GSearch: Ultra-Fast and Scalable Microbial Genome Search by Combining K-mer 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 Genomics, 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)

Running title: Approximate nearest neighbor (ANN) search for microbial genomes based on MinHash-like metrics

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 large databases. By combining k-mer hashing-based probabilistic data structures (e.g., (Prob/Super/Densified)-MinHash or SetSketch) to estimate genomic distance, with a graph-based nearest neighbor search algorithm (called Hierarchical Navigable Small World Graphs, or HNSW), we created a new data structure and developed an associated computer program, GSearch, that is orders of magnitude faster than alternative tools while maintaining high accuracy and low memory usage. For example, GSearch can identify/classify 8,000 query genomes against all available microbial or viral genomes (n=~318,000 or ~3,000,000) within a few minutes on a personal laptop, using only ~6GB of memory or less (e.g., 2.5G via SetSketch). Notably, GSearch will be even faster compared to other tools with even larger database size due to O(log(N)) time complexity and will scale well with billions of database genomes based on a database splitting strategy. Further, GSearch implements a three-step classification pipeline that accounts for the degree of novelty of query genomes relative to the database genome to maximize specificity and sensitivity. Therefore, GSearch solves a major bottleneck of microbiome studies that require genome search and/or classification of microbial or viral genomes. GSearch is available at: https://github.com/jean-pierreBoth/gsearch

Keywords: microbial genomes, MAGs, probabilistic data structure, sketching, MinHash, HyperLogLog, nearest neighbor search, classification

45 **Introduction**

46 Identifying or classifying microbial species based on either universal marker genes (e.g., 16S or
47 18S rRNA genes) or entire genomes represents a re-occurring task in environmental and clinical
48 microbiome studies. However, this task is challenging because the microbial genomes in nature
49 are still severely under-sampled by the available genomes. For instance, there are more than 10^{12}
50 prokaryotic and fungal species in nature according to a recent estimation based on 16S rRNA
51 gene or ITS (Internal Transcribed Spacer) analysis (1), and even more viral species (e.g. the
52 number of viral cells outnumbers that of prokaryotic cells by a about a factor of ten in most natural
53 habitats) (2). The number of total prokaryotic genomes (not necessarily completed, named or
54 described biologically) has reached ~318,000 in the newest release of the NCBI/RefSeq
55 prokaryotic database (until Feb. 2023), and > 12 million in the latest IMG/VR4 database for viruses,
56 representing 65,703 prokaryotic and 8.7 million viral distinct species if clustered at the 95% ANI
57 (average nucleotide identity) level (3,4). The number of microbial database genomes is growing
58 at a speed faster than transistors. This has created a new challenge, however; that is,
59 identifying/searching newly sequenced genomes against these large databases to find closely
60 related database/reference genomes for taxonomy classification has become impractical if an all-
61 versus-all comparison strategy is applied. Further, due to the recent improvements in DNA
62 sequencing and single-cell technologies, metagenomic surveys can now recover hundreds, if not
63 thousands, of these yet-to-be-described genomes (or MAGs, metagenomic assembled genomes)
64 from environmental or clinical samples in a single study (5,6), filling in the gap in the described
65 diversity mentioned above but also further exacerbating the database size problem. In addition to
66 the searching strategy, the actual algorithm used to determine overall genetic relatedness (e.g.,
67 ANI or its approximate) between the query and the databased genomes is critical. While the
68 traditional Blastn-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 (7-10), they are too slow to use when dealing with more than a few dozen of genomes. Recently, phylogenetic replacements methods using a handful of universal genes (n ≈ 100) have become popular, but these methods can be memory demanding and slow (11,12), especially for a large number of or a few deep-branching (novel) query genomes. 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 (11,13).

Faster and more memory efficient ANI estimation based on k-mer hashing and evolutionary models have been recently described in tools like FastANI, Mash, Dashing and BinDash (14-18) that aim to alleviate this computational bottleneck. The above-mentioned tools typically rely on probabilistic data structures (or sketching algorithms) such as MinHash (a class of locality sensitive hashing) (19), HyperLogLog (20) or a combination of two, called SetSketch (21) to estimate genomic distance. Importantly, MinHash-like algorithms have been shown to be an unbiased estimation of the Jaccard index $Jaccard(A,B) = \frac{A \cap B}{A \cup B}$ (note that Jaccard index considers only element/k-mer presence/absence in a set, unweighted) between two genomes, an accurate proxy to ANI or mutation rate after appropriate transformation (e.g., $ANI/(1 - Mash) = 1 + \frac{1}{k} log \frac{2*J}{1+J}$, where J is the Jaccard index and k is k-mer size, also called Mash equation) with Poisson distribution assumption and an evolutionary model (14). However, MinHash-like algorithm can be less space efficient (e.g., use more memory) for large number of genomes than

92 HyerLogLog, HyerLogLog++ or SetSketch despite being more accurate (21). The latter algorithms
93 can also be used to estimate Jaccard index after estimating the set cardinality (Supplementary
94 Note 1.) (16). However, Jaccard index (simple, unweighted) estimated by MinHash-like sketching
95 algorithms can be problematic for incomplete genomes due to bottom-k sketch implementation
96 (22) (14), which are commonly recovered from metagenomic surveys. Further, unweighted
97 Jaccard estimate could be similarly inaccurate for genomes with extensive repeats such as
98 several microbial eukaryotic genomes because this estimate does not take into account the
99 abundance of k-mers (k-mer multiplicity). Weighted Jaccard index such as those provided by
100 ICWS and ProbMinHash have been recently developed to address this limitation (23,24) (see
101 also Supplementary Note 1). The application of weighted (or unweighted) MinHash-like algorithms
102 for genome search (that is to search for the closest genomes) problem can still be slow despite
103 the algorithms themselves being fast, because these algorithms are typically applied in a "brute
104 force" manner. That is, all query genomes are searched against all database genomes (i.e., "all
105 vs. all"), and thus computational time grows in a linear fashion as the number of database genome
106 increases. More importantly, in the case of searching a database, locality sensitive hashing
107 property of those probabilistic data structures should be satisfied to ensure high recall or accuracy.
108 One of the most broadly used approaches for finding closely related information to a query,
109 while circumventing an all vs. all search, is the K-Nearest Neighbor Search (K-NNS). The K-NNS
110 approach has been used -for instance-for 16S rRNA gene-based classification followed by a vote
111 strategy (25). Approximate nearest neighbor search (ANNS) algorithms, such as locality-sensitive
112 hashing (LSH) (26), k-dimension tree (27), random projection tree (28), k-graph (29) and proximity
113 graph (30-32) have been recently used to greatly accelerate search processes with only small
114 loss in accuracy. Proximity graph, as implemented for example in the hierarchical navigable small
115 world graph (HNSW), has been shown to be one of the fastest ANN search algorithms(33,34)

116 with search time complexity $O(log(N))$. HNSW incrementally builds a multi-layer structure
117 consisting of a hierarchical set of proximity graphs (layers) for nested subsets of the stored
118 elements. Then, through smart neighbor selection heuristics, inserting and searching the query
119 elements in the proximity graphs can be very fast while preserving high accuracy, even for highly
120 clustered data (31). Therefore, finding the closest genomes in a database can be substantially
121 accelerated by combining two sub-linear algorithms while maintaining ANI/AAI accuracy:
122 MinHash-like or HyperLogLog sketching algorithms for genomic distance estimation and HNSW
123 for finding nearest neighbors. This is an idea that, to the best of our knowledge, has never been
124 applied to genome search despite its potential to greatly accelerate genome search.

125 Here, we describe GSearch (for Genome Search), a computer program that combines
126 one of the most efficient nearest neighbor search approaches (HNSW) with MinHash-based or
127 SetSketch-based universal approaches to measure genetic relatedness among any microbial
128 genomes, including fungal and viral genomes. Four MinHash-like algorithms, Densified MinHash
129 (35,36), ProbMinHash (24), SuperMinHash (37) and SetSketch (21), due to their locality sensitive
130 hashing property for nearest neighbor search, are provided as part of GSearch for the advantages
131 that each provides in terms of accuracy or speed. Densified MinHash, which borrows hash values
132 from non-empty bins to fill empty bins in the sketch vectors, which are generate based on One
133 Permutation MinHash, is by far the fastest MinHash-like algorithm due to just one hash function
134 used, also can be as accurate as classic MinHash. SuperMinHash is similar to classic MinHash
135 but optimized in terms of accuracy to calculate simple Jaccard index, sacrificing speed.
136 ProbMinHash (default) is based on shared k-mers, weighted by their abundance, and normalized
137 by total k-mer count. Accordingly, ProbMinHash can account for genome incompleteness of
138 prokaryotic genomes and repeats (k-mer multiplicity, or k-mer weight) commonly found in
139 eukaryotic and sometimes in prokaryotic genomes. Essentially, ProbMinHash computes the

140 normalized weighted Jaccard-like index J_P (Supplemenatary Note 1) between each pair of
141 genomes. SetSketch is a new data structure aiming at both space efficiency and speed, which
142 fills the gap between MinHash (speed and accuracy, locality sensitive hashing) and HyperLogLog
143 (space). SetSketch can be used for both cardinality estimation and set similarity (1-Jaccard index)
144 and is also a locality sensitive hashing algorithm, providing an advantage over HyperLogLog and
145 MinHash sketching algorithms (e.g., low memory usage and also high accuracy). The Jaccard
146 similarity (1- Jaccard index) calculated by either of the above methods is used as input to build
147 HNSW graph of the database genomes. Accordingly, the search of the query genome(s) against
148 the graph database to find the nearest neighbors for classification purposes becomes an ultra-
149 fast step and can be universally applied to all microbial genomes. The novelty of GSearch also
150 includes a hierarchical pipeline that involves both nucleotide-level (when query genomes have
151 close relatives at the species level) and amino-acid-level searching (when query genomes
152 represent novel species), which provides robust classification for query genomes regardless of
153 their degree of novelty relative to the database genomes, as well as a database-splitting strategy
154 that allows GSearch to scale up well to billions of database genome sequences. GSearch was
155 implemented in the Rust programming language for high performance.
156

157 **Materials and Methods**

158 GSearch is composed of the following steps. Initially, the genetic relatedness among a collection
159 of database genomes is determined based on the sketching algorithm Densifed MinHash,
160 ProbMinHash, SuperMinHash or SetSketch (24,35-37), which compute the normalized weighted
161 Jaccard-like index J_p using the probminhash3a algorithm (ProbMinHash option) or the simple
162 Jaccard index J in the SetSketch/SuperMinHash/Densified MinHash options, as implemented in
163 the probminhash package. The normalized weighted Jaccard-like similarity (1-J_p) or Jaccard

similarity (1-J) is 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) (Figure 1a). Genomes are subsequently recursively added as the nearest neighbors of each node in the built graph file with the same distance computation procedure (Figure 1b). The built graph database and sketches from the MinHash-like algorithm are stored on disk, including a graph file from HNSW as the main component of this new data structure. Each node/genome in the graph has a corresponding sketching in the sketching file (Figure 1c). Query genomes are then searched against graph database after loading the database files and subsequently, best neighbors are returned for classification/identification. In this process, the best neighbor (or neighbors depending on the number of neighbors requested) is also identified based on the smallest ProbMinHash distance ($1-J_p$) or Jaccard distance (1-J) obtained. The output can then be transformed into ANI/AAI values. The details of overall Rust parallel/concurrent implementation can be found in Figure S9. Parallel efficiency and memory usage of database building and searching can be found in Figures S10 and S11.

MinHash-like algorithm Implementation and Benchmark

A detailed description of the differences between ProbMinHash and traditional MinHash can be found in Supplenmentary Materials and Methods. We reimplemented the ProbMinHash algorithm in Rust to estimate genomic relatedness between any two genomes based on normalized weighted Jaccard-like similarity $1-J_p$ according to the original ProbMinHash paper (24)(Supplementary Note 1). The MSE (Mean Standard Error) of ProbMinHash is $J_p(1-J_p)/m$, where m is the sketch size or number of registers, similar to that of classic Jaccard index J(1-J)/m. Essentially, when objects/k-mers are hashed, they represent probability distributions (Relative Frequency of k-mers after normalization), and J_p is a natural extension of J with good

188 properties for estimating distance among various genomes. The Rust reimplementation of
189 ProbMinHash, and other related MinHash-like algorithms, can be found at:
190 https://github.com/jean-pierreBoth/probminhash. We relied on version 0.1.10 of probminhash
191 package for this study. There are 11 different MinHash-like algorithms in this library (all are metric
192 since J and J_p is metric): One Permutation MinHash with Optimal Densification, Faster
193 Densification, SuperMinHash, ProbMinHash1, ProbMinHash1a, ProbMinHash2, ProbMinHash3,
194 ProbMinHash3a, ProbMinHash4, OrderMinHash and SetSketch (locality sensitive hashing
195 estimator (LSH) and Joint Maximum Likelihood Estimator, JMLE). Details of the locality sensitive
196 hashing and JMLE implementation for SetSketch, densified MinHash can be found in
197 Supplementary Methods and Materials. Specifically, when Jaccard index is smaller than 0.01 for
198 queries with their best neighbors/genomes found, we then use JMLE instead of LSH estimator in
199 SetSketch since JMLE is more accurate for small Jaccard.

200 To benchmark ProbMinHash against Mash and Dashing 1/2 et.al., all tools were run with
201 the same sketch size (s=12000) and k-mer size (k=16) for bacterial genomes at the nucleotide
202 level and k-mer size (k=7) at the amino acid level for both database building and searching. For
203 fungal genomes a larger sketch size (s=48000) was used due to much larger fungal genome sizes.
204 Further details on choosing the k-mer size can be found in Supplementary Note 2. For convenient
205 comparison of GSearch results against those of the Mash and ANI based methods, we performed
206 the same transformation of Mash distance from normalized weighted Jaccard like distance J_p to
207 probMASH distance as a proxy of ANI using the equation. Details of the mentioned benchmark
208 softwares can be found in Supplementary file 3.

209 $$probMASH = 1 + \frac{1}{k} log \frac{2 * J_p}{1 + J_p}$$

210

211 *Hierarchical Navigable Small World Graphs (HNSW)*

212 Generally, the framework of graph-based ANN search algorithm (here HNSW) can be
213 summarized in the following two steps: 1) build a proximity graph (HNSW) where each node
214 represents a database vector (or genome profiles in our case). Each database vector will connect
215 with a few of its neighbors while maintaining small world property in each layer of HNSW. 2) Given
216 a query vector (or sequence, kmer profile in our case), perform a greedy search on the proximity
217 graph by comparing the query vector with database vectors under the searching measures (e.g.,
218 cosine similarity or L2 similarity; in our case ProbMinHash distance). Then, the most similar
219 candidates are returned as outputs. The key task for these two-step methods is to construct a
220 high-quality index graph, which provides a proper balance between the searching efficiency and
221 effectiveness. To guarantee the searching efficiency, the degree (number of maximum allowed
222 neighbors, denoted as M) of each node is usually restricted to a small number (normally 20~200)
223 while width of search for neighbor during inserting (denoted as ef_construct) is usually a larger
224 number (higher than 1000) to increase the chance to find best M neighbors by increasing the
225 diversity of neighbors due to the large number of neighbors retained. Graph is subsequently
226 clasped into hierarchical layers following exponential decay probability. Building graph (e.g., insert
227 into graph) and searching query against the graph follow the same greedy search procedure
228 except that there is an extra reverse updating of neighbors list for each vector when inserting
229 database vectors (genomes), one by one, into the existing graph until all genomes are inserted
230 (Figure 1a). For building, after searches are finished at the bottom layer for each inserted element,
231 a reverse update step will be performed to update the neighbor list of each node in the existing
232 graph, while for searching/quering against the graph this step is not needed. The overall database
233 building time complexity is O(N*log(N)), where N is the number of nodes in the graph. The first
234 phase of the searching process starts from the top layer by greedily traversing the graph to find
235 maximum M closest neighbors to the new node element P in the layer by doing *ef*_construct times

search (Figure 1c). 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 1d and e). 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)) as explained above (See detailed complexity analysis on 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 memory safety, efficient parallelism and speed. We also implemented memory map in the newest version, a feature not in the original C++ library that is useful for billions of data points or genomes. The implementation can be found at https://github.com/jean-pierreBoth/hnswlib-rs. Version 0.1.19 was used in this study. Benchmarking of this package against standard datasets and comparison with other implementations can be found in the Supplementary Materials and Methods.

Details of program implementation in Rust

There are 3 modules in total: tohnsw, add and request. Tohnsw is to build the 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 k-mer profile and sketches using ProbMinHash3a algorithm (or SuperMinHash, SetSketch and Densified MinHash) for distance calculation (Figure 1a and c). Next, tohnsw selects, at random, the first batch of genomes to insert into the graph (Figure 1a (1)), following HNSW constructing rules mentioned above and taking into account the computed ProbMinHash distance ($1-J_p$) or SetSketch/SuperMinHash/Densifed MinHash approximated Jaccard distances (1-J) between genomes to connect genomes based on their relatedness (Figure 1b) until all genomes in

260 database have been inserted (Figure 1a (2), (3) and (4)). Finding nearest genomes for the
261 genome to be inserted is essentially a search process but search in a partially built graph, which
262 is similar to the request/search module: whenever a genome is going to be searched against the
263 existing graph, each genome in the graph is associated with a list that stores the M closest
264 neighbors/genomes to the genome and the distance to these neighbors. Then, the distances of
265 this genome with the nearest neighbors (M) of entry genome in each layer will be computed
266 (ef_construct times) using probminhash3a algorithm or SetSketch and the smallest distance of
267 the neighbor genomes will be the new entry genome (Figure 1d and e). This process will be
268 repeated until the nearest genomes (<=M) in the layer are found and subsequently, the program
269 will go to the layer below, using the genome that was represented by the nearest genome in the
270 above layer as new entry genome in the new layer. The search layer algorithm is repeated until
271 the bottom layer is reached/analyzed (Figure 1c). In contrast to the default settings in the original
272 hnswlib, we allow the two parameters of neighbor selecting heuristics, *extendCandidates* to be
273 true and *keepPrunedConnections* to be false because our genomic data is extremely clustered
274 and there is no need to fix the number of connections per element considering the maximum
275 connection allowed. The "Add" module is to add new genomes to pre-built HNSW database using
276 default parameters loaded in the pre-built database files. Request module will load the graph
277 database and then search query genomes against it to return the best neighbors of each query,
278 following exact the same procedure with building step without updating the database. Tohnsw,
279 add, and request modules are all operating in parallel for high performance (see Supplementary
280 Note 6 and Table S9, S10 and S11). GSearch relies on kmerutils v0.0.10 (https://github.com/jean-
281 pierreBoth/kmerutils), which is a Rust package we developed to manipulate genomic fasta files
282 including k-mer string compression, recursive k-mer hashing, k-mer counting, filtering using
283 cuckoo filter. All tests performed in this study were based on GSearch version 0.1.3. Scripts for

284 reproducing the results of this study can be found here:
285 https://github.com/jianshu93/gsearch_analysis. The GSearch package can be found here:
286 https://github.com/jean-pierreBoth/gsearch.
287
288 *Prokaryotic classification pipeline*
289 The amino-acid level graph showed that the closest database neighbors were found, with high
290 recall, when the query shared at least 52% AAI to its best neighbor. For more divergent genomes
291 (<52% AAI), whole-genome amino-acid level graph loses accuracy and we had to switch to
292 universal, single-copy protein-coding genes. For the nucleotide-level graph, we used k-mer=16
293 for bacteria and archaea to have high specificity for closely related database genomes (e.g.,
294 sharing about 95% ANI). For building the whole-genome amino-acid graph, we used k=7 to have
295 the best specificity without compromising sensitivity, which is also consistent with previous results
296 on classification of amino acid sequences based on k-mers (38). For building graph based on
297 universal gene set, we use k=5 because of much smaller total amino acid size. For further details
298 on the range of k-mer to use for bacteria genome and proteome, and viral genome and proteome,
299 see Supplemental Notes 2.
300 The proteome of each genome was predicted by FragGeneScanRs v0.0.1 for performance
301 purposes as opposed to Prodigal software despite small loss in precision (39) (Supplementary
302 Table S11). Hmmsearch in the hmmer (v3.3.2)(40) software was used to extract the universal
303 gene set for bacteria and archaea genomes (universal gene graph). Note that for viral genomes,
304 this last step was not implemented because there are no universal single copy genes for viral
305 genomes. Evaluation of the speed and memory requirements for all steps mentioned above were
306 performed on a RHEL (Red Hat Enterprise Linux) v7.9 with 2.70 GHz Intel(R) Xeon(R) Gold 6226
307 CPU. Unless noted otherwise, all 24 threads of the node were available by default.

308

309 *Distributed implementation and database splitting*

310 To accommodate the increasing number of genomes that become available at an unprecedented
311 speed in recent years and will soon reach 1 million or more microbial genomes, we provide an
312 option to randomly split the database into a given number of pieces and build graph database
313 separately for each piece. In the end, all best neighbors returned from each piece will be pooled
314 and sorted by distance to have a new best K neighbor collection returned to the user for each
315 query genome. We hereby prove that in terms of requesting top K best neighbors, the database
316 split strategy is equivalent to non-split database strategy if the requested best neighbors for each
317 database piece is larger than or equals to requested best neighbors in the non-splitting strategy.
318 The underlying reason is that the best neighbors for a given query globally (non-splitting) are also
319 the best locally for the same piece of database (e.g., best 5 neighbors in piece #1 locally -meaning
320 only in this piece- maybe also be the best n=<5 neighbors genomes globally). The database
321 splitting and request can be done sequentially, on a single node, without multi-node support. In
322 theory, a large database can be split into any number of pieces. In practice, a reasonable way to
323 decide on the number of database pieces to use is so that memory requirement for each piece is
324 equal or smaller than the total memory of host machine.

325

326 *Species database and testing genomes for benchmarking and recall*

327 GTDB version 207 and entire NCBI/RefSeq prokaryotic genomes (as of Feb. 2023) was used to
328 build the database for bacteria and archaea genomes. The IMGVR database version 4, with
329 species representatives at a ≥95% ANI, was used to build the database for viruses (a total of ~3
330 million). For fungal genomes, all genomes downloaded from the MycoCosm project (as of Jan.
331 2022) were used (41). The amino acid sequences of predicted genes on the genomes were

332 obtained using FragGeneScanRs for bacteria/archaea and GeneMark-ES version 2 for fungi
333 (42). The Universal Single Copy Gene (USCG) gene set for GTDB genomes were extracted via
334 hmmer software (40).
335 To test the performance of our pipeline, we specifically chose genomes that are not included in
336 the GTDB database (the database was used for graph building). In particular, the
337 bacterial/archaeal genomes, mostly MAGs, reported by Ye and colleagues (43) and Tara Ocean
338 MAGs (total 8,466 MAGs) (44) were used. We randomly selected 1000 genomes/MAGs from
339 Ye's collection and use them as query genomes to test the performance and accuracy of
340 GSearch.
341
342 *Recall of AAI-, ANI- and MinHash-based nearest neighbor searching for bacteria/archaea, fungi*
343 *and viral genomes.*
344 To benchmark GSearch performance compared to traditional ANI/AAI- and/or MinHash-based
345 tools, we ran brute-force Blastn-ANI/FastANI (MUMMER for fungi ANI), Blastp-based AAI to find
346 the best neighbors for the same query genomes in different dataset and evaluated whether the
347 best neighbors found by GSearch were the same as those found by Blastn-ANI/FastANI and
348 blastp-based AAI. Briefly, we used the well-known average recall as the accuracy measurement
349 (45). Given a query genome, GSearch is expected to return k best genomes. Then we examined
350 how many genomes in this returned set were among the true k nearest neighbors (NN) of the
351 query found by the reference brute-force Blastn-ANI and/or Blastp-based AAI. Suppose the
352 returned set of k genomes given a query is R' and the true k nearest neighbors set of the query
353 (from Blastn-ANI or blastp-based AAI) is R, the recall is defined as:
354 $recall(R') = \frac{|R' \cap R|}{|R|}$. Then, the average recall represents the mean of all the query

355 datasets/genomes. We compare the accuracies of different algorithms by requiring different
356 number of nearest neighbors (NN) of each query genome, including 10-NN and 5-NN, using the
357 same approach. Since biological species databases are generally sparse due to the
358 undersampling of natural diversity and the existence of natural gaps in diversity among species,
359 a larger top k NN (e.g., 100) used in standard ANN benchmark experiments will offer little, if any
360 biological advantage, especially when the query genomes are relatively new, e.g., a new family
361 compared to database genomes. Therefore, we use top 5-NN and 10-NN. Further, if the genomic
362 distance (J_p) of query to some of the top 10 or top 5 neighbors found by GSearch at the nucleotide
363 level was larger than 0.9850 for bacterial genomes, these matches were filtered out and not used
364 in estimation of recall, because we have shown that above this threshold, k-mer based MinHash
365 methods at nucleotide level will lose accuracy and this is not related to HNSW search itself (e.g.,
366 8 out of 10 are kept, so only top 8 in R' is compared with top 8 in R). At proteome level, accuracy
367 was calculated following similar rules with the threshold value being 0.9720 (switching to the
368 universal gene graph at this level). For viral genomes, the threshold was 0.9800. Those thresholds
369 were chosen based on correlation with blast-based ANI or AAI. Detailed commands for running
370 each software mentioned above can be found in supplementary file 3.
371

372 **Results**

373 *ProbMinHash as a robust metric of genome relatedness for prokaryotic genomes*

374 Correlations between ProbMASH (we called it ProbMASH, after transformation from
375 ProbMinHash distance (J_p), see Materials and Methods for details) and ANI (determined by
376 FastANI) or Mash distance showed that ProbMinHash is robust and slightly better than Mash for
377 determining distances among bacterial genomes related at ~78% ANI, or above, i.e., genomes
378 assigned to the same or closely-related species (Spearman rho=0.9643 and 0.9640 for

379 ProbMinHash and Mash values against the corresponding ANI values for the same genome
380 comparisons, respectively, *P*<0.001, Figure S1a and S1b). For moderately related genomes, for
381 which ANI based on nucleotide k-mer is known to lose accuracy, ProbMASH (ProbMinHash)
382 based on amino acid k-mer was robust compared to Blast-based average amino acid identity or
383 AAI, especially among genomes showing between ~52% and 95% AAI (Spearman rho=0.90,
384 *P*<0.01, Supplementary Figure S2a and S2b). Below ~52% AAI, both ProbMinHash and Mash
385 distance lose accuracy compared to AAI. However, AAI of just universal genes provides a robust
386 measurement of genetic relatedness at this level of distantly related genomes, and we show here
387 that ProbMASH (from ProbMinHash distance) for this set of universal genes is also robust
388 (Spearman rho=0.9390, *P*<0.001, Supplementary Figure S3). Thus, for query genomes of
389 organisms with only distant relatives in the database (i.e., deep branching), for which their closest
390 matching genome in the database is related at the order level or higher, restricting the search to
391 the universal genes can provide robust classifications (11). Accordingly, GSearch implements a
392 three-step classification process, depending on the degree of novelty of the query genome against
393 the database genomes, using the AAI thresholds mentioned above (see also Fig. 2g). This
394 strategy and its accuracy are discussed further below. To show the accuracy of One Permutation
395 MinHash with Optimal Densification (Densifed MinHash), we calculated ANI by both Mash (classic
396 MinHash) and our implementation, we showed that they are almost identical (Figure S12).
397
398 *Comparisons of sequence search algorithms and MinHash-like algorithms*
399 We compared GSearch with other general-purpose sequence or genome search algorithms,
400 focusing on time complexity (big O annotation) of each sequence or genome search algorithm.
401 GSearch was, by far, the fastest genome search algorithm, with a O(log(N)) complexity (Table 1).
402 Other general-purpose sequence search algorithms (e.g. Sequence Bloom Tree, COBS et.al.)

403 are either sublinear under certain assumptions or not practical in real world applications. Also,
404 none of the general-purpose sequence or genome search algorithms were previously
405 benchmarked against an ANI-like distance; hence, their accuracy remains untested.

406 We then also compared available MinHash-like sketching algorithms in terms of their time
407 complexity (big Oh), space (memory), variance, and other important properties for large scale
408 applications such as mergeability (whether sketching part of dataset and then merge is equivalent
409 to sketching the entire dataset) to estimate Jaccard-like index (Table S1 and S2). SuperMinHash
410 and ProbMinHash are fast and mergeable MinHash algorithms, slower only than One Permutation
411 MinHash with Optimal Densification (as implemented in BinDash) (Table S1). The latter algorithm,
412 however, is not mergeable thus difficult to apply in a distributed computational environment.
413 HyperLogLog-like algorithms are significantly more space efficient albeit slightly slower. For
414 variance, we analyzed the rooted mean square (RMSE) error of all MinHash-like sketching
415 algorithms mentioned above. MinHash-like algorithms such as classic MinHash, ProbMinHash,
416 SuperMinHash had the smallest theoretical variance as m (the number of registers used for
417 sketching) increases (we use m=12,000, note that J*(1-J) is much smaller than 1 for all Jaccard),
418 following by Densified MinHash, SetSketch (as b→1) (Table S2). Our implementation of the
419 Densified MinHash has almost close RMSE with classic MinHash (Figure S13). SetSketch is also
420 space efficient and is similar to that of HyperLogLog in terms of space and speed. HyperLogLog
421 sketch estimators such as those implemented in Dashing has largest variance for estimating the
422 cardinality of sets (followed by inclusion-exclusion rule to estimated Jaccard index,
423 Supplementary Note 1), consistent with experimental results when searching query genomes
424 against database (Supplementary Table S14 and S15), especially for small Jaccard distance (e.g.,
425 75% to 78% ANI, corresponding to 0.009 and 0.015 Jaccard index), where variance is more
426 important for accuracy.

427

428 *Graph building and speed of search against reference prokaryotic genomes*

429 To build the database graph for the entire GTDB v207 database at the nucleotide level (65,703
430 unique, non-redundant prokaryotic genomes at the 95% ANI species level), the tohnsw module
431 (database build subcommand of GSearch) took 1.3 h on a 24-thread computing node and scaled
432 moderately well with increasing number of threads (0.27 h using 128 threads) (Figure 2a).
433 Maximum memory (RAM) required for the building step was 15.3 GB. The total size of written
434 database files on disk was ~3.0 GB. There are 3 layers for the resulting graph, with 65180, 519,
435 and 4 genomes for layer 0,1 and 2 respectively. The searching of query genomes against the
436 GTDB database graph, requesting best 50 neighbors for 1000 query genomes, which represented
437 different previously known as well as novel species of eight bacterial phyla (see Methods for
438 details on query genome selection), took 2.3 min (database loading was 6 s) on a 24-thread
439 machine and scaled well with increasing number of threads (Figure 2d). The memory requirement
440 for the request (search) step was only 3.0 GB for loading the entire database file in memory.

441

442 We also built a database of all NCBI/RefSeq prokaryotic genomes (~318K, 2 Terabytes in total),
443 which took 4.1 hours using 24 threads with maximum memory usage of ~21GB (1.2 h with 128
444 threads) and dumped database file size of 15GB (30GB for amino acid). Searching of 8,466 query
445 genomes against this RefSeq database took 9.32 min (Table 2) and ~16GB of memory, which
446 was significantly better than alternative state-of-the-art tools for the same purposes such as
447 Dashing1/2 and BinDash (brute force HyperLogLog or MinHash), which took 28/42 min and 21
448 min, respectively (Table S3, Figure S5). To evaluate the performance against Dashing or
449 Sourmash more fully, we increased the number of database genomes gradually from 60K to 318K
450 (all RefSeq prokaryotic genomes, 2 Terabytes) while using the same number of query genomes

451 (8,466). We observed that GSearch follows a log fitting, consistent with the O(log(N)) theoretical
452 prediction, while Dashing 1/2 and Sourmash follow linear fitting, and thus the latter tools will be
453 increasingly slower than GSearch as the number of database genome increases (Figure 3a and
454 c). For example, GSearch was 3 to 4 times faster than Dashing 1/2 and skani when the number
455 of database genomes increased from 65,703 to 318,000 (Figure 3a, Table S3) and 20 to 30 times
456 faster with the 3 million reference virus genomes (Figure 3b, Table S4. see also below). The
457 speedup will be even larger as number of database genomes continues to grow. With Densifed
458 MinHash as option, GSearch can be 2-3 times faster than ProbMinHash (Table S3)
459
460 To build the amino-acid-level graph for moderately related query genomes, all GTDB v207
461 genomes were used for gene calling by FragGeneScanRs and subsequently, the predicted amino
462 acid sequences for each genome were used for building the database graph. The graph building
463 step took 1.4 h (Figure 2b) with a maximum memory required to be 37.7 GB on the 24-thread
464 node. The total size of the written database files on disk was 5.9 GB. However, when the
465 SetSketch option was used, graph building step took 2.3 h but only 9.4GB maximum memory
466 needed, with the database size being 0.5GB (Figure 3 (d)). There were 65158, 543 and 2
467 genomes for layer 0,1 and 2 respectively. The entire RefSeq genome database (~318K) took 3.7
468 hours to build at the proteome level with similar maximum memory, dumped files size was 29GB
469 (only 2.5GB with the SetSketch option). Requesting 50 neighbors for querying 1000 genomes at
470 the amino-acid level took 1.52 minutes with a memory requirement of ~6.0 GB (database loading
471 9 seconds; Figure 2e) for the GTDB database. Requesting 50 neighbors for 8,466 query genomes
472 against the NCBI/RefSeq proteome graph, took about 6.2 min (Table 2). In comparison, Mash
473 dist took 207.2 min against NCBI/RefSeq with 24 threads for the same task (no amino acid option
474 for Dashing).

475

476 Finally, for most distantly related query genomes, the graph building for the universal gene set
477 follows the same logic as the amino acid level graph mentioned above except for using a smaller
478 k-mer size (k=5) due to the smaller k-mer space of ~120 universal genes vs. the whole-genome
479 level. It took 7.76 min to build the database (Figure 2c) and 12 seconds to request 50 neighbors
480 for 1000 queries on a 24 threads node (Figure 2f) against the GTDB database graph. Building for
481 the entire NCBI/RefSeq database took about 27 min while requesting 1,000 against it took about
482 0.445 min (Table 2).

483

484 *Searching accuracy and speed for prokaryotic genomes*

485 The accuracy of GSearch and other tools in identifying the best matching genomes among the
486 database genomes was evaluated by comparing against the best matching genomes for the same
487 query genome(s) identified by traditional alignment-based methods, namely Blast-based ANI for
488 query genomes with close relatives in the databases and Blast-based AAI for query genomes
489 distantly related to their best match in the database. K-mer-based genomic distance (e.g. all tools
490 mentioned above) is widely known to be less accurate with higher degree of novelty of the query
491 genomes relative to the database genomes as also explained in Supplementary Note 2 and 4. To
492 address this limitation, GSearch employs a three-step framework, calculating genomic distance
493 at nucleotide, whole-genome amino acid (proteome) and universal gene amino acid levels for
494 query genomes of increasing novelty (or genetic distance) to database genomes as described
495 above (Fig. 2g). For query genomes from two datasets, the Tara Ocean (8,466), and the collection
496 put together by Ye and colleagues (997), that had closely related genomes in the RefSeq
497 database (showing >78% ANI, 6,992 and 906 query genomes, respectively), we observed an
498 average top 10 recall of 96.2% and 95.1%, respectively. Recall was estimated by comparing

499 (meaning % overlap) GSearch's top-10 matching genomes to top-10 found by Blastn-ANI (similar
500 results were obtained for top-5 matching genomes; Table 3). For query genomes with no closely
501 related genomes in the database showing higher than 78% ANI, 1474 and 91 genomes from the
502 two datasets, respectively, recall was significantly lower, only around 60% or below (Table 3).
503 However, when we search these genomes at the proteome level using Blastp-AAI as the
504 reference standard for calculating recall, the recall value was 96.9% and 95.2%, respectively
505 (Table 3). There were 327 and 25 genomes, respectively, that, even at proteome level, they had
506 no related genome in the RefSeq database showing higher than 52% AAI and these genomes
507 accounted for most of differences with the top-10 matching genomes by blast-AAI. These
508 genomes apparently reflect the fact that the public genome databases are still sparse and do not
509 cover well all biological diversity. When we searched these deep-branching (highly novel)
510 genomes at the universal gene proteome level, using the universal gene AAI as the reference
511 standard, we found the average top-10 recall value of 95.5% and 94.6%, respectively (Table 3).
512 We also compared these results obtained by GSearch to those of BinDash, Dashing 1/2 and
513 Sourmash for the same query genomes. We found that Sourmash is as accurate as GSearch but
514 much slower, while Dashing is less accurate than GSearch and Sourmash or BinDash, especially
515 when the closest database genomes showed less than 80% ANI to the query genomes
516 (moderately or distantly related) (Table S14 and S15). The results were similar independent of
517 the estimator used in Dashing (e.g., MLE or JMLE), consistent with theoretical predictions (Table
518 S2). BinDash was as accurate as Mash but much faster as previously described (Table S1 and
519 Table S13). We provide a theoretical derivation based on k-mer conservation and empirical
520 results for deriving the 78% ANI and 52% AAI thresholds to decide the switching between whole-
521 genome nucleotide vs. amino acid (proteome) search and whole-genome proteome vs. universal
522 gene only, respectively, in Supplementary Notes 2 and 4. Detailed accuracy results for all tools

523 based on a single query genome (not average recall across all query genomes as shown above)
524 can be found in Supplementary Table S13, S14 and S15.
525
526 We also evaluated the effect of genome completeness on the accuracy of GSearch given that
527 bacterial genomes recovered from environmental metagenomic surveys are frequently
528 incomplete. GSearch was robust to genome incompleteness down to 50% completeness level.
529 We found that at genome completeness higher than 50%, the accuracy/recall in the top 10 best
530 matches was higher than 80% and decreased considerably below this completeness level (Table
531 S12). Hence, GSearch analysis is not recommended for genome completeness less than 50%
532 due to the normalization step in ProbMinHash.
533
534 *Graph database building and searching accuracy for viral and fungal genomes*
535 Graph building and requesting for viral genomes is not effective at the nucleotide level because
536 many viral genera are genetically too distant from each other and do not have close relatives in
537 the public genomic database; that is, the database is too sparse. Accordingly, k-mer-based
538 methods for genomic distance will often lead to imperfect graph structure for viral genomes.
539 Therefore, we built only an amino acid level graph for viral genomes, using all genes in the
540 genome due to the lack of universal genes for viral genomes. Database building took 13.89 h for
541 all ~3 million IMG/VR4 genomes on a 24-thread node and graph file on disk was 15.8 GB (Figure
542 S6a). Requesting 1000 top neighbors scaled well with increasing number of threads and took
543 about 3.63 min (database loading took an additional 1.1 min) using 24 threads (Figure S6b). The
544 top-5 neighbors for 1000 query phage genomes were highly overlapping (98.32% recall; Table
545 S6) compared with the top-5 neighbors of the brute-force AAI approach. For such large databases
546 in terms of number of genomes, GSearch is about 20X faster than the brute-force Mash and

547 Dashing tools in finding the top-5 neighbors (Figure 3b, Tables S4). We also compared GSearch
548 with a new database building method, called PhageCloud, which relies on manually curated
549 genome labels (e.g., environmental source) for graph database building using the Neo4j software
550 and Dashing for genetic distance computation. Since PhageCloud provides only a web
551 implementation with one genome query at a time, we limited the search to one viral query genome
552 against the same database used by PhageCloud i.e., Gut Phage Database [38]. It took 37 seconds
553 to find the two best matches with PhageCloud while GSearch took 15 seconds (database loading
554 14 seconds, search 1.5 second) for the same search. Mash, on the other hand, took 4 minutes to
555 find the same two best matches. It should be noted, however, that, because the database is
556 already available (loaded) on PhageCloud's website, 37 seconds is only for the searching step
557 and website responses (average value for 5 runs on 5 different days) whereas GSearch took only
558 1.5 second for this same step.
559
560 Graph building for fungal genomes is slower compared to prokaryotic genomes, despite the
561 smaller number of available fungal genomes (n=9700), because the average fungal genome size
562 is much larger and k-mer and sketch size are accordingly much larger (k=21, s=48000). It took
563 2.3 h on a 24-thread node to build the nucleotide level graph for these fungal genomes. Searching
564 step was also slower due to the larger k-mer space. Accordingly, it took 3.13 min to request 50
565 neighbors for 50 query fungal genomes while Mash tool 4.4 min. Nonetheless, top 10 recall was
566 still very high (~99.4%) against Mash and MUMMER-based ANI for the same datasets. For the
567 amino acid level graph, the time for graph building was only 0.61 h, shorter than the corresponding
568 prokaryotic graph. Identifying 50 neighbors for 50 query fungal genomes at the amino-acid level
569 took 1.24 min (Mash took 2.59 min) with similarly high top 5 and top 10 recall (99.7% and 98.5%,
570 respectively) against brute-force Mash and Blastp-based AAI. Note that the difference in run time

571 will be much larger between Mash and GSearch as the number of fungal database genomes
572 increases in the future, as also exemplified above for the bacterial genomes.
573
574 *Combining the three-step framework for prokaryotic genome classification and comparison with*
575 *standard tool GTDB-Tk*
576 To evaluate the accuracy of classification/searching results based on the complete 3-step
577 framework of GSearch, we compared the best neighbors found by GSearch with brute-force ANI
578 (estimated by FastANI) and GTDB-Tk, a standard for microbial taxonomy classification these days
579 that is phylogenetic-tree- and alignment-based. As mentioned above, when the query genome
580 does not find a match in the database better than ANI > 78%, corresponding to ProbMinHash
581 distance 0.9850, the nucleotide-level graph is abandoned, and the amino-acid level is used
582 instead; similarly for no match better than 52% AAI, GSearch switches to the universal gene graph
583 (Figure 2g). The overall running time to classify 1000 prokaryotic genomes of varied levels of
584 taxonomic novelty compared to the database genomes on different computing platforms is
585 showed in Table S5. On a 24-thread Linux node with Intel Xeon Gold 6226 CPU, it took GSearch
586 a total of 5.85 minutes while it took 19.49 minutes on an intel Core i7 laptop (2017 release) CPU
587 personal laptop (6.02 minutes on the most recent ARM64 CPU laptop). Classifying 1000 genomes
588 using GTDB-Tk took 5.91 h on the same Linux node with 24 threads (memory requirement was
589 ~328GB) while MASH took 53.7 min for 1000 genomes using 24 threads. We were not able to
590 assess Dashing or BinDash for this analysis because they do not provide proteome (amino acid)
591 level implementation.
592
593 In terms of classification accuracy, all query genomes that had a best match higher than 78% ANI
594 against the GTDB database genomes (i.e., a match at the same or closely related species, 699

595 out of the total 1000 queries) were identically classified by GSearch, GTDB-Tk and FastANI
596 (Supplementary file 1, only 100/699 are shown for simplicity), which means species-level
597 classifications are highly consistent. For the remaining 301 genomes that did not have same or
598 closely related species-level matches, for 266 of them (or 87.1%), GSearch also provided the
599 same classification with GTDB-Tk but several inconsistencies were observed for 39/301 genomes
600 (Figure 3e). Specifically, we noticed that for GTDB-Tk, which relies on RED values and tree
601 topology, several genomes (n=14) were still classified at the genus level even though the AAI
602 value against the best database genome in these case was below 60% (typically, genomes
603 assigned to the same genus show >65% AAI(46)), and some genomes (n=16) were still classified
604 at the family level but not at the genus level even though their best AAI value was above 65%.
605 Similarly, several genomes (n=9) were classified at the order level but not family level even though
606 their best AAI value was above 52%. Therefore, high consistency was overall observed between
607 GSearch and GTDB-Tk assignments, and the few differences noted were probably associated
608 with contaminated (low quality) MAGs or taxonomic inconsistencies, which was challenging to
609 assess further, and/or the peculiarities of each method. Since ProbMinHash distance correlated
610 well with blastp-based AAI after transformation in the range of AAI values between 52% and 95%,
611 the classification results were always consistent with AAI-based classification using previously
612 proposed thresholds. For example, best matches at AAI 65% ≥ AAI were classified in the same
613 genus by GSearch and blast-based AAI, and best matches of 52% < AAI < 65% were typically
614 classified in the same family (but different genus within the family).
615
616 *Database split for large genome databases*
617 For large databases (for example, >1 million bacterial genomes), the graph building and
618 requesting step could require a large amount of memory (due to the larger k-mer space) that is

619 typically not available in a single computer node. We therefore provide a database split solution
620 for such large databases. The average database building time on each node for each piece of the
621 database after the splitting step scales linearly with increasing nodes/processors (Figure S7a)
622 and requires much less memory (1/n total memory compared to when building in one node, where
623 n is the number database pieces after splitting; for GTDB v207 nucleotide graph building and n=5,
624 it will be only 28.3/5=5.66 GB). The searching time scales sub-linearly with increasing number of
625 nodes (Figure S7b) but offers the advantage of a reduced memory footprint with respect to the
626 single-node search. The top 10 best neighbor by splitting the database were exactly the same as
627 the non-splitting strategy (Supplementary file 2). Note that without multi-node support (e.g., run
628 database build sequentially), database build time is nearly the same with non-split strategy, but
629 memory requirement is only 1/n (GTDB v207, 28.3GB/5=5.66GB at nucleotide level and
630 27.7GB/5=5.54GB at amino acid level), even though total request time will be longer (time*n in
631 Figure S7(b)). However, since the request step is very fast, requiring only 1/n memory compared
632 to non-split strategy, overall runtime is still short with the database split approach even for a
633 relatively large number of genomes. The database split strategy is especially useful when memory
634 requirement is not satisfied on the host machine for larger genome database (e.g., millions of
635 genomes).
636
637 **Discussion**
638 A popular way to assess genetic relatedness among genomes is ANI, which corresponds
639 well to both 16S/18S rRNA gene identity and DNA-DNA hybridization values, the golden
640 standards of fungal and prokaryotic taxonomies (7). However, the number of available microbial
641 genomes has recently grown at an unprecedented speed. For example, there are 30% more
642 (new) species in NCBI/RefSeq 2022 compared to 2023, and the number of bacterial species

643 represented by genomes alone is expected to surpass 1 million soon. Therefore, the traditional
644 way that blast-based ANI or faster k-mer-based implementations (e.g., Skani, FastANI, Mash or
645 Dashing 1/2) are applied as an all vs. all search strategy (brute-force) does not scale because the
646 running time grows linearly with increasing number of query genomes and/or genomes in the
647 database. Phylogenetic approaches based on quick (approximate) maximum likelihood
648 algorithms and a handful of universal genes as implemented -for example- in GTDB-Tk could be
649 faster than brute-force approaches but are often not precise and require a large amount of
650 memory for the querying step (11) while the database building step could take several weeks of
651 run time because the underlying multiple sequence alignment of the database genomes is
652 computationally intensive. Further, approaches that reply on k-medoid clustering to avoid all vs.
653 all comparisons could be sometimes trapped into local minima because of arbitrary partitioning of
654 database genomes into clusters, a known limitation of these methods (13). GSearch effectively
655 circumvents these limitations by combining new k-mer hashing-based probabilistic data structures
656 for fast computation of genomic relatedness among genomes (i.e., ProbMinHash, SuperMinHash
657 and/or SetSketch) with a graph based nearest neighbor search algorithm (HNSW). Accordingly,
658 GSearch is at least an order of magnitude faster than alternative approaches (e.g., Mash,
659 Dashing, Sourmash, GTDB-Tk) for the same purposes and will be even faster as the number of
660 database genomes grows. Therefore, GSearch solves an important challenge associated with the
661 tasks to search and/or classify microbial genomes and will serve well these tasks for years to
662 come based on the current speed in the increase of the number of genomes that become
663 available.
664
665 To the best of our knowledge, no current tools have combined two sub-linear algorithms
666 and thus efficiently search very large genome databases while maintaining high accuracy in

667 calculating ANI/AAI among the genomes, and hence identifying the true best neighbors. Several
668 sub-linear algorithms such as Sequence Bloom Tree or COBS (cross-over between an inverted
669 index and Bloom filters) can do sequence to sequence (or genome to genome) search but do not
670 provide ANI/AAI values, which are key for microbial genome search and classification (Table 1).
671 Accordingly, these tools are generally less accurate than the ANI/AAI approach. GSearch can
672 handle several million of microbial genomes on a small-to-average computer cluster, or even
673 personal laptop (depending on the database size), since the created (dumped) database file size
674 is proportional to the total number of genomes in database for fixed sketch size and graph
675 parameters, and generally rather small. Specifically, with one million genomes, the dumped file
676 size will be 5.9GB*20=118 GB (currently, there are ~60K in the GTDB database, creating a
677 database file size of 5.9GB). With the SetSketch option, database file size will only be 0.5GB for
678 the GTDB database, 2.5G for the entire RefSeq genome databases (~318K genomes, 2
679 Terabytes), and 9.8GB for one million genomes. Further, due to the nature of the graph based
680 NNS algorithms, there is no need to build the entire database at once, but the database can be
681 split it into smaller pieces, as exemplified above and depending on the computational resources
682 available. For a modern laptop with 16 GB memory, a database on one million species can be
683 split into 10 pieces, so the dumped file for each piece will be only 11.8 GB, which can be loaded
684 into memory, and then collect the results from each piece within an approximate total running time
685 of 30 minutes (for querying 1000 genomes against 0.1 million database genomes for each of the
686 ten parts, which should take about 3 minutes). With this logic, a computing node with 24 threads
687 and 256 GB of memory available can easily deal with 20 million bacterial database genomes and
688 billions of bacterial genomes if using SetSketch. This represents a substantial improvement
689 compared to existing tools for the same purposes. This database split/partition idea of HNSW, or
690 other graph-based NNS libraries, has been successfully applied in several industrial-level

applications for other purposes (31,32). We also want to point out that the size of the compressed sketches (database file) from SetSketch is comparable to other space efficient algorithms such as HyperLogLog (with b=2, Setsketch asymptotically corresponds to HyperLogLog), ExtendedHyperLogLog (47), HyperLogLogLog (48) and UltraLogLog (49) (25% more space efficient than HyperLogLog). The Shannon entropy of SetSketch we implemented is $\frac{m}{ln(2)ln(b)}\left(\left(1-\frac{1}{b}\right)+\int_0^1 z^{1/(b-1)}\frac{(1-z)ln(1-z)}{zln(z)}dz\right)$, where m is the number of registers, b is the key parameter of SetSketch. For b=0.0005 that we used, the entropy is 13.24 per register (divided by m). Therefore, compared to an uncompressed register size of 16 bits, there is still some room for improvement for SetSketch. Theoretically, a 17.25% reduction in space can be achieved but would add additional implementation complexity. For any HyperLogLog-like algorithms (e.g., SetSketch, UltraLogLog), a theoretical limit for sketch size is $O(\epsilon^{-2}+log(n))$, or slightly worse but easy to implement (50,51), where n is the number of elements/k-mers while є is error. This means that we cannot improve further without losing accuracy as database size grows unless we rely on database split strategy. SetSketch shares this property with HyperLogLog and thus we cannot improve space further without theoretical breakthroughs.

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. The high efficiency in building graphs allows users to also test and optimize the key parameters of the graph, the M (maximum allowed neighbors) and ef_construct (width of search for neighbor during building) 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

714 large M and ef_construct may lead to slow speed without proportional improvement in recall
715 (Table S7). Therefore, there is a tradeoff between accuracy and speed that should be evaluated
716 first. However, for most users this task would not be necessary because they will work with pre-
717 built databases such as those provided here. Also, GSearch provides an option to add new
718 genomes to the database without the need to rebuild the database from scratch, making it
719 convenient and very fast to expand the current genome database as needed.

720 GSearch could also be applied to whole metagenome search and identification of the most
721 similar metagenomes in a series because the relatedness among metagenomes can be estimated
722 in a similar way to genomes using ProbMinHash as implemented -for example- in the HULK
723 software, which relies on the ProbMinHash algorithm in a streaming fashion (52). However, to
724 allow for sketching of much larger sequence files such as whole metagenomes simultaneously to
725 build the HNSW graph, weighted k-mer (e.g., ProbMinHash) approaches require much larger
726 memory than unweighted approaches due to the larger number of total k-mers (including
727 duplicates). Probabilistic or approximate counting of k-mer abundances such as Count-Min sketch
728 can be used instead for k-mer counting for metagenome search and/or when the computer
729 memory is limited (53,54) , or a combination of Count-Min Sketch with HyperLogLog sketch (55).
730 Similarly, we could seamlessly replace ProbMinHash with another relatedness algorithms should
731 such algorithms become available and has advantages in terms of speed and/or precision, or
732 more importantly, has the locality sensitive hashing property for nearest neighbor search. For
733 example, the SuperMinHash option is now provided for its high accuracy over traditional MinHash
734 but not faster (37), or other newer implementations such as the One Permutation MinHash with
735 Optimal Densification (OPMHOP) for estimating the unweighted Jaccard index, due to its speed
736 (only one hash function, average case O(n+m)) and locality sensitive hashing property (via
737 optimal densification or faster densification)(35,36) (Table S1 and S2). Note that the Bottom-K

738 sketch implementation in Mash or FracMinHash in Sourmash use just one single hash function
739 or a much smaller number of hash functions required by that of classic MinHash and thus lose
740 the locality sensitive hashing property and thus we cannot use it for nearest neighbor search
741 purposes (36). Further, BagMinHash (56) or DartMinHash (57) for weighted (but not normalized)
742 Jaccard index can also be used in place of ProbMinHash (also LSH). Since the number of
743 genomic distance computations is O(log(N)) in GSearch, the computational time for estimating
744 genomic distance for a pair of genomes is not a major factor in overall computational speed
745 because log(N) is always a small number. Accuracy in the genomic distance estimate is relatively
746 more important in GSearch and the main reason that ProbMinHash is the default option. Related
747 to this, ANI as currently implemented -for instance- in FastANI is not appropriate for this function
748 because it is not metric; that is, for the FastANI distances calculated among three genomes A, B,
749 and C, (A,B) + (B, C) is not necessary larger than (A,C) and ANI(A,B) is not equal to ANI (B, A)
750 [15]. Similarly, mutation rates (or ANI) estimated by FracMinHash, CMash and related tools (e.g.
751 Sourmash or skani calculated ANI) are also not metric (58-60). To solve this "metric" problem, a
752 norm adjusted proximity graph (NAPG) was proposed based on inner product and it shows
753 improvements in terms of both speed and recall using non-metric distances (61). This could be
754 another direction for further improving the speed and recall of GSearch. The three options we
755 provided as part of GSearch to estimate Jaccard-like index are all metric, which ensures neighbor
756 diversity when building the graph, and they are equally applicable to any microbial genome,
757 including viral and fungal genomes.

758 K-mer-based methods for genetic relatedness estimation such as ProbMinHash have
759 lower accuracy between moderately-to-distantly related genomes compare to alignment-based
760 tools (see Supplementary Note 2 and 4 for further discussion). Our empirical evaluation showed
761 that this relatedness level, for nucleotide searches, is around 78% ANI and 52% AAI for the amino-

762 acid searches (e.g., ProbMinHash distances do not correlate well with blast-based ANI and AAI
763 at and below these levels). To circumvent this limitation, we designed a 3-step framework as part
764 of GSearch to classify bacterial genomes that show different levels of novelty compared to the
765 database genomes, with high accuracy. This framework included a search at the universal gene
766 level for deep-branching genomes that are novel at the phylum level (AAI < 52%), for which
767 searching at the entire proteome level is less accurate. Recently, methods that employ k-mers
768 that allow mismatches, that is, spaced k-mers have shown promise in accurately estimating
769 genomic relatedness even among distantly related genomes with gains in speed (62). To apply
770 spaced k-mers to entire genomes, the recently developed "tensor sketch" approaches could be
771 explored in the future to simplify the distance computation for distantly related bacterial and viral
772 genomes (63) instead of relying on the mentioned 3-step framework. In the meanwhile, the
773 ProbMinHash approach, essentially a Jaccard-like distance estimation via weighted and
774 normalized MinHash for k-mers, is highly efficiently, and, importantly, can effectively deal with
775 incomplete genomes or genomes of (drastically) different length, a known limitation of traditional
776 Mash-based methods (22). Comparing genomes of different length is not uncommon, e.g.,
777 bacterial genome sizes can differ by more than ten-fold, as can be the case between MAGs of
778 different level of completeness or when searching a short sequence (e.g., a bacteriophage
779 genome) against a large genome collection (e.g., the whole viral genome database). Our own
780 analysis showed that ProbMinHash is robust down to 50% completeness level (Table S12), which
781 is also the most commonly used standard for selecting MAGs of sufficient/high quality for further
782 analysis and reporting (64). ProbMinHash is also robust for genomes with repeats or gene
783 duplications (e.g., fungal genomes) due to the k-mer weighting step by weighted MinHash, a
784 property not shared by simple MinHash.
785

786 In general, the genome relatedness estimated, or best database matching genomes
787 identified, by GSearch were highly consistent with blast-based ANI/AAI results or phylogenetic
788 placement of the genomes using GTDB-Tk, particularly for query genomes with close relatives in
789 the database related at the species or genus levels (Supplementary file 1, Figure 3e). For more
790 distantly related query genomes relative to database genomes, classification results of GSearch
791 showed some differences with GTDB-Tk. These differences were not always possible to assess
792 further for the most correct genome placement but could be due, at least partly, to the
793 incompleteness and/or contamination of query or/and database genomes, which renders the
794 resulting concatenated alignment of universal genes used by GTDB-Tk unreliable (64,65), as only
795 a few amino-acid positions per gene are used in the final alignment. In contrast, the AAI and
796 ProbMinHash approaches should be more robust to changes of a small number of genes because
797 the entire proteome is considered.

798 Graph-based NNS methods achieve good performance compared to tree based and
799 locality-sensitive hashing (LSH) methods or space partitioning methods (33). Building a HNSW
800 graph relies on proximity of the database elements; thus, if the distances among database
801 elements, in our case genomes, cannot be effectively estimated, the navigation of graph becomes
802 less efficient (e.g., gets trapped/lost in local minima). This is especially problematic for highly
803 sparse/distantly related and diverse datasets, like the viral genome database, in which two phage
804 genomes could often share very little genomic information (k-mers), and thus the edges to choose
805 from during graph navigation are not accurate estimations of the relatedness of the corresponding
806 genomes. This is confirmed by our own results when using nucleotide-level search to build the
807 viral genome graph. Hence, the amino acid level will be much more robust for viral genomes and
808 is the recommended level to use. Finally, recent advancements in proximity graph building could
809 further reduce database building time from O(N*log(N)) to O(N*c) (where c is a constant

810 independent of N, number of genomes) and/or achieve better search performance than existing
811 approximate proximity graph (APG) alone by combining locality sensitive hashing (LSH) with
812 approximate proximity graph (66) (or APG, e.g. HNSW is a APG) to have LSH-APG. This
813 approach essentially reduces the number of points/genomes to be compared during graph
814 building and will be explored in future versions of GSearch to provide additional speed-up and/or
815 robustness for graph database building. The HNSW graph, and graph-based K-NNS in general,
816 can also be further improved by adding shortcut edges and maintaining a dynamic list of
817 candidates compared to a fixed list of candidates used by default (67). Graph reordering, a cache
818 optimization that works by placing neighboring nodes in consecutive (or near-consecutive)
819 memory locations, can also be applied to improve the speed of HNSW (68). We also want to
820 mention that in our HNSW Rust implementation, a memory map was also implemented, which
821 will solve the memory limit problem when building large graph with billions of data points or
822 genome. Another direction to further improve GSearch could be the use of Graphics Processing
823 Unit (GPU) instead of CPU because GPUs are more efficient in handling matrix computations and
824 machine learning tasks (69). We will explore these options in future versions of GSearch.

825 Finally, GSearch provides a general framework/idea to create a new data structure:
826 combining probabilistic data structures (e.g., MinHash, HyperLogLog) with graph-based nearest
827 neighbor search algorithm that should be applicable not only to genomes but also text or
828 document searching purposes more broadly. For example, the MinHash algorithm can be applied
829 to document (e.g., website, text) similarity estimation, and be combined with HNSW. Further, we
830 can hash strings to approximate edit distance (a metric distance) to avoid expensive dynamic
831 programming (70) via order MinHash (71). We reimplemented order MinHash in our ProbMinHash
832 library, and it can be easily applied to do fast DNA string (not fragmented genomes) or general
833 string search.

To summarize, GSearch, based on MinHash-like algorithms and HNSW, solves a major current challenge in search and classification of microbial genomes due to its efficiency and scalability. Both the Densified MinHash and HNSW are approximating the theoretical optimal limits of similar algorithms in terms of speed and accuracy. GSearch will serve the entire microbial sciences for years to come since it can be equally applied to fungal, bacterial, and viral genomes, and thus offer a common framework to identify, classify and study all microbial genomes, at the million-to-billion database genome level.

Data and Code 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. Code can be found here: https://github.com/jean-pierreBoth/gsearch or here: https://gitlab.com/Jianshu_Zhao/gsearch. Scripts for reproducing the major results can be found here: https://github.com/jianshu93/gsearch_analysis.

Author Contribution

J.Z, 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. L.M.R provided constructive suggestions. J.Z did the analysis and benchmark. J.Z and K.K wrote the manuscript.

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 MinHash and bottom-k sketch implementation. We also want to thank Otmar Ertl for discussion on SetSketch Shannon entropy and implementation of SuperMinHash/ProbMinHash, Xiaofei Zhao for discussions on mergeability of One Permutation MinHash with Optimal densification, and Dingyu Wang for discussion on τ-GRA relative variance with respect to HyperLogLog. Finally, we would like to thank two anonymous reviewers for their suggestions, which help improve the manuscript substantially.

References

1. Locey, K.J. and Lennon, J.T. (2016) Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences*, **113**, 5970-5975.
2. Chevallereau, A., Pons, B.J., van Houte, S. and Westra, E.R. (2022) Interactions between bacterial and phage communities in natural environments. *Nature Reviews Microbiology*, **20**, 49-62.
3. Parks, D.H., Chuvochina, M., Rinke, C., Mussig, A.J., Chaumeil, P.-A. and Hugenholtz, P. (2021) GTDB: an ongoing census of bacterial and archaeal diversity through a phylogenetically consistent, rank normalized and complete genome-based taxonomy. *Nucleic Acids Research*.
4. Camargo, A.P., Nayfach, S., Chen, I.-M.A., Palaniappan, K., Ratner, A., Chu, K., Ritter, Stephan J., Reddy, T.B.K., Mukherjee, S., Schulz, F. *et al.* (2022) IMG/VR v4: an expanded database of uncultivated virus genomes within a framework of extensive functional, taxonomic, and ecological metadata. *Nucleic Acids Research*, **51**, D733-D743.
5. Almeida, A., Nayfach, S., Boland, M., Strozzi, F., Beracochea, M., Shi, Z.J., Pollard, K.S., Sakharova, E., Parks, D.H., Hugenholtz, P. *et al.* (2021) A unified catalog of 204,938 reference genomes from the human gut microbiome. *Nature Biotechnology*, **39**, 105-114.
6. Nayfach, S., Roux, S., Seshadri, R., Udwary, D., Varghese, N., Schulz, F., Wu, D., Paez-Espino, D., Chen, I.M., Huntemann, M. *et al.* (2021) A genomic catalog of Earth's microbiomes. *Nature Biotechnology*, **39**, 499-509.

7. Konstantinidis, K.T. and Tiedje, J.M. (2005) Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences*, **102**, 2567-2572.
8. Konstantinidis, K.T. and Tiedje, J.M. (2005) Towards a Genome-Based Taxonomy for Prokaryotes. *Journal of Bacteriology*, **187**, 6258-6264.
9. Goris, J., Konstantinidis, K.T., Klappenbach, J.A., Coenye, T., Vandamme, P. and Tiedje, J.M. (2007) DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology*, **57**, 81-91.
10. Lee, I., Ouk Kim, Y., Park, S.-C. and Chun, J. (2016) OrthoANI: An improved algorithm and software for calculating average nucleotide identity. *International Journal of Systematic and Evolutionary Microbiology*, **66**, 1100-1103.
11. Chaumeil, P.-A., Mussig, A.J., Hugenholtz, P. and Parks, D.H. (2019) GTDB-Tk: a toolkit to classify genomes with the Genome Taxonomy Database. *Bioinformatics*, **36**, 1925-1927.
12. Chaumeil, P.-A., Mussig, A.J., Hugenholtz, P. and Parks, D.H. (2022) GTDB-Tk v2: memory friendly classification with the Genome Taxonomy Database. *bioRxiv*, 2022.2007.2011.499641.
13. Rodriguez-R, L.M., Gunturu, S., Harvey, W.T., Rosselló-Mora, R., Tiedje, J.M., Cole, J.R. and Konstantinidis, K.T. (2018) 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.
14. Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S. and Phillippy, A.M. (2016) Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, **17**, 132.
15. Jain, C., Rodriguez-R, L.M., Phillippy, A.M., Konstantinidis, K.T. and Aluru, S. (2018) High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries. *Nature Communications*, **9**, 5114.
16. Baker, D.N. and Langmead, B. (2019) Dashing: fast and accurate genomic distances with HyperLogLog. *Genome Biology*, **20**, 265.
17. Baker, D.N. and Langmead, B. (2022) Dashing 2: genomic sketching with multiplicities and locality-sensitive hashing. *bioRxiv*.
18. Zhao, X. (2018) BinDash, software for fast genome distance estimation on a typical personal laptop. *Bioinformatics*, **35**, 671-673.
19. Broder, A.Z. (1997), *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, pp. 21-29.
20. Flajolet, P., Fusy, É., Gandouet, O. and Meunier, F. (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics and Theoretical Computer Science*, 137-156.
21. Ertl, O. (2021) SetSketch: filling the gap between MinHash and HyperLogLog. *Proc. VLDB Endow.*, **14**, 2244–2257.
22. Koslicki, D. and Zabeti, H. (2019) Improving MinHash via the containment index with applications to metagenomic analysis. *Applied Mathematics and Computation*, **354**, 206-215.
23. Ioffe, S. (2010) Improved Consistent Sampling, Weighted Minhash and L1 Sketching. *2010 IEEE International Conference on Data Mining*, 246-255.
24. Ertl, O. (2020) ProbMinHash – A Class of Locality-Sensitive Hash Algorithms for the (Probability) Jaccard Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 1-1.
25. Wang, Q., Garrity, G.M., Tiedje, J.M. and Cole, J.R. (2007) Naïve Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. *Applied and Environmental Microbiology*, **73**, 5261-5267.
26. Gionis, A., Indyk, P. and Motwani, R. (1999) Similarity search in high dimensions via hashing. *Vldb*, **99**, 518-529.
27. Ram, P. and Sinha, K. (2019) Revisiting kd-tree for nearest neighbor search. *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, 1378-1388.
28. Dasgupta, S. and Sinha, K. (2013) Randomized partition trees for exact nearest neighbor search. *Proceedings of the 26th Annual Conference on Learning Theory*, **30**, 317--337.

29. Dong, W., Moses, C. and Li, K. (2011) Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th international conference on World wide web*, 577-586.
30. Malkov, Y., Ponomarenko, A., Logvinov, A. and Krylov, V. (2014) Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, **45**, 61-68.
31. Malkov, Y.A. and Yashunin, D.A. (2020) Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**, 824-836.
32. Fu, C., Xiang, C., Wang, C. and Cai, D. (2017) Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*.
33. Aumüller, M., Bernhardsson, E. and Faithfull, A. (2020) ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, **87**, 101374.
34. Wang, M., Xu, X., Yue, Q. and Wang, Y. (2021) A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, **14**, 1964–1978.
35. Shrivastava, A. (2017) Optimal densification for fast and accurate minwise hashing. *International Conference on Machine Learning*, 3154-3163.
36. Mai, T., Rao, A., Kapilevich, M., Rossi, R., Abbasi-Yadkori, Y. and Sinha, R. (2020), *Uncertainty in Artificial Intelligence*. PMLR, pp. 831-840.
37. Ertl, O. (2017) Superminhash-A new minwise hashing algorithm for jaccard similarity estimation. *arXiv preprint arXiv:1706.05698*.
38. Déraspe, M., Boisvert, S., Laviolette, F., Roy, P.H. and Corbeil, J. (2020) Fast protein database as a service with kAAmer. *bioRxiv*, 2020.2004.2001.019984.
39. Van der Jeugt, F., Dawyndt, P. and Mesuere, B. (2022) FragGeneScanRs: faster gene prediction for short reads. *BMC Bioinformatics*, **23**, 198.
40. Finn, R.D., Clements, J. and Eddy, S.R. (2011) HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*, **39**, W29-W37.
41. Grigoriev, I.V., Nikitin, R., Haridas, S., Kuo, A., Ohm, R., Otillar, R., Riley, R., Salamov, A., Zhao, X. and Korzeniewski, F. (2014) MycoCosm portal: gearing up for 1000 fungal genomes. *Nucleic acids research*, **42**, D699-D704.
42. Ter-Hovhannisyan, V., Lomsadze, A., Chernoff, Y.O. and Borodovsky, M. (2008) Gene prediction in novel fungal genomes using an ab initio algorithm with unsupervised training. *Genome research*, **18**, 1979-1990.
43. Ye, L., Mei, R., Liu, W.-T., Ren, H. and Zhang, X.-X. (2020) Machine learning-aided analyses of thousands of draft genomes reveal specific features of activated sludge processes. *Microbiome*, **8**, 1-13.
44. Nishimura, Y. and Yoshizawa, S. (2022) The OceanDNA MAG catalog contains over 50,000 prokaryotic genomes originated from various marine environments. *Scientific Data*, **9**, 305.
45. Fu, C. and Cai, D. (2016) EFANNA: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. arXiv 2016. *arXiv preprint arXiv:1609.07228*.
46. Konstantinidis, K.T., Rosselló-Móra, R. and Amann, R. (2017) Uncultivated microbes in need of their own taxonomy. *The ISME Journal*, **11**, 2399-2406.
47. Ohayon, T. (2021) ExtendedHyperLogLog: Analysis of a new Cardinality Estimator. *arXiv preprint arXiv:2106.06525*.
48. Karppa, M. and Pagh, R. (2022) HyperLogLogLog: Cardinality Estimation With One Log More. *arXiv preprint arXiv:2205.11327*.
49. Ertl, O. (2023) UltraLogLog: A Practical and More Space-Efficient Alternative to HyperLogLog for Approximate Distinct Counting. *arXiv preprint arXiv:2308.16862*.
50. Błasiok, J. (2019) Optimal streaming and tracking distinct elements with high probability. *ACM Transactions on Algorithms (TALG)*, **16**, 1-28.
51. Kane, D.M., Nelson, J. and Woodruff, D.P. (2010), *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 41-52.

52. Rowe, W.P.M., Carrieri, A.P., Alcon-Giner, C., Caim, S., Shaw, A., Sim, K., Kroll, J.S., Hall, L.J., Pyzer-Knapp, E.O. and Winn, M.D. (2019) Streaming histogram sketching for rapid microbiome analytics. *Microbiome*, **7**, 40.
53. Cormode, G. and Muthukrishnan, S. (2005) An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, **55**, 58-75.
54. Ting, D. (2018) Count-Min: Optimal Estimation and Tight Error Bounds using Empirical Error Distributions. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2319–2328.
55. Ting, D. (2019) Approximate Distinct Counts for Billions of Datasets. *Proceedings of the 2019 International Conference on Management of Data*, 69–86.
56. Ertl, O. (2018) BagMinHash - Minwise Hashing Algorithm for Weighted Sets. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1368–1377.
57. Christiani, T. (2020) DartMinHash: Fast Sketching for Weighted Sets. *arXiv preprint arXiv:2005.11547*.
58. Hera, M.R., Pierce-Ward, N.T. and Koslicki, D. (2022) Debiasing FracMinHash and deriving confidence intervals for mutation rates across a wide range of evolutionary distances. *bioRxiv*, 2022.2001.2011.475870.
59. Liu, S. and Koslicki, D. (2022) CMash: fast, multi-resolution estimation of k-mer-based Jaccard and containment indices. *Bioinformatics*, **38**, i28-i35.
60. Shaw, J. and Yu, Y.W. (2023) Fast and robust metagenomic sequence comparison through sparse chaining with skani. *bioRxiv*, 2023.2001. 2018.524587.
61. Tan, S., Xu, Z., Zhao, W., Fei, H., Zhou, Z. and Li, P. (2021), *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, Virtual Event, Singapore, pp. 1552–1560.
62. Břinda, K., Sykulski, M. and Kucherov, G. (2015) Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, **31**, 3584-3592.
63. Joudaki, A., Rätsch, G. and Kahles, A. (2021) Fast Alignment-Free Similarity Estimation By Tensor Sketching. *bioRxiv*.
64. Parks, D.H., Imelfort, M., Skennerton, C.T., Hugenholtz, P. and Tyson, G.W. (2015) CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome research*, **25**, 1043-1055.
65. Bowers, R.M., Kyrpides, N.C., Stepanauskas, R., Harmon-Smith, M., Doud, D., Reddy, T.B.K., Schulz, F., Jarett, J., Rivers, A.R., Eloe-Fadrosh, E.A. *et al.* (2017) Minimum information about a single amplified genome (MISAG) and a metagenome-assembled genome (MIMAG) of bacteria and archaea. *Nature Biotechnology*, **35**, 725-731.
66. Zhao, X., Tian, Y., Huang, K., Zheng, B. and Zhou, X. (2023) Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *VLDB Endowment*.
67. Prokhorenkova, L. and Shekhovtsov, A. (2020) Graph-based Nearest Neighbor Search: From Practice to Theory. *Proceedings of the 37th International Conference on Machine Learning*, **119**, 7803--7813.
68. Coleman, B., Segarra, S., Shrivastava, A. and Smola, A. (2021) Graph Reordering for Cache-Efficient Near Neighbor Search. *arXiv preprint arXiv:2104.03221*.
69. Groh, F., Ruppert, L., Wieschollek, P. and Lensch, H. (2022) GGNN: Graph-based GPU Nearest Neighbor Search. *IEEE Transactions on Big Data*, 1-1.
70. Šošić, M. and Šikić, M. (2017) Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, **33**, 1394-1395.
71. Marçais, G., DeBlasio, D., Pandey, P. and Kingsford, C. (2019) Locality-sensitive hashing for the edit distance. *Bioinformatics*, **35**, i127-i135.

Figures

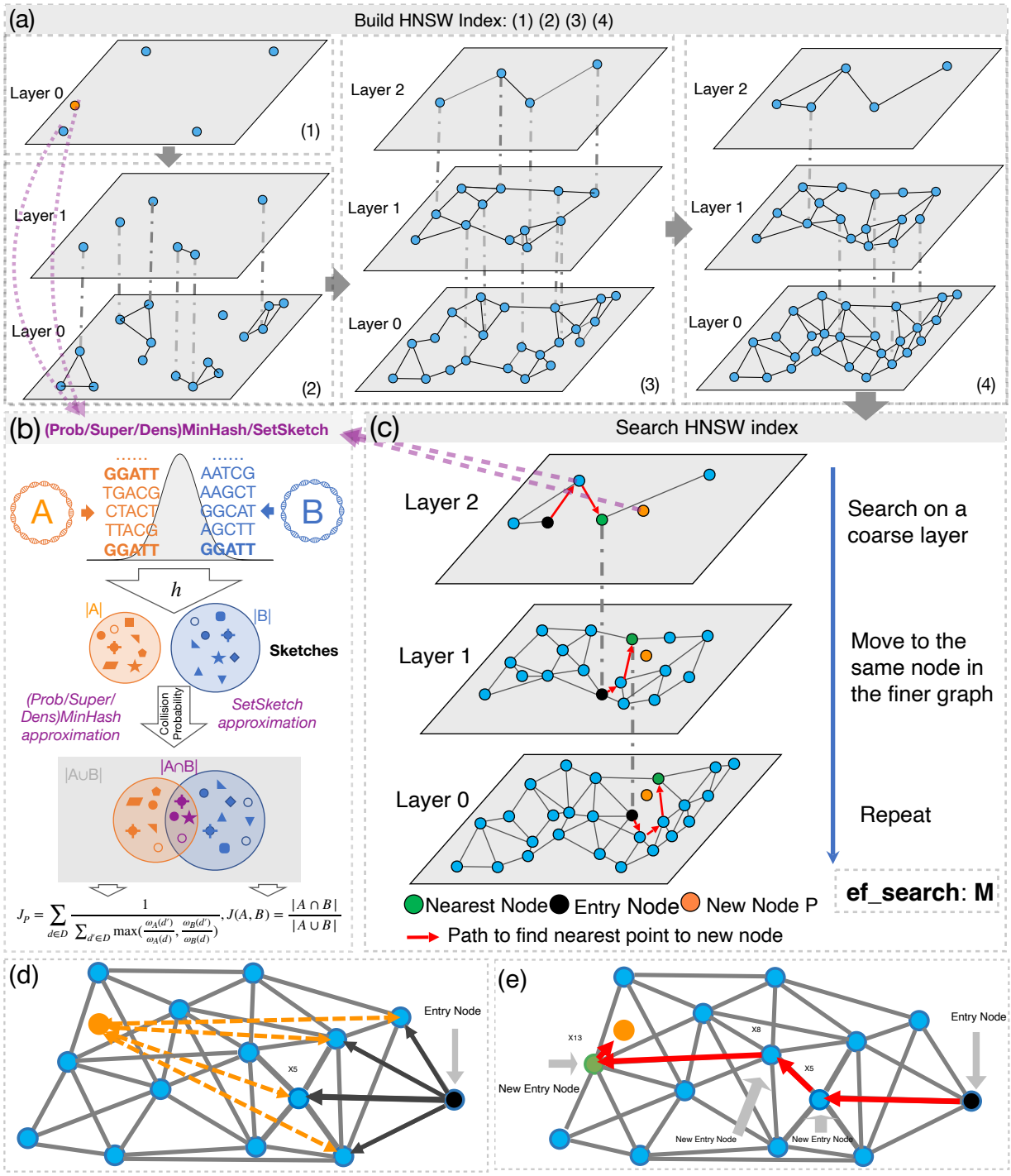

Figure 1. **Schematic overview of GSearch building graph (a), graph searching (c) and (Super/Prob/Dens)MinHash/SetSketch distance estimated (b)**. **(a)** Overview of the HNSW building process. GSearch (tohnsw module) starts from a randomly chosen genome to gradually build the graph at Layer 0 (1) and gradually collapse genomes(dash

1090 grey line indicate newly clustered/collapsed representatives compare to previous step)
1091 into Layer 1 (2) and layer 2 (3) as more genomes are being incorporated in Layer 0 until
1092 all genomes in database are inserted into layer 0 (4). When inserting new genomes in the
1093 graph (database), GSearch essentially searches in a partially built graph to find nearest
1094 genomes, based on ProbMinHash/SetSketch distance values among the genomes; see
1095 (c) for details. After the required number of nearest neighbors are found for each inserted
1096 genome, a reverse update step is performed to update neighbor list of all nodes in the
1097 graph. **(b)** Overview of (Super/Prob)MinHash/SetSketch algorithms to calculate J_p and J,
1098 where h is the hash function to hash k-mers from two genomes (orange and blue) and
1099 store the hashed values as sketches (shapes in the two big circles). ProbMinHash
1100 calculates J_p from shared sketches while SuperMinHash, Densified MinHash and
1101 SetSketch calculate J from shared sketches. **(c)** To search/identify a new genome P
1102 (orange) against the graph (request module), starting from an entry node (black, random
1103 or inherited from layer above it, depending on whether it is the top layer or not), GSearch
1104 finds the closest connected neighbor of the entry node (black) to the new node P to be
1105 searched (orange) by calculating the (Super/Prob/Dens)MinHash/SetSketch distance of
1106 the new node P (orange) with all neighbors (blue) of entry node (black) and assigns the
1107 closes one (X5 in this case) as the new entry point (that is X5 will be the new entry node)
1108 (panel **(d)**). GSearch is then traverses in a greedy manner (i.e., update the entry point
1109 using the newly found closest connected neighbor of X5) until the nearest neighbors in
1110 the layer are found (specifically, the path goes from entry node to X5, X8 and X13) (panel
1111 **(e)**), and then goes to next layer. This process is repeated until the required number of
1112 nearest neighbors (N) are all found for the given new querying data point P and
1113 subsequently, reported to the user.
1114
1115
1116
1117
1118

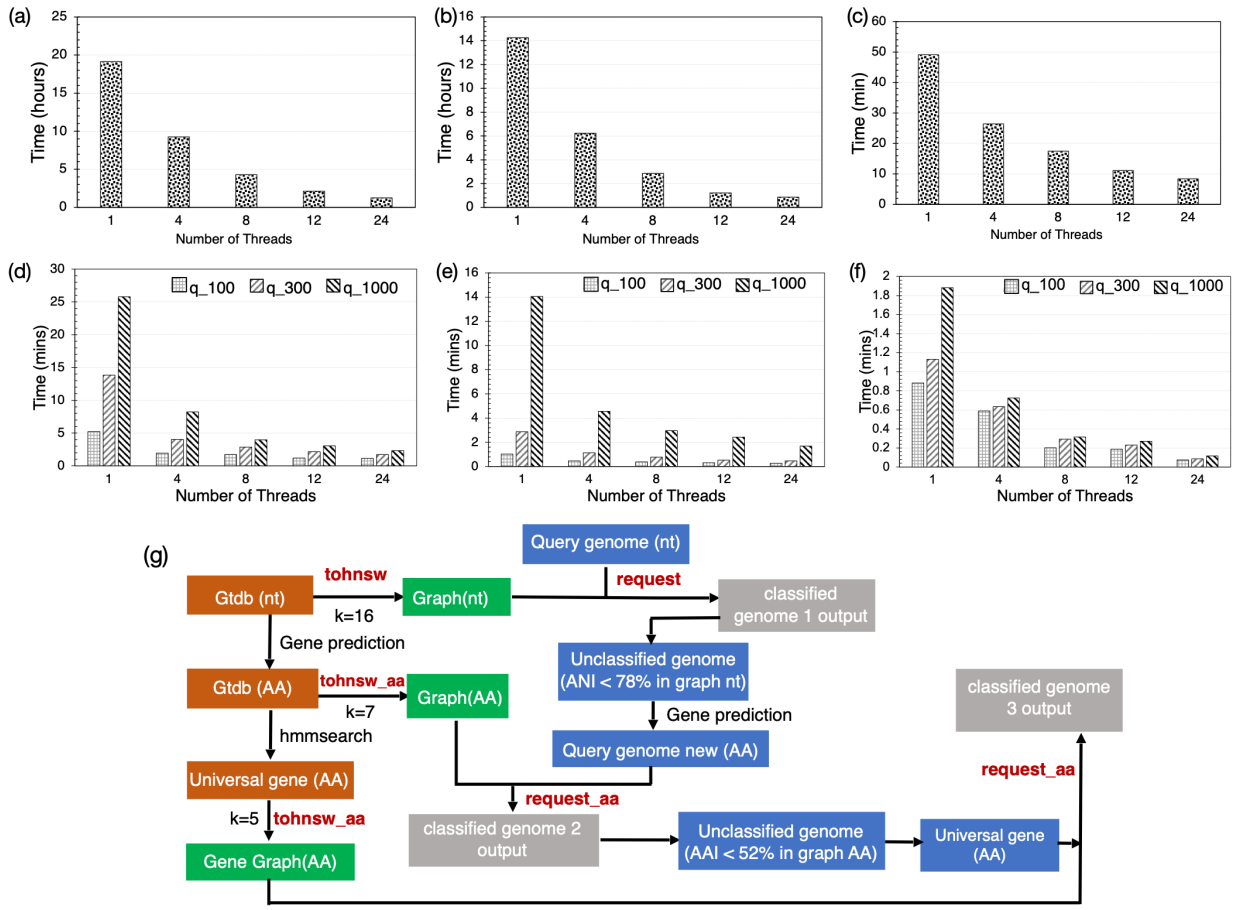

Figure 2. **Performance and scalability of database building (tohnsw) and searching (request) steps as the number of threads increases**. Upper panel shows total wall time (y axes) for building the GTDB v207 (65,703 genomes) at the nucleotide level **(a)**, whole-genome proteome (amino acid level) **(b)** and universal gene set proteome **(c)** reference graphs (or databases). Middle panel shows searching performance and scalability against number of threads used for different sets of query genomes at nucleotide **(d)**, amino acid level **(e)** and universal protein level **(f)**. Specifically, 100, 300 and 1000 query genomes (figure key) were used. All tests were run on a 24-thread Intel (R) Xeon (R) Gold 6226 processor, with 40GB memory available. **(g)** Overview of GSearch's the 3-step 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. Two key steps of GSearch: tohnsw and request are used to build graph database and request (or search) new genomes against the database, 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; that is, 78% ANI and 52% AAI, corresponding to Probminhash distance 0.9850 and 0.9375, respectively (see main text for details).

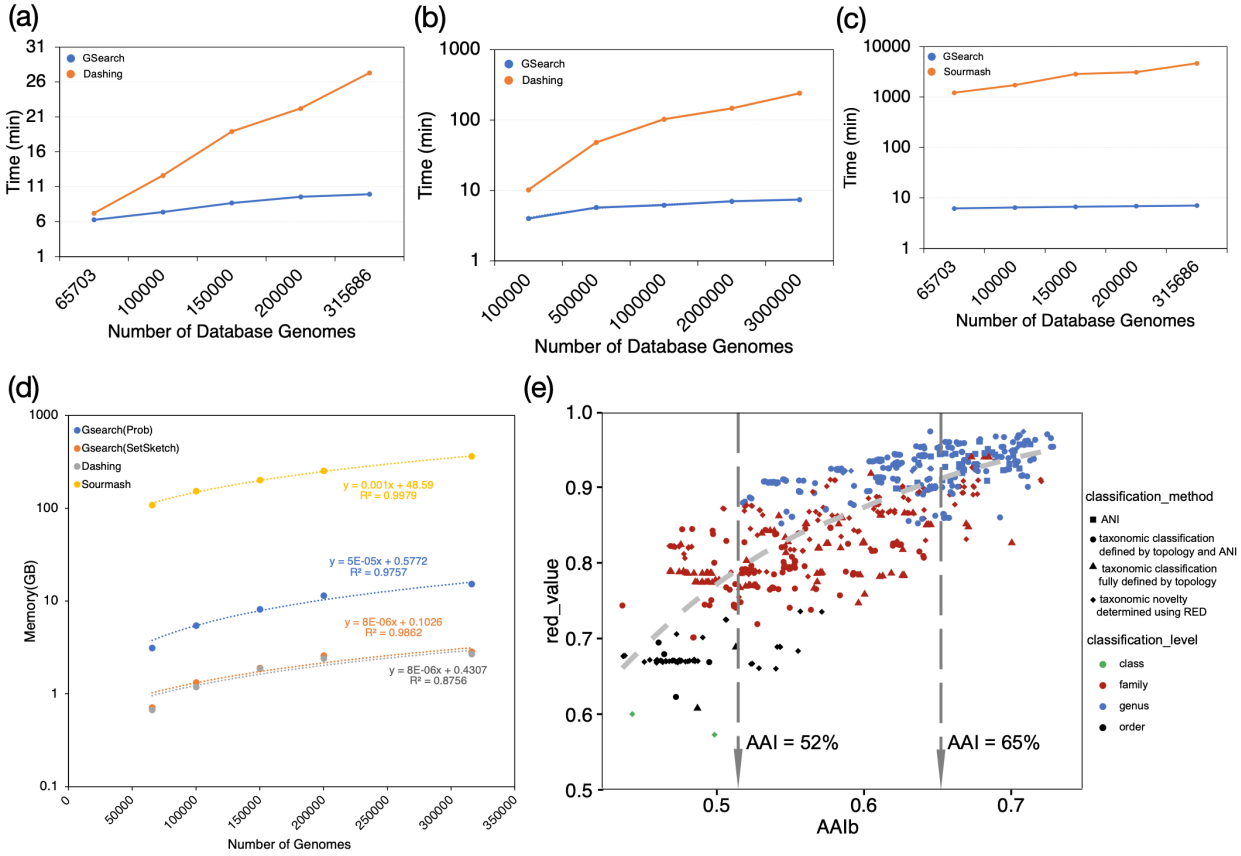

Figure 3. **Running time, memory consumption and classification accuracy of GSearch against Dashing, Sourmash and Blast-based ANI/AAI tools. (a)** Running time of GSearch versus Dashing for searching 8,466 query genomes against the RefSeq prokaryotic genome database as a function of the number of genomes used in the database (x-axis) at the nucleotide level. **(b)** Running time of GSearch (blue) versus Dashing (orange) for 10,000 query viral genomes against the IMG/VR v4 database at amino acid level. **(c)** Same as in (a) above but comparison is against Sourmash (orange). **(d)** Memory consumption of GSearch during versus Dashing and Sourmash search. Search is to load database into memory, thus maximum memory is directly related to

database size. Since Sourmash search is not parallelized, GNU parallel was used for process-level parallelism. **(e)** Comparison of GSearch classification results with GTDB-Tk and Blastp-based AAI tools for moderate-to-distantly related query genomes based on the bacterial proteome database (e.g., ANI between the query genome and its best match in the database was lower than 78% for these genomes). Each point represents a comparison between two genomes, query and the best match found by GSearch, showing RED values generated by GTDB-tk (y-axis) versus Blastp-based AAI between these two genomes. The taxonomic rank that the query and the best database match share is shown (see figure key). Two vertical lines indicate Blastp-based AAI threshold for family and genus level classification threshold. Note that the best match was always the same genome between GSearch and all vs. all Blastp AAI, and the overall consistency between GSearch/AAI and GTDB-tk in identifying the same best database genomes for the same query genomes. Note that y-axis values are in log scale in panels b, c, and d.

Table 1. Bioinformatics algorithms for sequence/genome searching and their comparisons. S represents a document/genome. N is the total number of documents/genomes. $U_{s \in S}$ |s| represents the total number of terms in N documents/genomes and $\sum_{s \in S}$ |s| is total unique terms/k-mer. MPH is minimal Perfect Hashing. For the Inverted Index size, the extra log (N) comes from the bit precision document IDs. For SBTs, log(N) is the height of the tree, and for Bloom Filters at each level is $O(\sum_{s \in S}$ |s|) in total. MinHash time and space complexity was based on k hash functions for N sets, each set has d' non-zero element. Note that the most recent B-bit MinHash with optimal densification can further improve to O(N*(d'+k)). O(vdlog(N)) for GSearch is O(log(N)) in practice since v and d is small and single pair genome comparison is a constant (Supplementary Note 3). τ depends on the user-specified parameter and is generally less than 0.01 in practice.

	Query Time	Space	Recall Benchmark (Biologically)	Comments
Inverted Index[1]	$O(1)$	$Best: log(N) \bigcup_{s \in S} \lvert s \rvert$	Only for document search/retrieval, could be applied to genomic/sequence search	**Long construction time; impractical for bigger datasets; best case needs MPH and a known k-mer (term) distribution**
BIGSCI/COBS[2,3]	$O(N)$	$\sum_{s \in S} \lvert s \rvert$	A hybrid between an inverted index and Bloom filters (COBS), high false positive rate, no benchmark with mutation rate by ANI/Mash	Query time is linear in N, small index size
Sequence Bloom Tree[4]	$Best: O(\log(N)),$ $Worse: O(N)$	$log(N) \sum_{s \in S} \lvert s \rvert$	Given the k-mers from a query sequence, the task is to determine which of the N documents contain all the k-mers present in the query; no benchmark with mutation rate by ANI/Mash	**Sequential query process is bottleneck; designed for sequential implementation**
RAMBO[5]	$O(\sqrt{N} * log(N))$	$\Gamma log(N) \sum_{s \in S} \lvert s \rvert$	Similar to SBT, finding which of the N documents/genomes contain all the k-mers present in the query, no benchmark with ANI/Mash	Only for $\Gamma < 1$, query time is sub-linear
MinHash[6]	$O(N * d' * k)$	$O(N * k)$	Average Nucleotide Identity (ANI) or mutation rate via Mash distance	Query time is linear in N
GSearch (ProbMinHash+HNSW)	$O(vdlog(N))$	$O((1+\tau) * N * d + k * N)$	Average Nucleotide Identity (ANI) via Mash-like mutation rate/index	Long database construction time $O(N * log(N))$, but users are free from construction.
FLINNG[7]	$O(log^2\left(\frac{1}{\delta}\right) log^3(N) N^{\frac{1}{2}+\gamma})$	$N^{\frac{3}{2}} log^2(N)$	No Benchmark with Average Nucleotide Identity (ANI) via Mash-like mutation rate/index, only 15% of RefSeq genome meet the γ-stable criteria	The γ-stable query condition is a relatively strong requirement for the query. Limitation: works for queries for which the neighbors are all above a (relatively high) similarity threshold to the query

[1]Croft, Metzler and Strohman [1], [2]Bradley, Den Bakker, Rocha, McVean and Iqbal [2], [3]Bingmann, Bradley, Gauger and Iqbal [3], [4]Solomon and Kingsford [4], [5]Gupta, Yan, Coleman, Kille, Elworth, Medini, Treangen and Shrivastava [5], [6]Broder [6],[7]Engels, Coleman and Shrivastava [7]

Table 2. GSearch's performance on major CPU platforms for searching 8,466 query genomes against the RefSeq reference database (~318K genomes). ProbMinHash was used as the genomic distance algorithm for this benchmark, with 50 neighbors requested.

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)[d]**
Intel (R) Xeon (R) Gold 6226[a]	24	2.70	**9.329**	3.348	**6.234**	3.524	**0.445**
Intel (R) Core i9-8950HK[b]	16	2.30	**13.654**	6.764	**7.91**	6.214	**0.613**
AMD EPYC 7513a[a]	32(24 used)	2.60	**10.245**	4.452	**6.021**	3.345	**0.402**
Apple M1 Pro[b]	10	3.22	**9.669**	4.31	**6.166**	4.603	**0.568**

[a] RHEL v7.9, Linux v3.10.0-1160, all threads used. 512G RAM available.
[b]MacOS v12.3, Darwin 21.4.0 on MacBook Pro laptops, all threads used. Database was split into two pieces to have smaller memory requirement on those 16GB RAM laptops since database size was ~15GB for entire NCBI/RefSeq.
[c]Parallel package was used to run multiprocess at the same time. FGSrs stands for FragGeneScanRs (comparisons with Prodigal can be found in Table S11). Note that in practice only those genomes failing in the request step at the nucleotide (nt) level (best match found was less than 78% ANI) will be used in this amino-acid step.
[d]Only 1000 genomes were used for testing hmmsearch and USCG request in GSearch because this step was for assessing a few genomes that were novel at the order level or above. Parallel Packages was used to run multiple processes of hmmsearch, one thread per process for hmmsearch.

Table 3. Recall of GSearch for two query genome datasets containing various levels of novel genomes relative to the genomes in the reference database. The query genome datasets are the same as those used in Table 2. Blastn-ANI, blastp-AAI and universal gene blastp-ANI were used as the reference results (or standards) to compare against. Recall is the average across all query genomes.

Dataset	Dataset size (# of genomes)	level	Recall (top 5)	Recall (top 10)
Tara (with best in database)[1]	**6,992**	**nucleotide**	**98.3%**	**96.2%**
Tara (without best hit in database)	1474	nucleotide	62.4%	43.1%
Tara (with best in database)[2]	**1147**	**proteome**	**97.2%**	**96.9%**
Tara (without best hit genome in database)	327	proteome	65.0%	50.8%
Tara (with best in database)[3]	**327**	**universal**	**96.4%**	**95.5%**
Ye (with best in database)[1]	**906**	**nucleotide**	**97.7%**	**95.1%**
Ye (without best hit in database)	91	nucleotide	54.8%	48.5%
Ye (with best in database) [2]	**66**	**proteome**	**96.3%**	**95.2%**
Ye (without best hit in database)	25	proteome	76.0%	55.5%
Ye (with best in database)[3]	**25**	**universal**	**95.7%**	**94.6%**

[1]Definded as query genomes that had best match in the database better than 78% ANI; [2]defined as query genomes that had best match in the database better than 52% AAI; [3]defined as query genomes that had best match in the database better than 55% AAI based on universal genes.

Table References

1. Croft, W.B., Metzler, D. & Strohman, T. Search engines: Information retrieval in practice, Vol. 520. (Addison-Wesley Reading, 2010).
2. Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G. & Iqbal, Z. Ultrafast search of all deposited bacterial and viral genomic data. *Nature biotechnology* **37**, 152-159 (2019).
3. Bingmann, T., Bradley, P., Gauger, F. & Iqbal, Z. in String Processing and Information Retrieval: 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26 285-303 (Springer, 2019).
4. Solomon, B. & Kingsford, C. Fast search of thousands of short-read sequencing experiments. *Nature biotechnology* **34**, 300-302 (2016).
5. Gupta, G. et al. Fast processing and querying of 170tb of genomics data via a repeated and merged bloom filter (rambo). *Proceedings of the 2021 International Conference on Management of Data*, 2226-2234 (2021).
6. Broder, A.Z. in Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171) 21-29 (IEEE, 1997).
7. Engels, J., Coleman, B. & Shrivastava, A. Practical near neighbor search via group testing. *Advances in Neural Information Processing Systems* **34**, 9950-9962 (2021).