GSearch: Ultra-Fast and Scalable Microbial Genome Search by Combining Kmer Hashing with Hierarchical Navigable Small World Graphs

Jianshu Zhao[1,2,7], Jean Pierre Both[3,7], Luis M. Rodriguez-R[4,5,6], Konstantinos T. Konstantinidis[1,2,4,*]

[1]Center for Bioinformatics and Computational Biology, Georgia Institute of Technology, Atlanta, Georgia, USA

[2]School of Biological Sciences, Georgia Institute of Technology, Atlanta, Georgia, USA

[3] Université Paris-Saclay, CEA, List, Palaiseau, France

[4]School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA

[5]Department of Microbiology, University of Innsbruck, Innsbruck, Austria

[6]Digital Science Center (DiSC), University of Innsbruck, Innsbruck, Austria

[7]Those authors contribute equally

[*]Corresponding author, Konstantinos T. Konstantinidis (kostas.konstantinidis@gatech.edu)

Abstract

Genome search and/or classification is a key step in microbiome studies and has recently become more challenging due to the increasing number of available (reference) genomes and the fact that traditional methods do not scale well with larger databases. By combining a kmer hashing-based genomic distance metric (ProbMinHash) with a graph based nearest neighbor search algorithm (called Hierarchical Navigable Small World Graphs, or HNSW), we developed a new program, GSearch, that is at least ten times faster than alternative tools due to O(log(N)) time complexity while maintaining high accuracy. GSearch can identify/classify 8,000 query genomes against all available microbial and viral species with sequenced genome representatives (n=~65,000) within several minutes on a personal laptop, using only ~6GB of memory. Further, GSearch can scale well with millions of database genomes based on a database splitting strategy. Therefore, GSearch solves a major bottleneck of microbiome studies that require genome search and/or classification.

Keywords: genome search, microbial genomes, MAGs, MinHash, nearest neighbor search, classification, hierarchical small world graphs, HNSW

Introduction

Identifying or classifying microbial species based on either universal marker genes (e.g., 16S or 18S rRNA genes) or entire genomes represents a re-occurring task in environmental and clinical microbiome studies. However, this task is challenging because i) whether or not microbes (bacteria, fungi) and viruses form discrete population clusters (or species), remains an open question [1, 2], and ii) the microbial species in nature are still severely under-sampled by the available genomes. For instance, there are more than 10^{12} prokaryotic and fungal species in nature according to a recent estimation based on 16S rRNA gene or ITS (Internal Transcribed Spacer) analysis [3] and even more viral species (e.g. the number of viral cells outnumbers that of prokaryotic cells by a about a factor of ten in most natural habitats) [4]. Yet, only ~17,000 bacterial species have been described and even fewer (around 15,000) are represented by complete or draft genome [5]. Due to the recent improvements in DNA sequencing and single-cell technologies, metagenomic surveys can now recover hundreds, if not thousands, of these yet-to-be-described species from environmental or clinical samples [6, 7], filling in the gap in the described diversity mentioned above. This has created a new challenge, however; that is, identifying these new genomes against the exponentially increasing number of available (described) genomes has become computationally intractable. Nonetheless, the recent high-throughput sequencing of isolate genomes as well as metagenomic studies of natural populations have shown that species may exist and be commonly circumscribed based on a 95% genome-aggregate average nucleotide identity (ANI) threshold, at least for prokaryotes and viruses [8, 9]. This threshold represents convenient means in searching and identifying new genomes against the already descried species and determining whether or not they represent novel species [10].

The number of curated draft or complete prokaryotic genomes has reached 317,542 in the newest release of the GTDB database, and 2,332,702 in the latest IMG/VR database for viruses, representing 65,703 prokaryotic and 935,122 viral distinct species at the 95% ANI level [11, 12]. Searching of query genomes against these large databases to find closely-related database/reference genomes for taxonomy classification based on the traditional brute-force methods, meaning, performing all vs. all searches, has become impractical, even for fast searching algorithms and/or small-to-medium computer clusters. For this task, faster search strategies are necessary. In addition to the searching strategy, the actual algorithm used to determine overall genetic relatedness between the query and the databased genomes is critical. While the traditional blast-based ANI among closely related genomes at the species level, and the genome-aggregate average amino acid identity (AAI) for genomes related at the genus level or above, have been proven to be highly accurate for genetic relatedness estimation across microbial and viral genomes [13-15], they are too slow to use when dealing with more than a few dozen of genomes. Faster implementations based on k-mer counting have been recently described to alleviate this bottleneck such as FastANI and MASH [16, 17], but these methods still do not scale with an increasing number of database (or query) genomes, especially based on an all vs. all search strategy. Further, defining genetic distance (or relatedness) based on kmer profiles can be problematic for incomplete genomes, which are commonly recovered from metagenomic surveys, and/or genomes with extensive repeats such as those found in several microbial eukaryotic genomes. Kmer-weighted approaches are advantageous in the latter cases because repeated genomic fragments can be considered when hashing but they have not been widely adopted yet [18, 19]. Recently, a phylogeny-based approach using a handful of universal genes (n ≈ 100) was developed to accelerate genome classification [20]. However, phylogenetic replacement based on a concatenated universal gene tree can be memory demanding and slow, especially for a large number of or a few deep-branching (novel) query genomes, and this approach cannot be applied to viral genomes, which lack universal genes. Further, universal genes due to their essentiality, are typically under stronger purifying selection and thus, evolve slower than the genome average. This property makes universal genes appropriate for comparisons among distantly related genomes, e.g., to classify genomes belonging to a new class or a new phylum, but not the species and genus levels [20, 21].

One of the most generally used approaches for finding closely related information to a query, while circumventing an all vs. all search, is the K-Nearest Neighbor Search (K-NNS). The K-NNS approach has been used for 16S rRNA gene-based classification followed by a vote strategy [22, 23] and, more recently, for whole genome and metagenome comparisons based on shared kmers [16]. Approximate nearest neighbor search (ANN) algorithms, such as locality-sensitive hashing (LSH) [24, 25] , k-dimension tree [26], random projection trees [27], k-graph [28] and proximity graph [29, 30] have been recently used to accelerate search processes. Proximity graph, as implemented for example in the hierarchical navigable small world graph (HNSW) [31], has been shown to be one of the fastest ANN search algorithms [32]. HNSW incrementally builds a multi-layer structure consisting of a hierarchical set of proximity graphs (layers) for nested subsets of the stored elements. Then, through smart neighbor selection heuristics, inserting and searching the query elements in the proximity graphs can be very fast while preserving high accuracy, even for highly clustered data [29, 31]. Therefore, finding the closest genomes in a database can be substantially accelerated by using HNSW.

Here, we describe GSearch (for Genome Search), a tool that combines one of the most efficient nearest neighbor search approaches (HNSW) with a universal approach to measure genetic relatedness among any microbial genome, including viral genomes, ProbMinHash [33], implemented in the Rust language for higher speed. ProbMinHash is based on shared kmers, weighted by their abundance and normalized by total kmer count, which can account for genome incompleteness of prokaryotic genomes and repeats commonly found in eukaryotic and sometimes in prokaryotic genomes. Essentially, ProbMinHash computes the normalized weighted Jaccard distance between each pair of genomes and subsequently, the weighted Jaccard distance normalized by total kmer count is used as input to build HNSW to create the graph of the database genomes. Accordingly, the search of the query genome(s) against the graph to find the nearest neighbors for classification purposes becomes an ultra-fast step using GSearch and can be universally applied to all microbial genomes. The novelty of GSearch also includes a hierarchical pipeline that involves both nucleotide-level (when query genomes have close relatives at the species level) and amino-acid-level searching (when query genomes represent novel species), which provides robust classification for query genomes regardless of their degree of novelty relative to the database genomes, as well as a database-splitting strategy that allows GSearch to scale up well to millions of database genome sequences.

Results

Probminhash as a robust metric of genome relatedness for prokaryotic genomes

Correlations between ProbMASH distance (we called it ProbMASH after transformation from ProbMinHash distance, see Methods & Materials) and ANI (determined by FastANI) or MASH distance showed that ProbMinHash is robust and slightly better than MASH for determining distances among bacterial genomes related at ~78% ANI, or higher, i.e., genomes assigned to the same or closely-related species (Spearman rho=0.9643 and 0.9640 of ProbMinHash and MASH values against corresponding ANI values for the same genome comparisons, respectively, *P*<0.001, Figure S1a and S1b; note that for finding best matches using the ANI approach as the reference, Spearman rank correlation is more relevant than Pearson correlation). For moderately related genomes, for which nucleotide-level ANI is known to lose accuracy, ProbMinHash was still robust compared to MASH for bacterial genomes (using the best matches found by average amino acid distance or AAI as the reference), especially among genomes showing between ~52% and 95% AAI (Spearman rho=0.90, *P*<0.01, Supplementary Figure S2a and S2b). Below ~50% AAI, both ProbMinHash and MASH distance lose accuracy compared to AAI. However, AAI of just universal genes provides a robust measurement of genetic relatedness at this level of distantly related genomes [21], and we show here that ProbMinHash distance for the same set of universal genes is also robust (Spearman rho=0.9390, *P*<0.001, Supplementary Figure S3). Thus, for query genomes of organisms with only distant relatives in the database (i.e., deep-branching), for which their closest represented genome in the database is related at the order level or higher, restricting the search to the universal genes can provide robust classifications.

Graph building and search against reference prokaryotic genomes is faster than alternative methods

To build the database graph for the entire GTDB v207 database (65,703 unique, non-redundant, at the species level, prokaryotic genomes) at the nucleotide level, the tohnsw module of GSearch took 2.3 h on a 24-thread computing node and scaled moderately well with increasing number of threads (Figure 2a). Maximum memory (RAM) required for the building step was 28.3 GB. The total size of written database files on disk was ~3.0 GB. There are 3 layers for the resulting graph, 65180, 519, and 4 genomes for layer 0,1 and 2 respectively. The searching of query genomes against this database graph, requesting best 50 neighbors for 1000 query genomes, which represented different previously known as well as novel species of eight bacterial phyla (see Methods for details on query genome selection), took 2.3 min (database loading 6 seconds) on a 24-thread machine and also scaled well with increasing number of threads (Figure 3a). The memory requirement for the request (search) step was only 3.0 GB for storing the entire database file in memory. To evaluate the accuracy of these results, we compared the best neighbors found by GSearch with brute-force FastANI and GTDB-Tk. All best neighbors found by brute-force FastANI and GTDB-Tk for query genomes with close relatives in the database (e.g., ANI > 78%) were found by GSearch (Supplemental File 1). Top 5 neighbors were 99.4% overlapping and top 10 were 96.3% overlapping between GSearch and the other two methods for the testing query genomes. We also compared the speed with MASH for the same kmer and sketch size and the MASH dist step took 7.51 min to compare 1000 genomes with database using 24 threads. The speed difference compared to MASH was even greater for ~8,000 query genomes. Specifically, it took 12.5 min for

GSearch to find the top 50 best hits (Supplementary Figure S4a) while MASH took 80.8 minutes on the same 24-thread machine. However, for a given number of database genomes, the speedup of GSearch is saturated to log(N) as the number of query genome increases, where N is the number of database genomes. Therefore, GSearch will be orders of magnitude faster than MASH for larger species database with millions of genomes (see also viral section below). GSearch search time for a given number of query genomes is related to the number of database genomes in a O(log(N)) manner while brute-force methods are O(N), and our empirical analysis is consistent with the theoretical log(N) prediction (Supplementary Figure S4b and Supplementary Note 3).

To build the amino-acid-level graph for moderately related query genomes, all GTDB v207 genomes were used for gene calling by FragGeneScanRs and subsequently, the predicted amino acid sequences for each genome were used for the tohnsw module. The graph building step took 1.4 h (Figure 2b) with a maximum memory required for the building step to be 37.7 GB. The total size of written database files on disk by GSearch was 5.9 GB. There were 65158, 543 and 2 genomes for layer 0,1 and 2 respectively. Requesting 50 neighbors for 1000 genomes at the amino-acid level took 1.52 minutes with a memory requirement of ~6.0 GB (database loading 9 seconds; Figure 3b). The top 5 neighbors had a 98.9% recall compared to the brute-force MASH or blast-based AAI approaches, with 97.1% overlap for the 10 top neighbors. In comparison, MASH dist took 5.96 min using 24 threads; for 8000 query genomes, MASH dist took 47.2 min while GSearch took 5.6 min.

Finally, for most distantly related query genomes, the graph building for the universal gene set follows the same logic as the amino acid level graph mentioned above except for using a smaller kmer size (k=5) due to the smaller kmer space of ~120 universal genes vs. the whole-genome level (e.g., a few thousand genes). It took 7.76 min to build the database (Figure 2c) and 32 seconds to request 50 neighbors for 1000 queries on a 24 threads node (Figure 3c) with a recall similarly high to the amino-acid level search (with top 5 and top 10 recall ranging between 98.2% and 96.1%, respectively).

We also evaluated the effect of genome completeness on search and classification accuracy given that bacterial genomes recovered from environmental metagenomes are frequently incomplete. GSearch was robust to genome incompleteness down to 50% completeness level, e.g., with 80% of top 10 best matches are found, while accuracy decreased considerably below this level (Supplementary table S6).

Graph database building and searching for viral and fungal genomes

Graph building and requesting for viral genomes is not effective at the nucleotide level because many viral genera are too diverse and do not have close relatives in the public genomic database; that is, the database is too sparse. Accordingly, kmer-based methods (e.g., MASH and ProbMinHash) will often lead to imperfect graph structure for viral genomes. Therefore, we build only an amino acid level graph for viral genomes, using all genes in the genome due to the lack of universal genes for viral genomes. Database building took 13.895 h on a 24-thread node and graph file on disk is 15.8 GB (Supplementary Figure S6 (a)). Requesting 1000 neighbors scaled well with increasing number of threads and took about 3.63 min (database load takes additional 1.1 min) using 24 threads (Supplementary Figure S6 (b)). The top 10 neighbors for 1000 query phage genomes were still highly overlapping (98.32% recall; Supplemental Table S1) with the brute-force MASH-based approach. For such large database, GSearch is about 20X faster than the brute-force MASH (Supplementary Tables S1). We also compared GSearch with a new database building method called PhageCloud, which relies on manually curated genome labels (e.g., environmental source) for graph database building in Neo4j database software and Dashing software for distance/relatedness computation. Since PhageCloud provides only a website and allows only one genome query at a time, we searched only one viral genome at a time with GSearch and MASH against the same database (Gut Phage Database [34]). It took 37 seconds to find the two best matches with PhageCloud while GSearch took 15 seconds (database loading 14 seconds, search 1 second) for the same search. MASH on the other hand took 4 minutes to find the same 2 best matches. It should be noted, however, that, because the database is already available (loaded) on PhageCloud's website, 37 seconds is only for search and website responses (average value for 5 runs on 5 different days) whereas GSearch took only 1.5 second for the same step.

Graph building for fungal genomes is slower compared to prokaryotic genomes, despite the smaller number of available fungal genomes (n=9700) because the average fungal genome size is much larger and kmer and sketch size are accordingly much larger (k=21, s=48000). It took 2.3 h on a 24-thread node to build the nucleotide level graph for these fungal genomes. Searching step was also slower due to the larger kmer space.

Accordingly, it took 3.13 min to identify 50 neighbors for 50 query fungal genomes while MASH tool 4.4 min. Nonetheless, top 5 recall was still very high (~99.4%) against MASH and MUMMER-based ANI for the same datasets. For the amino acid level graph, the time for graph building was only 0.61 h, shorter than the corresponding prokaryotic graph due to the lower coding density of fungal genomes relative to the prokaryotic genomes. Identifying 50 neighbors for 50 query fungal genomes at the amino-acid level took 1.24 min (MASH took 2.59 min) with similarly high top 5 and top 10 recall (99.7% and 98.5%, respectively) against brute-force MASH (-a) and blastp-based AAI. Note that the difference in run time will be much larger between MASH and GSearch as the number of fungal database genomes increases in the future, as also exemplified above for the bacterial genomes

Combining the three graphs/levels together and comparison with GTDB-Tk for prokaryotic genome classification

A three-step pipeline was developed to allow the identification and classification of a query genome, depending on its level of novelty compared to the database genomes (Figure 4). Specifically, when the query genome does not find a match in the database better than ANI > 78%, corresponding to ProbMinHash distance 0.9850, the nucleotide-level graph is abandoned, and the amino-acid level is used instead. If no match against the latter graph is found above 52% AAI, corresponding to 0.9375 ProbMinHash distance, the amino-acid level is abandoned, and the universal gene graph is used instead (uAAI based on universal gene below 80% indicates new order or higher taxonomic rank; Figure 4). The overall running time to classify 1000 prokaryotic genomes of varied levels of taxonomic novelty on different computing platforms is showed in Table 1. On a 24-thread Linux node with Intel Xeon Gold 6226 CPU, it took a total of 5.85 minutes while it took 19.49 minutes on an intel Core i7 laptop (2017 release) CPU personal laptop (6.02 minutes on the most recent ARM64 CPU laptop). Classifying 1000 genomes using GTDB-Tk took 5.91 h on the same Linux node with 24 threads (Figure 3 (d), memory requirement was ~328G) while MASH took 53.7 min for 1000 genomes using 24 threads for the 3 steps.

In terms of accuracy, all query genomes that had a best match higher than 78% ANI against the GTDB database genomes (i.e., a match at the same or closely related species, 699 out of the total 1000) were identically classified by GSearch, GTDB-Tk and FastANI (Supplementary File 1, only 100/699 are shown for simplicity). For the remaining 301 genomes that did not have same or closely related species-level matches, for 266 of them (or 87.1%), GSearch also provided the same classification with GTDB-Tk but several inconsistencies were observed for 39/301 genomes (Supplementary Figure S5). Specifically, we noticed that for GTDB-Tk, which relies on RED values and tree topology, several genomes (n=14) were still classified at the genus level even though the AAI value against the best database genome in these case was below 60% (typically, genomes assigned to the same genus show >65% AAI [21]), and some genomes (n=16) were still classified at the family level but not at the genus level even though their best AAI value was above 65%. Similarly, several genomes (n=9) were classified at the order level but not family level even though their best AAI value was above 52%. Therefore, high consistency was overall observed between GSearch and GTDB-Tk assignments, and the few differences noted were probably associated with contaminated (low quality) MAGs or taxonomic inconsistencies, which was challenging to assess further, and/or the peculiarities of each method. Since ProbMinHash distance correlated well with blastp-based AAI in the range of AAI values between 52% and 95%, the classification results were always consistent with AAI-based classification using previously proposed thresholds. For example, best matches at AAI 65% ≥ AAI were classified in the same genus by GSearch and blast-based AAI and best matches of 52% < AAI < 65% were typically classified in the same family [35] .

Database split for large genomic species database

For large databases (for example, >1 million bacterial genomes), the graph building and requesting step could require a large amount of memory (due to the larger kmer space) that is typically not available in a single computer node. We therefore provide a database split solution for such large databases. The average database building time on each node (for each piece of the database after the splitting step) scales linearly with increasing nodes/processors (Supplementary Figure S7(a)) and requires much less memory (1/n total memory compared to when building in one node where n is the number database pieces after splitting; for GTDB v207 nucleotide graph building and n=5, it will be only 28.3 G/5=5.66 G). The searching time scales sub-linearly with increasing number of nodes (Supplementary Figure S7(b)), but offers the advantage of a reduced memory footprint with respect to the single-node search. The top 10 best neighbor by splitting the database were exactly the same as the non-splitting strategy (Supplementary file 2). Note that without multi-node support (e.g., run database build sequentially), database build time is nearly the same with non-split strategy, but memory requirement is only 1/n (GTDB v207, 28.3G/5=5.66G at nucleotide level and 37.7G/5=7.54G at amino acid level), despite the fact that total request time will be larger (time*n in Supplementary Figure S7(b)). However, since the request step is very fast with only 1/n memory requirement (e.g., loaded graph database files for GTDB v207 will be about only 3G/5=0.6G), even for a decent number of pieces, overall runtime is still short with the database split approach. The database split strategy is especially useful when memory requirement is not satisfied on host machine for larger genomic species database (e.g., millions of genomes).

Discussion

A popular way to assess genetic relatedness among genomes is ANI, which corresponds well to both 16S/18S rRNA gene identity and DNA-DNA hybridization values, the golden standards of fungal and prokaryotic taxonomies [13]. However, the number of available microbial genomes has recently grown at an unprecedented speed. For example, there are 30% more (new) species in GTDB v202 (2020) vs. v207 (2022), and the number of bacterial species represented by genomes alone is expected to surpass 1 million soon. Therefore, the traditional way that blast-based ANI or faster kmer-based implementations (e.g., FastANI or MASH) are applied as an all vs. all search strategy (brute-force) does not scale because the running time grows linearly with increasing number of query genomes and/or genomes in the database. Phylogenetic approaches based on quick (approximate) maximum likelihood algorithms and a handful of universal genes as implemented -for example- in GTDB-Tk could be faster than brute-force approaches but are often not precise and require a large amount of memory for the querying step [20, 36] while the database building step could take several weeks of run time because the underlying multiple sequence alignment of the database genomes is computationally intensive. Further, approaches that reply on k-medoid clustering to avoid all vs. all comparisons could be sometimes trapped into local minima because of arbitrary partitioning of database genomes into clusters, a known limitation of these methods [21]. Our GSearch software effectively circumvents these limitations by combining a new kmer hashing-based algorithm for fast computation of genetic relatedness among genomes (ProbMinHash) with a graph based nearest neighbor search algorithm (HNSW). Accordingly, GSearch is at least an order of magnitude faster than alternative approaches for the same purposes. Note that GSearch could also be applied to whole metagenome search and identification of the most similar metagenomes in a series because ProbMinHash can estimate metagenomic distance in a similar way to genomes.

To the best of our knowledge, no current tool can efficiently search very large genome databases. GSearch is able to handle a million microbial genomes on a small-to-average computer cluster since the dumped database file size is proportional to the total number of genomes in database for fixed sketch size and graph parameters. Specifically, with one million genomes, the dumped file size (amino acid) will be 5.9G*20=118 GB (now we have only ~60K, for which database file size is 5.9G), a modest computational requirement for current computer clusters or even personal laptop computer. Further, due to the nature of graph based NNS algorithms, there is no need to build the entire database at once, but the database can be split it into smaller pieces and thus, a separate graph database be built for each piece as exemplified above and depending on the computational resources available. For a modern laptop with 16 GB memory, a database on one million species can be split into 10 pieces, so the dumped file for each piece will be only 11.8 GB, which can be loaded into memory, and then collect the results from each piece within an approximate total running time of 30 minutes (assuming each part will be 3 minutes for 1000 query genomes against 0.1 million database genomes). With this logic, a computing node with 24 threads and 256 GB of memory available can easily deal with 20 million bacterial database genomes. This represents a substantial improvement compared to existing tools for the same purposes.

It is also important to note that we could seamlessly replace ProbMinHash with another relatedness algorithm should such an algorithm become available and has advantages in terms of speed and/or precision. Related to this, ANI as currently implemented -for instance- in FastANI is not appropriate for this function because it is not metric (that is, for the FastANI distances calculated among three genomes A, B, and C, (A,B) + (B, C) is not necessary larger than (A,C), especially for genomes related at the phylum level). To solve this “metric” problem, a norm adjusted proximity graph (NAPG) was proposed based on inner product and it shows improvements in terms of both speed and recall [37]. This could be another direction for further improving the speed and recall of GSearch and/or the use of other metrics in place of ProbMinHash distances. In the meanwhile, ProbMinHash was used in GSearch because it is metric [33, 38], which ensures neighbor diversity when building the graph, but it is equally applicable to any microbial genome, including viral and fungal genomes, in addition to its advantages for kmer weighting and normalization mentioned above.

Another distinguishing aspect of GSearch (tohnsw module) is the speed and flexibility in building reference databases. Users could build reference databases (graphs) for any number and type (e.g., prokaryotic vs. viral) of genomes, up to several millions of genomes. The high efficiency in building graphs allows users to also test and optimize the key parameters of the graph, the M and ef_construct parameters. For any given database size, M and ef_construct determine the quality of the graph and graph build speed. Small M and ef_construct may lead to frequent traps in local minima and thus, low recall while large M and ef_construct may lead to slow speed without proportional improvement in recall (Supplemental Table S2). Therefore, there is a tradeoff between accuracy and speed that should be evaluated first. However, for most users this task would not be necessary because they will work with pre-built databases such as those provided here. Further, the search step against these pre-build databases with query genomes of known taxonomy for evaluating recall and tradeoffs can be performed, within minutes, on any modern laptop with 5-6 GB of memory (Table 1).

Kmer-based methods for genetic relatedness estimation such as ProbMinHash have lower accuracy between moderately-to-distantly related genomes compare to alignment-based tools (see supplement Note 4 for further discussion). Our empirical evaluation showed that this relatedness level, for nucleotide searches, is around 78% ANI and 52% AAI for the amino-acid searches (e.g., ProbMinHash distances do not correlate well with blast-based ANI and AAI at these levels). To circumvent this limitation, we designed a 3-step framework as part of GSearch to classify bacterial genomes that show different levels of novelty compared to the database genomes, with high accuracy. This framework included a search at the universal gene level for deep-branching genomes that are novel at the phylum level (AAI < 52%), for which searching at the entire proteome level is less accurate. Recently, methods that employ kmers that allow mismatches, that is, spaced kmers [39], have shown promise in accurately estimating genomic relatedness even among distantly related genomes with gains in speed. To apply spaced kmers to entire genomes, the recently developed "tensor sketch" approaches could be explored in the future to simplify the pipeline for bacterial and viral genomes [40]. In the meanwhile, the ProbminHash approach, essentially a Jaccard distance estimation via MinHash-based analysis of kmers, is highly efficiently, and, importantly, can effectively deal with incomplete genomes or genomes of (drastically) different length, an known limitation of MASH-based methods [41]. Comparing genomes of different length is not uncommon, e.g., bacterial genome size can differ by more than two-fold, as can be the case between MAGs of different level of completeness or when searching a short sequence (e.g., a bacteriophage genome) against a large genome collection (e.g., the whole viral genome database). Our own analysis showed that ProbMinHash is robust down to 50% completeness level (Supplemental Table S6), which is also the most commonly used standard for selecting MAGs of sufficient/high quality [42]. ProbMinHash is also robust with completed genomes with repeats or gene duplications due to the kmer weighting step.

In general, the genome relatedness estimated, or best database matching genomes identified, by GSearch were highly consistent with blast-based AAI results or phylogenetic placement of the genome using GTDB-Tk, particularly for query genomes with close relatives in the database related at the species or genus level (Supplementary File 1, Supplementary Figure S5). For more distantly related query genomes relative to database genomes, classification results of GSearch showed some differences with GTDB-Tk. These differences were not always possible to assess further for the most correct genome placement but could be due, at least partly, to the incompleteness and/or contamination of query or/and database genomes, which renders the resulting concatenated alignment of universal genes used by GTDB-Tk unreliable [43] as only a few amino-acid positions per gene are used in the final alignment. In contrast, the AAI and ProbMinHash approaches should be more robust to changes of a small number of genes because the entire proteome is considered [17].

Graph-based NNS methods achieve good performance compared to tree based and locality-sensitive hashing (LSH) methods. Building a HNSW graph relies on proximity of the database elements; so, if the distances among database elements, in our case genomes, cannot be effectively estimated via hashing algorithms, the navigation in graph will be less efficient (e.g., gets trapped in local minima) because the edges to choose from will not be accurate estimations of the relatedness of the corresponding genomes. This is especially problematic for highly sparse/distantly related and diverse datasets, like the viral genome database, in which two phage genomes could often share very little genomic information (kmers). This is confirmed by our own results when using nucleotide-level search to build the viral graph. Hence, the amino acid level will be much more robust for viral genomes and is the recommended level to use. Finally, the HNSW graph, and graph-based K-NNS in general, can be further improved by adding shortcut edges and maintaining a dynamic list of candidates, compared to a fixed list of candidates by default [44]. Graph reordering, a cache optimization that works by placing neighboring nodes in consecutive (or near-consecutive) memory locations, can also be applied to improve the speed of HNSW [45]. Another new direction for graph based NNS will be using Graphics Processing Unit (GPU) instead of CPU because GPUs are more efficient in handling matrix computations and machine learning tasks [46]. We will explore these options in future versions of GSearch.

To summarize, GSearch, based on Probminhash and HNSW, solves a major current challenge in classification of microbial genomes, especially given the exponential increase in the number of newly sequenced genomes due to its efficiency and scalability. GSearch will serve the entire microbial sciences for years to come since it can be applied to fungal, bacterial and viral genomes, while offering a common framework to identify, classify and study all microbial genomes, and will accelerate the process to find new biological knowledge.

Data availability

All the mentioned pre-built database for bacteria, fungi and phage genomes can be found at: http://enve-omics.ce.gatech.edu/data/gsearch

Author Contribution

J.Z, L.M and K.K designed the work, J.Z and J.P-B wrote the code (Genome part and algorithm part respectively), J.P-B implemented the Rust libraries of Kmerutils, Probminhash and Hnswlib-rs. J.Z and K.K wrote the paper. J.Z did the analysis and benchmark.

Acknowledgment

This work was supported, in part, by the US National Science Foundation (Award No 1759831 and 2129823 to KTK). We want to thank PACE (Partnership for Advanced Computing Environment) at Georgia Tech for providing computing resources. We want to thank Kenji Gerhardt for helpful discussions on benchmarking against traditional ANI/AAI based tools and Chirag Jain for discussions on the graph based nearest neighbor search.

References

1. Bobay, L.-M. & Ochman, H. Biological species in the viral world. *Proceedings of the National Academy of Sciences* **115**, 6040-6045 (2018).
2. Bobay, L.-M. & Ochman, H. Biological Species Are Universal across Life's Domains. *Genome Biology and Evolution* **9**, 491-501 (2017).
3. Locey, K.J. & Lennon, J.T. Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences* **113**, 5970-5975 (2016).
4. Chevallereau, A., Pons, B.J., van Houte, S. & Westra, E.R. Interactions between bacterial and phage communities in natural environments. *Nature Reviews Microbiology* **20**, 49-62 (2022).
5. Federhen, S. Type material in the NCBI Taxonomy Database. *Nucleic Acids Research* **43**, D1086-D1098 (2014).
6. Almeida, A. et al. A unified catalog of 204,938 reference genomes from the human gut microbiome. *Nature Biotechnology* **39**, 105-114 (2021).
7. Nayfach, S. et al. A genomic catalog of Earth's microbiomes. *Nature Biotechnology* **39**, 499-509 (2021).
8. Caro-Quintero, A. & Konstantinidis, K.T. Bacterial species may exist, metagenomics reveal. *Environmental microbiology* **14**, 347-355 (2012).
9. Deng, L. et al. Viral tagging reveals discrete populations in Synechococcus viral genome sequence space. *Nature* **513**, 242-245 (2014).
10. Rodriguez-R, L.M., Jain, C., Conrad, R.E., Aluru, S. & Konstantinidis, K.T. Reply to: "Re-evaluating the evidence for a universal genetic boundary among microbial species". *Nature Communications* **12**, 4060 (2021).
11. Parks, D.H. et al. GTDB: an ongoing census of bacterial and archaeal diversity through a phylogenetically consistent, rank normalized and complete genome-based taxonomy. *Nucleic Acids Research* (2021).
12. Roux, S. et al. IMG/VR v3: an integrated ecological and evolutionary framework for interrogating genomes of uncultivated viruses. *Nucleic acids research* **49**, D764-D775 (2021).

13. Konstantinidis, K.T. & Tiedje, J.M. Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences* **102**, 2567-2572 (2005).
14. Konstantinidis, K.T. & Tiedje, J.M. Towards a Genome-Based Taxonomy for Prokaryotes. *Journal of Bacteriology* **187**, 6258-6264 (2005).
15. Goris, J. et al. DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology* **57**, 81-91 (2007).
16. Ondov, B.D. et al. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology* **17**, 132 (2016).
17. Jain, C., Rodriguez-R, L.M., Phillippy, A.M., Konstantinidis, K.T. & Aluru, S. High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries. *Nature Communications* **9**, 5114 (2018).
18. Brown, C.T. & Irber, L. sourmash: a library for MinHash sketching of DNA. *Journal of Open Source Software* **1**, 27 (2016).
19. Bovee, R. & Greenfield, N. Finch: a tool adding dynamic abundance filtering to genomic MinHashing. *Journal of Open Source Software* **3**, 505 (2018).
20. Chaumeil, P.-A., Mussig, A.J., Hugenholtz, P. & Parks, D.H. GTDB-Tk: a toolkit to classify genomes with the Genome Taxonomy Database. *Bioinformatics* **36**, 1925-1927 (2019).
21. Rodriguez-R, L.M. et al. The Microbial Genomes Atlas (MiGA) webserver: taxonomic and gene diversity analysis of Archaea and Bacteria at the whole genome level. *Nucleic Acids Research* **46**, W282-W288 (2018).
22. Wang, Q., Garrity, G.M., Tiedje, J.M. & Cole, J.R. Naïve Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. *Applied and Environmental Microbiology* **73**, 5261-5267 (2007).
23. Schloss, P.D. et al. Introducing mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities. *Applied and Environmental Microbiology* **75**, 7537-7541 (2009).
24. Indyk, P. & Motwani, R. in Proceedings of the thirtieth annual ACM symposium on Theory of computing 604-613 (1998).
25. Gionis, A., Indyk, P. & Motwani, R. in Vldb, Vol. 99 518-529 (1999).
26. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM* **18**, 509-517 (1975).
27. Dasgupta, S. & Sinha, K. in Proceedings of the 26th Annual Conference on Learning Theory, Vol. 30. (eds. S.-S. Shai & S. Ingo) 317--337 (PMLR, Proceedings of Machine Learning Research; 2013).
28. Dong, W., Moses, C. & Li, K. in Proceedings of the 20th international conference on World wide web 577-586 (2011).
29. Malkov, Y., Ponomarenko, A., Logvinov, A. & Krylov, V. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* **45**, 61-68 (2014).
30. Fu, C., Xiang, C., Wang, C. & Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).
31. Malkov, Y.A. & Yashunin, D.A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **42**, 824-836 (2020).

32. Aumüller, M., Bernhardsson, E. & Faithfull, A. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* **87**, 101374 (2020).
33. Ertl, O. ProbMinHash – A Class of Locality-Sensitive Hash Algorithms for the (Probability) Jaccard Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 1-1 (2020).
34. Camarillo-Guerrero, L.F., Almeida, A., Rangel-Pineros, G., Finn, R.D. & Lawley, T.D. Massive expansion of human gut bacteriophage diversity. *Cell* **184**, 1098-1109.e1099 (2021).
35. Konstantinidis, K.T., Rosselló-Móra, R. & Amann, R. Uncultivated microbes in need of their own taxonomy. *The ISME Journal* **11**, 2399-2406 (2017).
36. Chaumeil, P.-A., Mussig, A.J., Hugenholtz, P. & Parks, D.H. GTDB-Tk v2: memory friendly classification with the Genome Taxonomy Database. *bioRxiv*, 2022.2007.2011.499641 (2022).
37. Tan, S. et al. in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery &amp; Data Mining 1552–1560 (Association for Computing Machinery, Virtual Event, Singapore; 2021).
38. Moulton, R. & Jiang, Y. in 2018 IEEE International Conference on Data Mining (ICDM) 347-356 (2018).
39. Břinda, K., Sykulski, M. & Kucherov, G. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics* **31**, 3584-3592 (2015).
40. Joudaki, A., Rätsch, G. & Kahles, A. Fast Alignment-Free Similarity Estimation By Tensor Sketching. *bioRxiv* (2021).
41. Koslicki, D. & Zabeti, H. Improving MinHash via the containment index with applications to metagenomic analysis. *Applied Mathematics and Computation* **354**, 206-215 (2019).
42. Bowers, R.M. et al. Minimum information about a single amplified genome (MISAG) and a metagenome-assembled genome (MIMAG) of bacteria and archaea. *Nature Biotechnology* **35**, 725-731 (2017).
43. Tan, G. et al. Current Methods for Automated Filtering of Multiple Sequence Alignments Frequently Worsen Single-Gene Phylogenetic Inference. *Systematic Biology* **64**, 778-791 (2015).
44. Prokhorenkova, L. & Shekhovtsov, A. in Proceedings of the 37th International Conference on Machine Learning, Vol. 119. (eds. D. Hal, III & S. Aarti) 7803--7813 (PMLR, Proceedings of Machine Learning Research; 2020).
45. Coleman, B., Segarra, S., Shrivastava, A. & Smola, A. Graph Reordering for Cache-Efficient Near Neighbor Search. *arXiv preprint arXiv:2104.03221* (2021).
46. Groh, F., Ruppert, L., Wieschollek, P. & Lensch, H. GGNN: Graph-based GPU Nearest Neighbor Search. *IEEE Transactions on Big Data*, 1-1 (2022).

Methods and Materials

Briefly, GSearch is composed of the following steps. Initially, the genetic relatedness among a collection of database genomes is determined based on the ProbMinHash algorithm, which computes the normalized weighted Jaccard distance using the probminhash3a algorithm implemented in the ProbMinHash paper[1]. The normalized weighted Jaccard distances are then used as input for building HNSW graphs (note that a distance computation is required only when that genome pair is required for graph building, thus GSearch avoids all vs. all distance computations). Genomes are subsequently recursively added as the nearest neighbors of each node in the built graph file with the same distance computation procedure. The built graph database file is stored on disk. Query genomes are then searched against graph database and subsequently, best neighbors are returned for classification/identification. In this process, the best neighbor (or neighbors) is also identified based on the smallest normalized weighted Jaccard distance obtained.

ProbMinHash

Details of differences between ProbMinHash and traditional MinHash can be found in Supplenmentary Methods & Materials. We reimplemented the Probminhash algorithm in Rust to estimate genetic relatedness between any two genomes based on normalized (weighted) Jaccard distances according to the original ProbMinHash paper [1] (Supplementary Note 1) . The Rust reimplementation of Probminhash can be found at: https://github.com/jean-pierreBoth/probminhash. Two important parameters of Probminhash are the sketch size and kmer size. Similar to MinHash sketches, Probminhash sketches are also shared hashes from hashed kmer set by taking into account the kmer weights and also total kmer count (See Figure 1 of MASH paper). Time complexity analysis for ProbMinHash is shown in Supplementary Note 3.

To benchmark probminhash against MASH, both tools were run with the same sketch size (s=12000) and kmer size (k=16) for bacterial genomes at the nucleotide level and kmer size (k=7) at the amino acid level for both database building and searching. For fungal genomes a larger sketch size (48000) was used due to much larger genome sizes. Details of kmer choosing logic can be found in Supplementary Note 2. For graph search results, we also performed the same transformation of MASH distance from normalized weighted Jaccard distance to probMASH distance for convenient comparison to ANI based methods.

$$probMASH = -\frac{1}{k} ln \frac{2 * J_p}{J_p + 1}$$

Hierarchical Navigable Small World Graphs (HNSW)

Generally, the framework of graph-based ANN search algorithm (here HNSW) can be summarized as the following two steps: 1) build a proximity graph (HNSW) where each node represents a database vector. Each database vector will connect with a few of its neighbors while maintaining small world property in each layer of HNSW. 2) Given a query vector (or sequence, kmer profile in our case), perform a greedy search on the proximity graph by comparing the query vector with database vectors under the searching measures (e.g., cosine similarity or L2 similarity, in our case probminhash distance). Then, the most similar candidates are returned as outputs. The key point for these two-step methods is step 1, to construct a high-quality index graph, which provides a proper balance between the searching efficiency and effectiveness. To guarantee the searching efficiency, the degree (number of maximum allowed neighbors, denoted as M) of each node is usually restricted to a small number (normally 20~200) while width of search for neighbor during inserting (denoted as ef_construct) is usually a larger number (higher than 1000) to increase the chance to find best M neighbors by increasing the diversity of neighbors due to the large number of neighbors retained. Building graph and searching query against the graph follow very similar greedy search procedures except that there is an extra reverse updating of neighbors list for each vector when inserting database vector (building), one by one, into the existing graph (Figure 1a). The first phase of the insertion/building process starts from the top layer by greedily traversing the graph in order to find maximum M closest neighbors to the inserted element P in the layer by doing *ef*_construct times search (Figure 1a). After that, the algorithm continues the search from the next layer using the closest neighbor found from the previous layer as entry point, and the process repeats until to the bottom layer. Closest neighbors at each layer are found by a greedy and heuristic search algorithm (Figure 1b and c). For building, after searches are finished at the bottom layer for each inserted element, a reverse update step will be performed to update the neighbor list of each node in the existing graph while for searching this is not needed. The overall database building time complexity is

O(N*log(N)), where N is the number of nodes in the graph. For searching, since there is no need to reverse update best neighbor list for each node in the graph, time complexity is (only) O(log(N)) (See Supplementary Note 3). Theoretical guarantee of graph-based algorithm can be found in Supplementary Note 5. We reimplemented the original hnswlib library written in C++ using the Rust programming language for its memory safety and thread use efficiency [11], which can be found here (https://github.com/jean-pierreBoth/hnswlib-rs). Benchmarks for this package against standard datasets can be found in the Supplementary Methods & Materials.

Details of program implementation in Rust

There are 2 modules in total: tohnsw and request. Tohnsw is to build graph by gradually inserting genomes into graph while request is to query new genomes against the graph database built in the tohnsw step. Tohnsw starts from reading database genomes and generating kmer profile and sketches for distance calculation. By selecting a random genome as the first genome to insert to the graph, tohnsw module gradually add genomes to existing graph file following HNSW constructing rules mentioned above by computing ProbMinHash distance between genomes. Whenever a genome is going to be inserted into the existing graph, each genome in the graph is associated with a list that stores the M closest neighbors/genomes to the genome and the distance to these neighbors. Then, the distances of this genome with the nearest neighbors (M) of entry genome in this layer will be computed/searched (ef_construct times) using probminhash3a algorithm and the smallest distance of the neighbor genomes will be the new entry genome. This process will be repeated until the nearest genomes (<= M) in the layer are found and subsequently, the program will go to the layer below, using the genome that was represented by the nearest genome in the above layer as new entry genome in the new layer. The search layer algorithm is repeated until to the bottom layer is reached/analyzed. In contrast to the default settings in the original hnswlib, we allow the two parameters of neighbor selecting heuristics, *extendCandidates* to be true and *keepPrunedConnections* to be false because our genomic data is extremely clustered and there is no need to fix the number of connections per element considering the maximum connection allowed. Request module will load the graph database and then search query genomes against it to return the best neighbors of each query, following exact the same procedure with building step without updating the database. Both tohnsw and request module are operating in parallel for high performance (see Supplementary Note 6). The GSearch software can be found here: https://github.com/jean-pierreBoth/gsearch GSearch relies on Kmerutils (https://github.com/jean-pierreBoth/kmerutils), which is a Rust package we developed to manipulate genomic fasta files including kmer string compression, kmer counting, filtering using cuckoo filter et.al.

Installation guide, manual and pre-built binaries can also be found on the website. We provide static binaries on the release page for major platforms such as Linux and MacOS, with support for different CPU structures, e.g. Intel x86_64 or ARM64. GSearch program can be run like this : 1) Build a graph database, which can be done running the following command: tohnsw -d ./GTDB_r207 -k 16 -s 12000 -n 128 --ef 1600; 2) Request neighbors of query genomes: request -b . -r ../query_folder -n 50 (--aa). Note that with the --add option in tohnsw module, genomes in the directory will be added to existing graph database, loaded from current directory, thus avoiding building graph database from the very beginning when there are only a small number of new genomes species compared to the current database. However, for larger number of new genome species, rebuild from start is suggested to be able to choose an optimal M and ef_construct to maintain high accuracy.

Prokaryotic classification pipeline

The amino-acid level graph showed that closest neighbors were found, with high recall, when the query shared at least 52% AAI to its best neighbor. For more divergent genomes, showing lower than 52% AAI, whole-genome amino-acid level graph loses accuracy and we had to switch to universal, single-copy protein-coding genes. For the nucleotide-level graph, we used kmer=16 for bacteria and archaea to have high specificity for closely related database genomes (e.g., sharing about 95% ANI). For building the whole-genome amino-acid graph, we used k=7 to have the best specificity without *compromising* sensitivity, which is also consistent with previous results on classification of amino acid sequences based on kmers [2]. For building graph based on universal gene set, we use k=5 because of much smaller total amino acid size. For further details on the range of kmer to use for bacteria genome and proteome, viral genome and proteome, see Supplemental Notes 2.

The proteome of each genome was predicted by FragGeneScanRs v0.0.1 for performance purpose as opposed to Prodigal despite small loss in precision (Supplementary Table S5) [3]. Hmmsearch in the hmmer (v3.3.2) software [16] was used to extract the universal gene set for bacteria and archaea genomes (universal gene graph).

Note that for viral genomes, this last step was not used because there are no universal single copy genes for viral genomes. Evaluation of the speed and memory requirements for all steps mentioned above were performed on a RHEL (Red Hat Enterprise Linux) v7.9 with 2.70 GHz Intel(R) Xeon(R) Gold 6226 CPU. Unless noted otherwise, all 24 threads of the node are available by default.

Distributed implementation and database splitting

To accommodate the increasing number of genomes that become available at an unprecedented speed in recent years and will soon reach 1 million or more, we provide an option to randomly split the database into a given number of pieces and build graph database separately for each piece. In the end, all best neighbors returned from each piece will be pooled and sorted by distance to have a new best K neighbor collection returned to the user for each query genome. We hereby prove that in terms of requesting top K best neighbors, the database split strategy is equivalent to non-split database strategy as long as the requested best neighbors for each database piece is larger than or equals to requested best neighbors in the non-split strategy. The underlying reason is that the best neighbors globally are also the best locally [4]. The database split and request will be done sequentially, on one node, without multi-node support. For now, we split GTDB database in to 5 pieces for testing purposes. In theory, a large database can be split into any pieces as long as each piece can be used to build HNSW. In practice, a reasonable way to decide on the number of database pieces to use is so that memory requirement for each piece is equal or smaller than the total memory of host machine.

The database split idea has been used in several graph-based larger scale (e.g., billions) nearest neighbor search tasks in industry [4, 5].

Species database and testing genomes for benchmarking and recall

GTDB version 207 was used to build the database for bacteria and archaea genome species [6]. The IMGVR database version 3, with species representatives at a $\geq$95% ANI, was used for for viral database building [7]. For fungal genomes, all genomes downloaded from the MycoCosm project (on 24th Jan., 2022) were used [8]. The amino acid sequences of predicted gene on the genomes were obtained using FragGeneScanRs. The Universal Single Copy Gene (USCG) gene set for GTDB genomes were extracted via hmmer software.

To test the performance of our pipeline, we specifically chose genomes that are not included in the GTDB database (the database was used for graph building). In particular, the bacterial/archaeal genomes, mostly MAGs, reported by Ye and colleagues [9] and Tara Ocean MAGs (total 8,466 MAGs) [10] were used. We randomly selected 1000 genomes/MAGs from Ye's collection and use them as query genomes to test the performance and accuracy of GSearch. To compare with other database search tools for large database e.g., the viral database, we compare GSearch with PhageCloud [11], which builds a graph database based on the labels of each viral genome (e.g., environment source) and its search algorithm is Dashing2[12].

Recall of AAI-, ANI- and MinHash-based nearest neighbor searching for bacteria/archaea, fungi and viral genomes.

To benchmark how GSearch performs compared to ANI/AAI- and MinHash-based tools, we ran FastANI, Diamond blastp-based AAI and Mash to find the best neighbors for the same query genome dataset and evaluated whether or not the best neighbors found by GSearch were the same. FastANI parameters for the bacterial dataset were the following: fastANI --ql query_path.txt --rl gtdb_path.txt -k 16 -p 24 --minFrac 3000 -o ANI.txt. GTDB database was split into 50 subsets and each subset was parallelly run on a multi-node supercomputer to reduce memory requirement. MASH parameters were: mash sketch -a (for AA only) -k 21 (7 for AA) -s 12000 -p 24 GTDB/*.fna > gtdb.msh; mash dist -p 24 gtdb.msh query.msh. For AAI calculation, the corresponding script of the enveomics package [13] was used: aai.rb -1 query.faa -2 db.faa -p diamond -t 24. Hmmer was used to search for universal single copy gene against pre-built hmm profiles (120 for archaea and 122 for bacteria respectively); the profiles were obtained from the GTDB-Tk software. For viral genomes, FastANI fragment size of 1000 was used instead of 3000 while aai.rb fragment size was 500 instead of 1000 with minimal number of matches of 5. For viral genomes, MASH kmer size of 11 and 7 was used for nucleotide and amino acid levels, respectively. For fungal genomes, we use MUMMER v4.0.0 with default parameters for ANI calculation [14]. Gene prediction for fungal genomes was performed using GeneMark-ES v2 (--fungus --ES) [15]. Kmer size 21 and 11 was used for fungal genomes in MASH for nucleotides and amino acid levels, respectively. Detailed description of kmer size for each type of genome can be found in Supplemental Note 2.

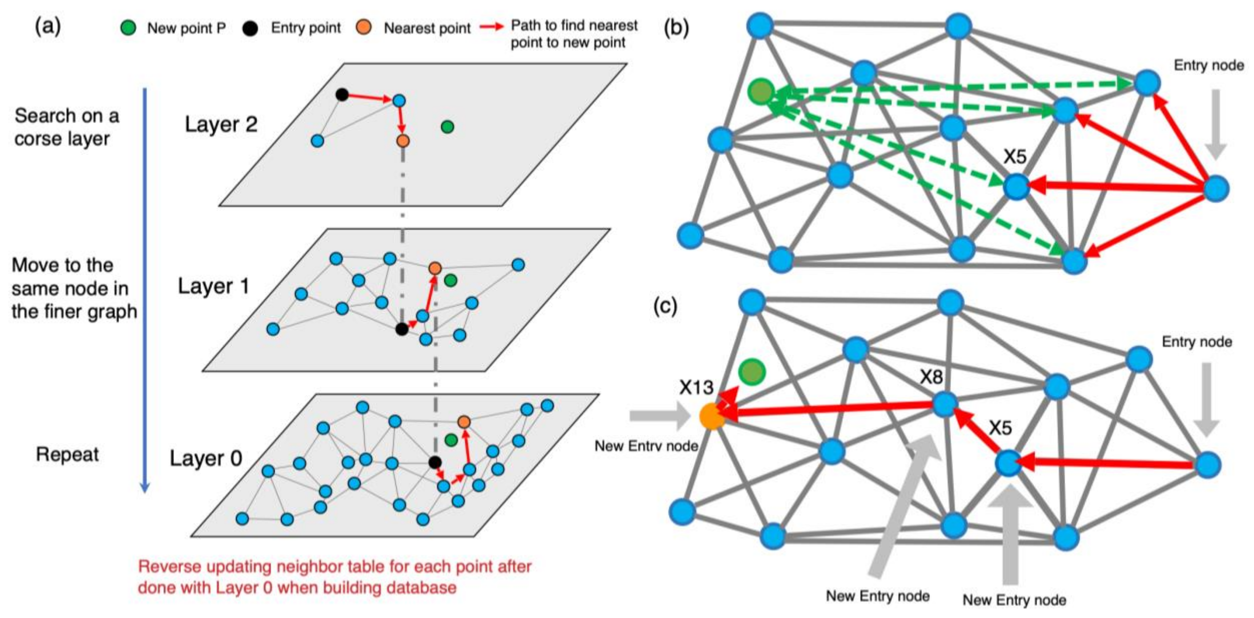
We calculated recall for our tool compared to standard ANI/AAI and MASH in the following way: since biological species database are generally sparse because we are far away from sequencing all species in the environment and likely the existence of natural gaps in diversity, a larger top K by HNSW (e.g., 100) compared to the value used in standard benchmark dataset will offer little, if any advantage, especially when the query genomes are relatively new, e.g. a new family compare to database genomes. Therefore, we use top 5 and 10. Top 5 and top 10 recall are calculated based on top 5 and 10 neighbors found by GSearch and the alterantive tools, and if all top 5 or 10 found by the latter tools were also in top 5 or 10 of our tool, then recall was 100%. Similarly, if only 4 or 9 are found by our tools, then recall was 80% and 90% respectively. However, if the distance of query to some of the top 10 or top 5 neighbors found by GSearch at the nucleotide level was larger than 0.9850 for bacterial genomes, these matches will be filtered out and only those neighbors below 0.9850 will be used (e.g. 8 out of 10 are kept, so only top 8 is compared) because we have shown that above this threshold, MinHash-based methods will lose accuracy and this is not specific to HNSW. Similar rules were applied for the amino acid level searches with the threshold value of 0.9720 used for filtering out bacterial genomes.

References

1. Ertl, O. ProbMinHash – A Class of Locality-Sensitive Hash Algorithms for the (Probability) Jaccard Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 1-1 (2020).
2. Déraspe, M., Boisvert, S., Laviolette, F., Roy, P.H. & Corbeil, J. Fast protein database as a service with kAAmer. *bioRxiv*, 2020.2004.2001.019984 (2020).
3. Van der Jeugt, F., Dawyndt, P. & Mesuere, B. FragGeneScanRs: faster gene prediction for short reads. *BMC Bioinformatics* **23**, 198 (2022).
4. Malkov, Y.A. & Yashunin, D.A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **42**, 824-836 (2020).
5. Fu, C., Xiang, C., Wang, C. & Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).
6. Parks, D.H. et al. GTDB: an ongoing census of bacterial and archaeal diversity through a phylogenetically consistent, rank normalized and complete genome-based taxonomy. *Nucleic Acids Research* (2021).
7. Roux, S. et al. IMG/VR v3: an integrated ecological and evolutionary framework for interrogating genomes of uncultivated viruses. *Nucleic acids research* **49**, D764-D775 (2021).
8. Grigoriev, I.V. et al. MycoCosm portal: gearing up for 1000 fungal genomes. *Nucleic acids research* **42**, D699-D704 (2014).
9. Ye, L., Mei, R., Liu, W.-T., Ren, H. & Zhang, X.-X. Machine learning-aided analyses of thousands of draft genomes reveal specific features of activated sludge processes. *Microbiome* **8**, 1-13 (2020).
10. Nishimura, Y. & Yoshizawa, S. The OceanDNA MAG catalog contains over 50,000 prokaryotic genomes originated from various marine environments. *Scientific Data* **9**, 305 (2022).
11. Rangel-Pineros, G. et al. From Trees to Clouds: PhageClouds for Fast Comparison of∼ 640,000 Phage Genomic Sequences and Host-Centric Visualization Using Genomic Network Graphs. *PHAGE* **2**, 194-203 (2021).
12. Baker, D.N. & Langmead, B. Dashing 2: genomic sketching with multiplicities and locality-sensitive hashing. *bioRxiv* (2022).
13. Rodriguez-R, L.M. & Konstantinidis, K.T. (PeerJ Preprints, 2016).
14. Marçais, G. et al. MUMmer4: A fast and versatile genome alignment system. *PLOS Computational Biology* **14**, e1005944 (2018).
15. Ter-Hovhannisyan, V., Lomsadze, A., Chernoff, Y.O. & Borodovsky, M. Gene prediction in novel fungal genomes using an ab initio algorithm with unsupervised training. *Genome research* **18**, 1979-1990 (2008).

Figures

Figure 1. Schematic overview of GSearch building graph and searching graph


(a)
New point P
Entry point
Nearest point
Path to find nearest point to new point
Search on a corse layer
Move to the same node in the finer graph
Repeat
Layer 2
Layer 1
Layer 0
Reverse updating neighbor table for each point after done with Layer 0 when building database
(b)
Entry node
X5
(c)
Entry node
X13
X8
X5
New Entry node
New Entry node
New Entry node

steps. (a) Graph was clasped into hierarchical layers following exponential decay probability. In this graph, *ef* and *M*, represent the number of searches when finding nearest neighbors and maximum allowed number of neighbors for each node, respectively (See Materials and Methods for details). In each layer, starting from an entry node (random or inherit from layer above it, depending on whether it is the top layer or not), GSearch finds the closest connected neighbor of the entry node and assigns it as the new entry point P (b), and then traverses in a greedy manner (i.e., update the entry point using the newly found closest connected neighbor (c)) until the nearest neighbor in the layer is found, and then goes to next layer. This process is repeated until the required number of nearest neighbors are all found for the given new querying/inserting point. For building graph, after the required number of nearest neighbors are found, a reverse update step is performed to update neighbor list of all nodes in the graph.

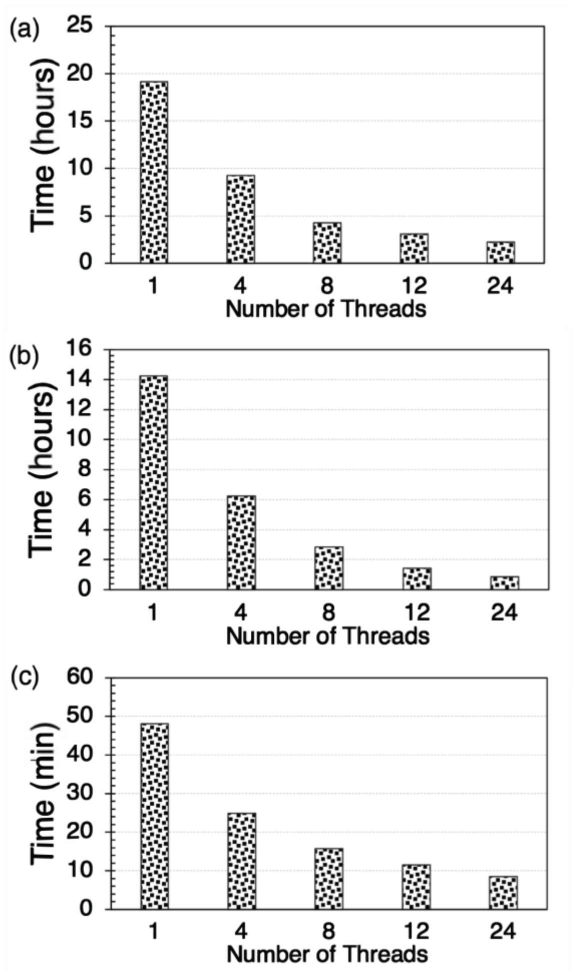


Figure 2. **Scalability of database building process with the number of threads used**. Panels show total wall time (y=axes) for building GTDB genome (nucleotide level) (a), whole-genome proteome (amino acid level) (b) and universal gene set proteome (c) databases. All tests were run on a 24-thread Intel (R) Xeon (R) Gold 6226 processor, with 40GB memory available.

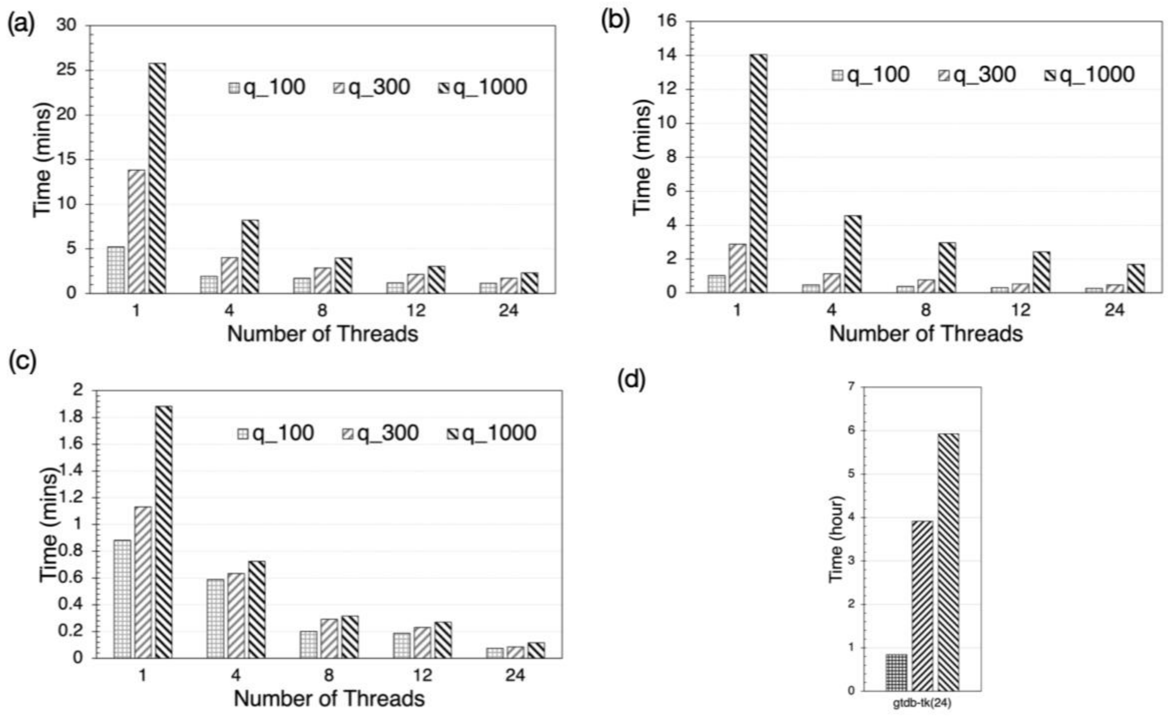
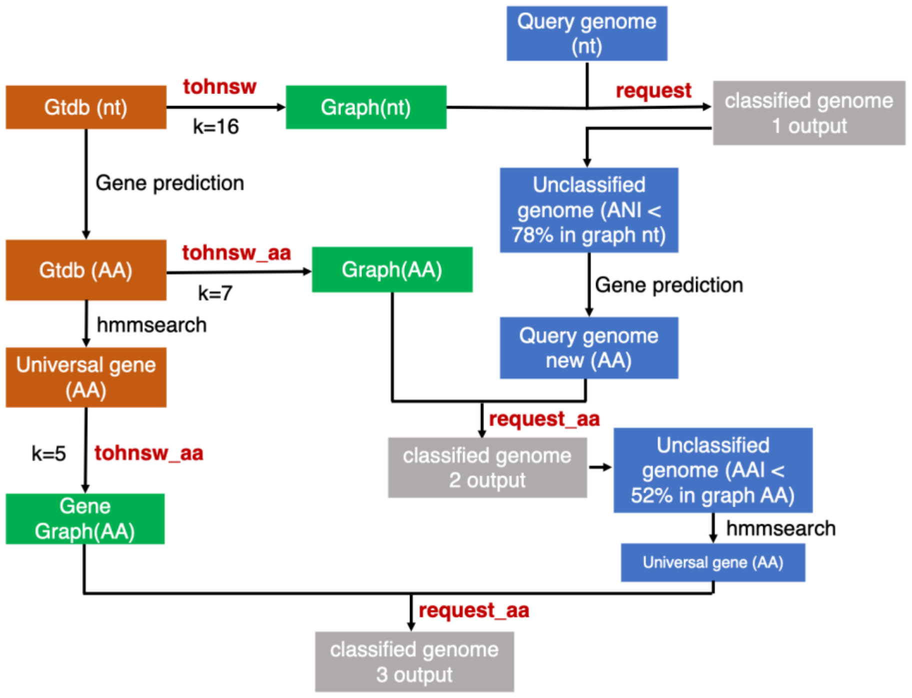


Figure 3. **Total request time (wall time) for searching query genomes against the pre-built reference databases**. Shown are all GTDB genomes (v207) at the whole-genome nucleotide (a), whole-genome proteome (b) and universal gene set proteome (c) levels. 100, 300 and 1000 query genomes (figure key) were used on a 24-thread Intel (R) Xeon (R) Gold 6226 processor. On average, database loading time ranged from 5-10 seconds. (d) is time needed to classify the same genomes using GTDB-Tk on the same 24-thread node.

Figure 4. **Overview of the GSearch pipeline for classifying prokaryotic genomes**. Orange boxes denote steps that aim to prepare genome files, in different formats, for graph building



while green boxes denote building steps of the graph database (in nucleotide or amino acid format). Blue boxes indicate input/query genomes to search against the database while grey boxes indicate classification output for each input. Gene prediction was done using FragGeneScanRs and hmmsearch as part of the hmmer software for homology search. Two key steps of GSearch: tohnsw (aa) and request (aa) are used to build graph database and request new genomes, respectively. Two thresholds are used in the pipeline to decide between whole nucleotide vs. whole-genome amino acid search and whole-genome amino acid vs. universal gene amino acid, 78% ANI and 52% AAI, corresponding to Probminhash distance 0.9850 and 0.9375, respectively (see main text for details).

Tables

Table 1. Request/search performance on major CPU platforms for GTDB v207 database for 1000 queries.

CPU	Number of threads	Clock speed (GHz)	**Request time for nt (min)**	Gene Prediction-FGSrs (min)[c]	**Request time for proteome (min)**	hmmsearch time (min)[d]	**Request time for USCG (min)**
Intel (R) Xeon (R) Gold 6226[a]	24	2.70	**2.329**	1.348	**1.334**	0.524	**0.117**
Intel (R) Core i7-7770HQ[b]	8	2.80	**8.654**	6.764	**2.041**	1.534	**0.510**
AMD EPYC 7513a[a]	32(24 used)	2.60	**1.937**	1.120	**1.021**	0.345	**0.102**
Apple M1 Pro[b]	10	3.22	**2.369**	2.12	**0.866**	0.498	**0.168**

[a] RHEL v7.9, Linux v3.10.0-1160, all threads used.
[b] MacOS v12.3, Darwin 21.4.0, all threads used.
[c] Parallel package was used to run multiprocess at the same time. FGSrs stands for FragGeneScanRs. Note that in practice only those genomes failed in the Request for nt step (best found is less than 78% ANI) will be used in this step.
[d] Only 100 genomes are used for testing hmmsearch because this step is for very new genomes at order level or above and we often do not have that many new genomes in a real-world dataset. Parallel Packages was used to run multiple processes of hmmsearch, one thread per process for hmmsearch.