses the fractional min-hashing technique, similarly to Sourmash (Brown & Irber, 2016). A sliding window of length k goes over the sequence and hashes each canonical k-mer. This collection of hashes is often called the sequence's *sketch*. The lower $1/s$ hashes are retained, for an input scale parameter s. A high s means fewer hashes used in downstream calculations, trading accuracy for better CPU and RAM performance. Once k-mer hashes are extracted, the sequence is discarded and only its sketch is used for downstream operations. These sketches can be saved to files and reused.

Similarity estimation

Blini uses Mash distance (Ondov et al., 2016) to estimate average nucleotide identity (ANI) between sequences. This formula translates the Jaccard similarity between two k-mer sets to an estimation of the ANI between the original sequences. For containment matching, the hashes of the query sequence are compared against their intersection with the hashes of the reference sequence.

Search

The first step of searching is indexing the reference dataset. After the reference sequences are fingerprinted, the 25% lowest hashes are used for indexing. The index is a mapping from hash value to a list of sequence identifiers of the reference sequences that had that hash in their fingerprints. The number 25% was chosen as a sweet spot between saving memory and retaining enough information for the search. As an optimization, hashes with a single reference sequence are kept in a separate 'singletons' map. Since in practice most of the index elements are singletons, this helps reduce RAM consumption and garbage collection times.

In the second stage, each query sequence is read and fingerprinted. The hash values are looked up in the index, and candidate reference sequences are fetched. Then, the query sequence is compared against each candidate sequence using Mash distance, and matches that pass the similarity threshold are reported.

Clustering

The clustering (dereplication) procedure follows a similar scheme to LinClust (Steinegger & Söding, 2018). Sequences are indexed and ordered from the longest to the shortest. Then, going by that order, each sequence is searched for using the search procedure. Matches that pass the similarity threshold are joined with the query sequence and are considered a cluster. These matches are then removed from the search loop's candidates. This clustering procedure does not produce inter-cluster distances for hierarchy generation.

Performance evaluation

Search - small

The search function was tested on RefSeq's viral reference (Pruitt et al., 2007). Blini was compared against Sourmash and MMseqs. 100 viral genomes were randomly selected for the test. The algorithms were then run on the 100 genomes as queries, and the original database as reference. Each algorithm was expected to match each genome to its source in the database. In a second run, random SNPs were introduced to 1% of the genomes' bases, and the same test was rerun. For each test, the number of matches with sequences other than the query's

source was also measured. The searches were run against an index of the reference dataset, created by each tool.

All three tools were able to match all 100 queries with their sources in the database (Figure 1a). The number of non-source matches was 824 and 712 in Blini, 865 and 660 in Sourmash, and 3143 and 3019 in MMseqs, in the raw and mutated datasets, respectively (Figure 1b).

Run-time was measured for searching the 100 sequences sequentially. Blini and MMseqs were executed once and searched for all the queries in one run, while Sourmash had to be executed once for each individual query. Each run was repeated five times and the average run-time is reported. Blini completed the run in 0.5 seconds, Sourmash completed the run in 126 seconds, and MMseqs completed the run in 151 seconds (Figure 1c). The times shown here do not include reference-preprocessing time.

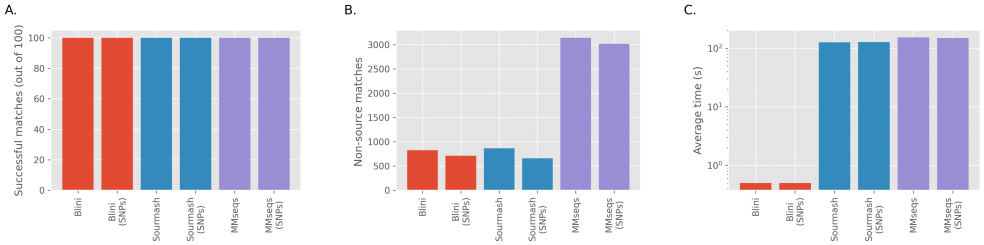

Figure 1: Search results for the viral dataset. Each tool was run on 100 randomly chosen viral genomes, to find them in the original dataset. (A) shows how many of the 100 genomes were correctly mapped to their source in the database. (B) shows how many additional matches were found in addition to the 100 chosen genomes. (C) shows the search times for the 100 queries together.

Search - big

To test the search function on a large dataset, the bacterial contigs from Pasolli et al. (2019) were used. This 10GB dataset contains 934K contigs from almost 5K bacterial species. Each of the compared tools was run to create an index of the dataset.

The simulated query dataset consisted of 100K random fragments of length 10K bases, sampled uniformly from the bacterial contigs. Each fragment was mutated with random SNPs in 0.1% of its bases. Blini, Sourmash and MMseqs were run on the query dataset, to search it in the bacterial reference. Because of the long search times, only Blini was run on the full set of queries, while the other tools were run on one or ten queries out of the 100K.

MMseqs took longer than 30 minutes to search for a single query, and was therefore terminated prematurely. Sourmash was run on one query and on ten queries and took 31 seconds per query. Blini took 6 seconds for one and ten queries, and 25 seconds for the entire set of 100K queries (Figure 2). This means a throughput of 5100 queries per second after the 6 seconds of loading the reference index. Blini matched all 100K queries with their correct source in the reference, with 2444 additional non-source matches (false-positives).

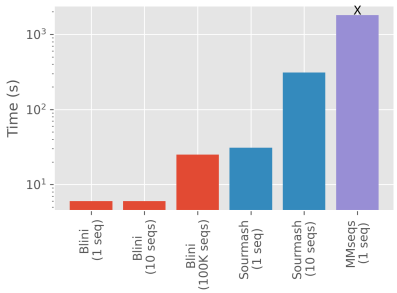

Figure 2: Search times for the bacterial dataset. Each tool was run on randomly chosen 10 kilobase fragments from a 10GB bacterial dataset. MMseqs is marked with an X because it was stopped manually before it could finish running.

Clustering

The clustering function was tested on two simulated datasets created from the 100 chosen genomes of the previous test. In one dataset each sequence had multiple counterparts with random SNPs. In the second dataset random fragments were extracted from each root sequence. In the SNPs dataset, each of the 100 original sequences had another 100 mutated counterparts. Each counterpart had random SNPs in 1% of its bases. In the fragments dataset, each of the 100 original sequences had 300 random fragments extracted from it, of length at least 1000 bases. The algorithms were expected to group each sequence with its mutated counterparts or with its fragments. Performance was evaluated using the Adjusted Rand-Index (ARI). Blini's *scale* refers to the fraction of k-mers considered for the operation. Scale 100 means that 1/100 of k-mers were used.

In the SNPs dataset, both Blini and MMseqs achieved an ARI between 0.999 and 1.0, except for Blini with scale 200 which achieved an ARI of 0.997 (Figure 3b). Blini created 100, 100, 101 and 110 clusters using scales 25, 50, 100 and 200 respectively. MMseqs created 103 clusters (Figure 3a). Blini took on average 10.5 seconds, and MMseqs took 46 seconds with one thread, and 14 seconds with four threads (Figure 3b). In terms of memory, Blini had a maximal memory footprint of 255, 129, 65, and 38 MB using scales 25, 50, 100 and 200 respectively. MMseqs had a maximal memory footprint of 3072 MB (Figure 3c).

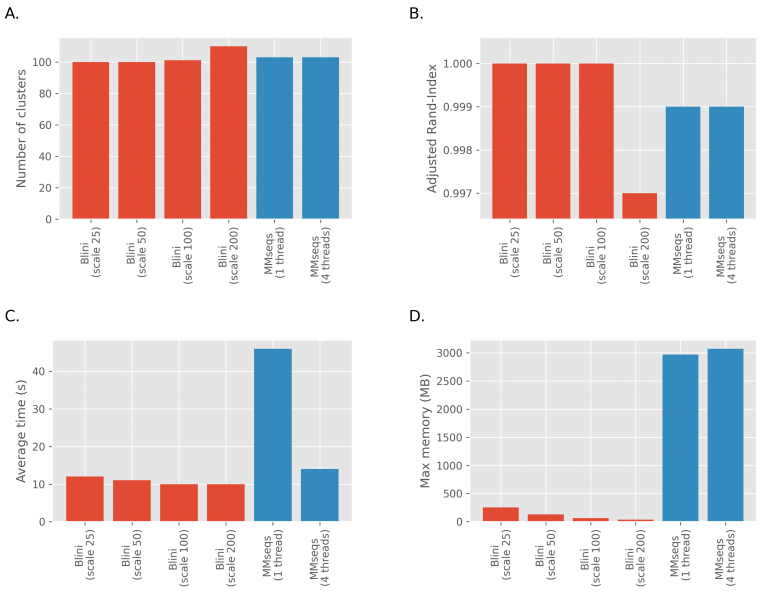

Figure 3: Clustering results for the SNPs dataset. Each of the 100 viral genomes from the search benchmark was used to create 100 mutant sequences with SNPs in 1% of their bases. The tools were run on this collection of 10,100 genomes and were expected to cluster them into 100 groups, corresponding to the original genomes.

In the fragments dataset, MMseqs achieved an ARI of 1.0 while Blini achieved an ARI of 0.999, 0.999, 0.998 and 0.989 with scales 25, 50, 100 and 200 (Figure 4b). Blini grouped the dataset into 100, 104, 135 and 386 clusters, while MMseqs grouped the dataset into 101 clusters (Figure 4a). The decline in performance with increasing scale is discussed below under Limitations. Blini took on average 20 seconds, and MMseqs took 80 seconds with one thread, and 24 seconds with four threads (Figure 4c). In terms of memory, Blini had a maximal memory footprint of 462, 233, 119, and 67 MB using scales 25, 50, 100 and 200 respectively. MMseqs had a maximal memory footprint of 5632 MB (Figure 4d).

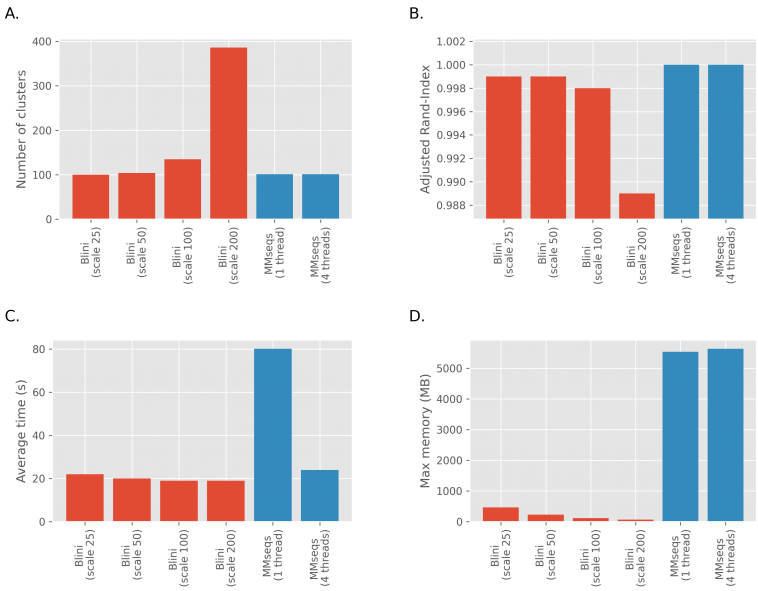

Figure 4: Clustering results for the fragments dataset. Each of the 100 viral genomes from the search benchmark was used to create 300 random fragments of length 1000 bases and above. The tools were run on this collection of 30100 genomes and were expected to cluster them into 100 groups, corresponding to the original genomes.

Limitations

The scale parameter controls how much information Blini can work with. While a higher scale reduces resource consumption, it also means that distances are calculated based on less information, which means a higher sampling error. For example, a 100-base sequence has fewer than 100 k-mers, which means fewer than 100 hashes. With the default scale of 100, it means on average less than one hash for a 100-base sequence. Using a binomial proportion error estimation $\sigma_{err} = \sqrt{\frac{J(J-1)}{|A_{sub} \cap B_{sub}|}}$, where J is the true Jaccard similarity and A_{sub} and B_{sub} are the subsamples of sets A and B, a subsample of at least 25 k-mers is required in order to reduce the Jaccard estimation error below 10%. Therefore, Blini is effective for sequences at least 25 times longer than the chosen scale value. For the default value of 100, sequences shorter than 2,500 bases are likely to be falsely missed. This can be seen in Figure 3, where clustering of sequences of length 1000+ bases was less accurate with a scale value of 200. While the scale can be tweaked, this tool might not be suitable for short reads.

Blini also currently only works for nucleotide sequences. Amino acid sequences might be added in the future. On the technical side, Blini is currently single-threaded. Multithreading can be considered in the future if a concrete need arises.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Brown, C. T., & Irber, L. (2016). Sourmash: A library for MinHash sketching of DNA. *Journal of Open Source Software*, *1*(5), 27. https://doi.org/10.21105/joss.00027

Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., &

Phillippy, A. M. (2016). Mash: Fast genome and metagenome distance estimation using MinHash. *Genome Biology*, *17*, 1–14. https://doi.org/10.1186/s13059-016-0997-x

Pasolli, E., Asnicar, F., Manara, S., Zolfo, M., Karcher, N., Armanini, F., Beghini, F., Manghi, P., Tett, A., Ghensi, P., & others. (2019). Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle. *Cell*, *176*(3), 649–662. https://doi.org/10.1016/j.cell.2019.01.001

Pruitt, K. D., Tatusova, T., & Maglott, D. R. (2007). NCBI reference sequences (RefSeq): A curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, *35*(suppl_1), D61–D65. https://doi.org/10.1093/nar/gkl842

Simmonds, S. E., Ly, L., Beaulaurier, J., Lim, R., Morse, T., Thakku, S. G., Rosario, K., Perez, J. C., Puschnik, A., Mwakibete, L., & others. (2024). CZ ID: A cloud-based, no-code platform enabling advanced long read metagenomic analysis. *bioRxiv*, 2024–2002. https://doi.org/10.1101/2024.02.29.579666

Steinegger, M., & Söding, J. (2018). Clustering huge protein sequence sets in linear time. *Nature Communications*, *9*(1), 2542. https://doi.org/10.1038/s41467-018-04964-5