# Supplementary Information for

# Compressive genomics allows computational analysis to keep pace with genomic data

Po-Ru Loh[1*], Michael Baym[1,2*†], Bonnie Berger[1†]

[1]Department of Mathematics and Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA

[2]Department of Systems Biology, Harvard Medical School, Boston, MA

[*]These authors contributed equally to this work.

[†]To whom correspondence should be addressed;

E-mails: bab@mit.edu, baym@mit.edu.



Nature Biotechnology: doi:10.1038/nbt.2241

# Supplementary Methods

## CaBLAST

This section describes our compressive implementation of BLAST [1], herein called CaBLAST (Compression-accelerated BLAST). The algorithm is schematically depicted in Figure S1.

**Preprocessing phase.** Conceptually, the preprocessing phase of CaBLAST (Fig. S1A) views the original database as an incoming stream of DNA, reading it base by base (with distinct sequences from the database processed in any order) while building four main data structures: the **unique database** and list of **link pointers** with corresponding **edit scripts**—together forming the compressed output of the preprocessing phase—along with a table of **seed 10-mer locations** used internally during preprocessing. As sequence data is processed, it is either:

(a) aligned to existing sequence data in the unique database and represented using a link pointer and edit script; or

(b) added to the unique database (with information about its origin), in which case its 10-mers are added to the table of seeds.

Thus, the unique database roughly contains non-redundant data from the original database, while redundant data is represented using link pointers and edit scripts. For convenience, we refer to the link pointers and edit scripts together as the **links table**.

The main algorithmic challenge in preprocessing is to efficiently identify redundancy within the original database: as we process incoming data, we need to quickly determine whether it aligns well to any part of the unique database being built. We do so by using the seed 10-mer table to rapidly identify seed matches which we then attempt to extend to longer alignments, similar to BLAT [2].



Nature Biotechnology: doi:10.1038/nbt.2241

While the overall approach is fairly straightforward, a few important subtleties arise. First, because both matches stored in the links table and sequence data saved to the unique database are likely to consist of sequence fragments rather than complete entries from the original database, it is necessary for the purpose of maintaining search accuracy to lengthen saved fragments so that they overlap (see below for further discussion). Our implementation stores an overlap of 100 bases at transitions between adjacent fragments. The overlap results in an overhead cost per fragment; consequently, we only apply link replacement to alignments of length at least 300 during preprocessing.

Additionally, repetitive genomic elements create anomalously high numbers of matches to a small number of (low-complexity) 10-mer seeds that would substantially reduce performance. Our current implementation circumvents this problem by capping the number of locations stored for a given seed at 500; in practice, aligning regions typically contain at least some seeds that are not of low-complexity and thus the loss of compression is negligible.

We now describe the algorithm more explicitly. The preprocessing phase maintains two pointers that keep track of progress through the incoming data stream:

- a **current base pointer** to the current location being processed; and
- a **last-fragment pointer** to the end of the sequence fragment most recently aligned or copied to the unique database,

and proceeds as follows:

1. Initialize unique database by copying first 10,000 bases of original database; set current base pointer and last-fragment pointer to end of initial region. Initialize table of seed locations with all 10-mers occurring in the copied region. Initialize links table as empty.
2. (Begin loop.) Advance current base pointer one base. Look up 10-mer seed ending at current pointer position, along with its reverse complement, in table of seed locations.



Nature Biotechnology: doi:10.1038/nbt.2241

3. For each seed match from table, attempt to extend match (as detailed in next subsection). If extended match of length at least 300 is found:

   (a) Augment links table with link to match and edit script allowing reconstruction (details below).

   (b) Augment unique database with copy of original sequence data preceding alignment that is unaccounted for in unique database: specifically, bases from last-fragment pointer (minus 100 bases) to start of current alignment (plus 100 bases).

   (c) Augment table of seed locations with locations (in the unique database) of all 10-mers occuring in the copied region.

   (d) Move current base pointer and last-fragment pointer to end of alignment and go back to step 2.

4. If 10,000 bases have been processed with no alignments found, copy 10,000-base chunk to unique database, augment table of seed locations accordingly, move both pointers to end of chunk, and go back to step 2. Otherwise, go back directly to step 2.

**Details of alignment extension during preprocessing.** CaBLAST must rapidly check if a 10-mer seed match extends to an alignment of length 300 or more, allowing only a limited amount of mutation in any chunk of consecutive bases. With this intuition, the algorithm attempts extension in each direction by hopping between matching 4-mers. The extension procedure alternates between the following two steps:

1. Attempt ungapped extension. Scan forward along both sequences looking for the next 4-mer match assuming no gaps. When no 4-mer match is found within 10 bases or when there is less than 50% sequence identity between one 4-mer match and the next, move on to step 2.



Nature Biotechnology: doi:10.1038/nbt.2241

2. Attempt gapped extension by local dynamic programming on the next 25 bases of each sequence. Tentatively extend the alignment by taking the path through the dynamic programming table containing the most matching bases. Along this path, identify matching 4-mers and extend the alignment to each successive 4-mer match that occurs within 10 bases of the previous and with at least 50% identity in between (as in step 1). If no acceptable 4-mer match is found, quit. Otherwise, restart step 1 after the last successful extension to a 4-mer match.

The final alignment is accepted as a match if it has length at least 300 and sequence identity exceeding a user-specified threshold (70-90%) in every 100-base window of the alignment.

**Edit script compression.** Upon identifying a suitable alignment between a sequence fragment and a reference section of the unique database, the sequence data contained in the fragment is compressed by encoding the fragment as a pointer to the original sequence along with a string containing a series of edits that can be applied to the reference segment to produce the aligned fragment. As high sequence identity between the aligned fragments can be assumed from the alignment step, edits can be treated efficiently as islands of difference surrounded by regions of exact matching.

For each differing island, we store a character indicating whether the modification is an insertion (`i`) or substitution/deletion (`s`). We then indicate the number of bases (encoded in octal) since the previous edit, counting along the reference segment, and finally the sequence to insert or substitute. Note here that deletions are treated as substitutions of dashes for the reference bases. Thus, for the following pair of aligned sequences (reference on top, substitutions/deletions in red, insertions in blue for clarity):

```
GTTCACTTATGTATTC--ATATGATTTTGGCAA
GTTCACG--TGTATATTTATATAATTTTGGCAA
```



Nature Biotechnology: doi:10.1038/nbt.2241

we generate the following edit script (every other command in green for clarity):

**s6G--s10ATi2TTs6A**

We precede each edit script by a 0 or 1, indicating whether the pair of aligned sequences is same-strand or reverse complemented, and also delineating the end of an edit script: on disk, all the edits are concatenated into a single file. The full character set used by the edit scripts thus consists of 16 characters (0–7, A, C, G, T, N, –, i, s) allowing a 4-bit-per-character encoding.

We store pointers in a separate file. Each link pointer contains 20 bytes of information: the start index of the aligned fragment within its originating sequence (4 bytes), the start index of the reference fragment within the unique database (4 bytes), the start index of the edit script within the scripts file (4 bytes), and the lengths of the two aligned sequences (2 bytes each). (Some of this information can be deduced from the edit scripts, but since we build a data structure containing this information for use during search, we included it all in the links table files so as to truly be searching the compressed data without decompressing.)

**Choice of parameters.** Many constants appear in the above description of our implementation. We did not attempt to optimize all of the various parameters as we intended CaBLAST simply to be a working prototype rather than a recipe for best performance, but the rough rationale and engineering trade-offs involved in each are as follows.

We chose 10,000 bases as the maximum chunk size in order to be large enough to make the impact of link overhead minimal, while small enough to be able to decode hit regions within chunks without an excessive amount of decoding of extraneous regions. The 10,000-base limit also conveniently fit within a 2-byte integer.

We set the overlap of consecutive fragments to 100 bases so that a hit spanning consecutive fragments would likely be picked up as a hit to at least one of them; 100 bases was sufficient



Nature Biotechnology: doi:10.1038/nbt.2241

for this at typical search sensitivity thresholds for BLAST and BLAT. We then lower-bounded usable fragment lengths by 300 bases to reduce the impact of the overhead cost of overlaps.

As for the alignment extension parameters, all were chosen to strike a reasonable balance between preprocessing speed and gap extension sensitivity. The basic idea is to run dynamic programming on a limited scale in order to detect gaps but give up when further alignment extension seems improbable, so as not to slow down the preprocessing. The parameters reported are simply the ones we tried initially which performed adequately for this purpose.

Finally, the percentage identity requirement on 100-base windows of alignments represents a more significant trade-off between search sensitivity and compression; we discuss this in detail later in this supplement.

**Runtime of preprocessing.** Algorithmically, the runtime of the CaBLAST preprocessing phase is in between linear and quadratic (with an extremely small constant factor) in the database size depending on the amount of structure in the database. This phase has to be run only once to create the searchable unique database plus links table; there is no need to run it for subsequent searches. The quadratic scaling arises on unstructured sequence data because at each position, the expected number of seed matches found roughly equals the current size of the unique database divided by the number of different seeds—in our 10-mer implementation, close to one million. When the original database contains significant regions of similarity, however, the runtime is closer to linear: in such situations, alignment extension (which takes time only linear in the alignment length) accounts for most of the bases processed; seed matches need only be looked up at the small fraction of positions between alignments. The upshot is that the runtime performance of preprocessing is highly data-dependent.

In practice, we did not take pains to optimize our code for speed and were satisfied being able to complete runs on a standard workstation in a convenient amount of time for testing



Nature Biotechnology: doi:10.1038/nbt.2241

(a few minutes for 750MB of closely-related microbial sequences and under an hour for 1GB worth of more divergent fly genomes). We are confident that this runtime could be reduced substantially if desired, either by code optimization or by increasing seed lengths to reduce the rate of seed match lookups (at the cost of finding slightly fewer alignments).

**Search phase.** The search phase of CaBLAST (Fig. S1B) applies the procedure below given a query along with coarse and final E-value thresholds:

1. Coarse BLAST. BLAST the query against the unique database to find hits passing the coarse E-value threshold.

2. Link-tracing. For each hit in the unique database, check the links table for other sequence segments outside the unique database that align to the hit region; recover original sequence segments corresponding to the hit by local decompression using stored edit scripts. Extend these segments by 50 bases on each side (in case the linked regions admit longer alignments to the query than the initial hit).

3. Fine BLAST. Re-BLAST against the expanded hits using the final E-value threshold.

**Trade-off between compression and accuracy.** The percent identity threshold per 100-base window, used during preprocessing to decide whether an alignment qualifies as a link, is of interest because it represents a trade-off between compression and accuracy: relaxing the threshold allows greater database reduction and hence search speedup, but also increases the risk of overlooking alignments during search, due to greater differences between original sequence fragments and their representatives in the unique database. To quantify this trade-off, we ran the CaBLAST preprocessing phase with 100-base identity thresholds ranging from 70-90% on microbial datasets (as discussed in the main text) as well as *Drosophila* subtrees. As expected,



Nature Biotechnology: doi:10.1038/nbt.2241

accuracy improves while search speedup decreases as the similarity threshold is made stricter (Fig. S6). The best parameter choice thus depends on the target application; in our computations we used 80% for the fly genomes and 85% for the microbial datasets.

## CaBLAT

We applied our compressive framework to BLAT in a manner directly analogous to our use of BLAST in the coarse and fine search phases. Specifically, we perform the same preprocessing phase as before to create a unique database and links table.

We then call BLAT directly on the unique database for the coarse search, parse the output (in tab-delimited psl format), perform link-tracing, and regenerate likely hit regions of the original library using the edit scripts from the links table, and finally call BLAT's internal `ffFind()` ("fuzzyfind") and `scoreAli()` routines on these regions to match BLAT's alignment and scoring. Note that BLAT by default uses a simple score threshold based on matches and mismatches between query and target as opposed to BLAST's E-value; thus, we used BLAT's `minIdentity` parameter for coarse and fine search thresholding. In our tests we used the default value of 90 for the final threshold and 80 for the coarse threshold.

A minor technical difference is that BLAT appears to perform slightly better when sequence chunks in the unique database are stored as separate entries, whereas BLAST performs better when all sequences in the unique database are concatenated into one large sequence. We tuned our implementations accordingly. Additionally, BLAT's fuzzyfind algorithm which we used for fine alignment was noticeably more efficient when given hit regions cropped closer to the aligning fragments, so we padded hit regions by only 10 bases during CaBLAT link-tracing.



Nature Biotechnology: doi:10.1038/nbt.2241

## Simulated BLAST/BLAT queries

We generated simulated BLAST and BLAT queries by sampling genomic fragments from the genomic library being tested (e.g., for the *melanogaster* subgroup, we took from the *melanogaster* subgroup; for bacteria genera we sampled from the appropriate genus) and then applying random perturbations. More precisely, we randomly selected sequence fragments of random lengths up to 200, after which we replaced a random percentage (0-100%) of the bases in each fragment with random bases. Finally, we rejected a small proportion of these fragments (2% or less), typically from low-complexity regions of the genomes, that aligned to anomalously large numbers of sequence regions and would otherwise have skewed our results.

## Simulated genomes

To simulate clades of recently-diverged species, we used INDELible v1.03 [3], a tool for simulating genome evolution. We used default evolutionary parameters both for the phylogenetic birth-death model and for base-level mutation (JC base substitution probabilities, a geometric indel length distribution, and insertion and deletion rates of 0.08 and 0.12, respectively, relative to the substitution rate).

**Fairly testing CaBLAT vs. BLAT.** Before performing search, BLAT by default processes the search database indexing $k$-mers; the time taken by this step can account for a significant amount of BLAT's runtime. For a given query set, we therefore ran BLAT (resp. CaBLAT) on the query set (10,000 simulated queries) followed by a second run on a set containing all queries twice. This does not change the size of the compiled data structure. As initialization time is independent of the number of queries, the difference in times between the two runs gives the time taken for search alone. (To avoid timing discrepancies due to caching of previous files in memory after access, we also preceded this procedure by running the query once first, ignoring



Nature Biotechnology: doi:10.1038/nbt.2241

the results.) We report the average of five runs.

## Implementation and testing

We implemented CaBLAST and CaBLAT in C++ using the NCBI C++ Toolkit (Version 7.0.0, May 2011) and version 34 of the BLAT source tree, respectively, both for reference runs and in direct calls during the CaBLAST and CaBLAT coarse and fine search phases. Source code is available online at `cast.csail.mit.edu`.

We performed tests on the microbial datasets on a 3.33 GHz Intel Xeon X5680 CPU with 12 MB L3 cache and 48 GB RAM. We ran tests on the fly and simulated datasets on a 3.0 Ghz Intel Xeon CPU with 2 MB L2 cache and 24 GB RAM.



Nature Biotechnology: doi:10.1038/nbt.2241

# Supplementary Figures



Nature Biotechnology: doi:10.1038/nbt.2241

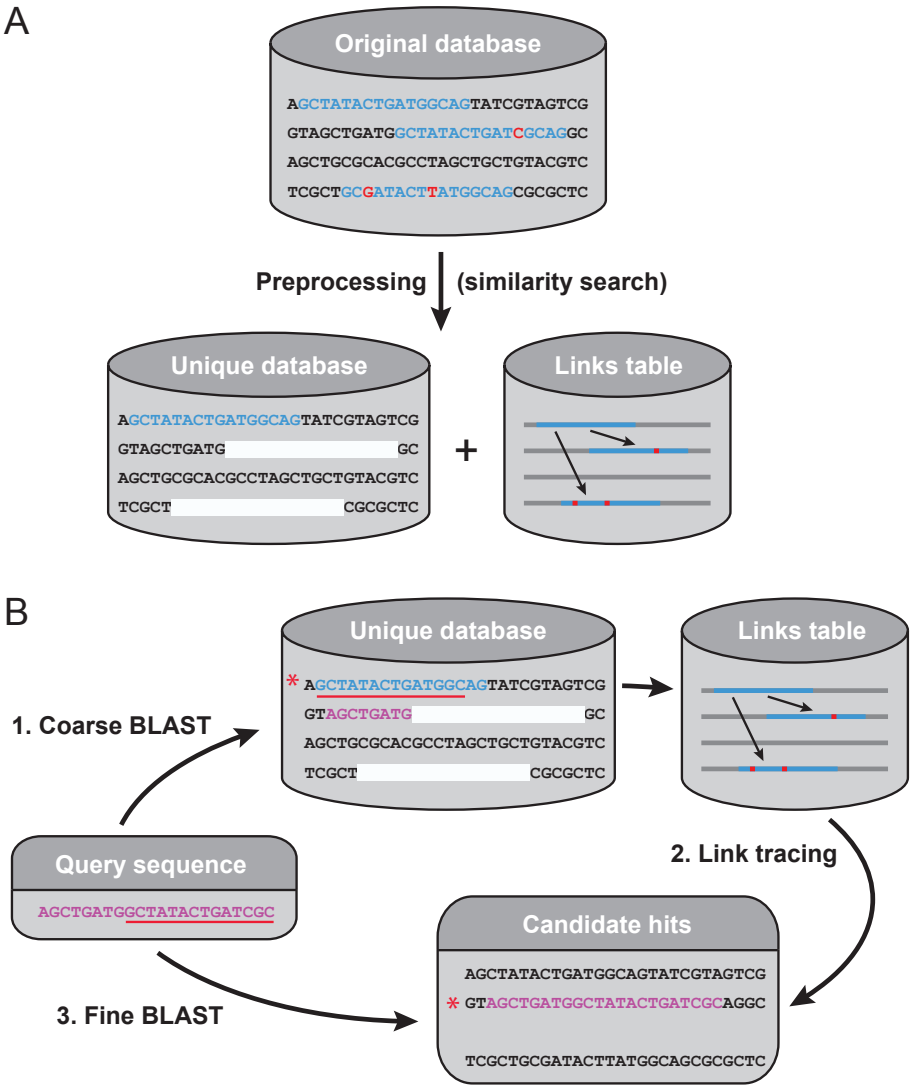


Figure S1: The CaBLAST algorithm. (*A*) Preprocessing: Scan through the genomic database, eliminating redundancy by identifying regions of high similarity and encoding them as links to previously seen sequences. The blue sequence fragments in the figure match perfectly, aside from a few discrepancies (red). Thus, only the first occurrence appears in the unique database; information about the other two fragments is stored in the links table. (*B*) Search: (1) BLAST against unique database with relaxed E-value threshold; (2) Recover additional candidate hits via links table; (3) BLAST against candidate hits to select final results. Here, the query (purple) matches the second original sequence, but most of the second sequence does not appear in the unique database because of a similar region in the first sequence (blue). The coarse BLAST query thus hits only the first sequence; the second sequence is recovered as a candidate after checking the links table.



Nature Biotechnology: doi:10.1038/nbt.2241

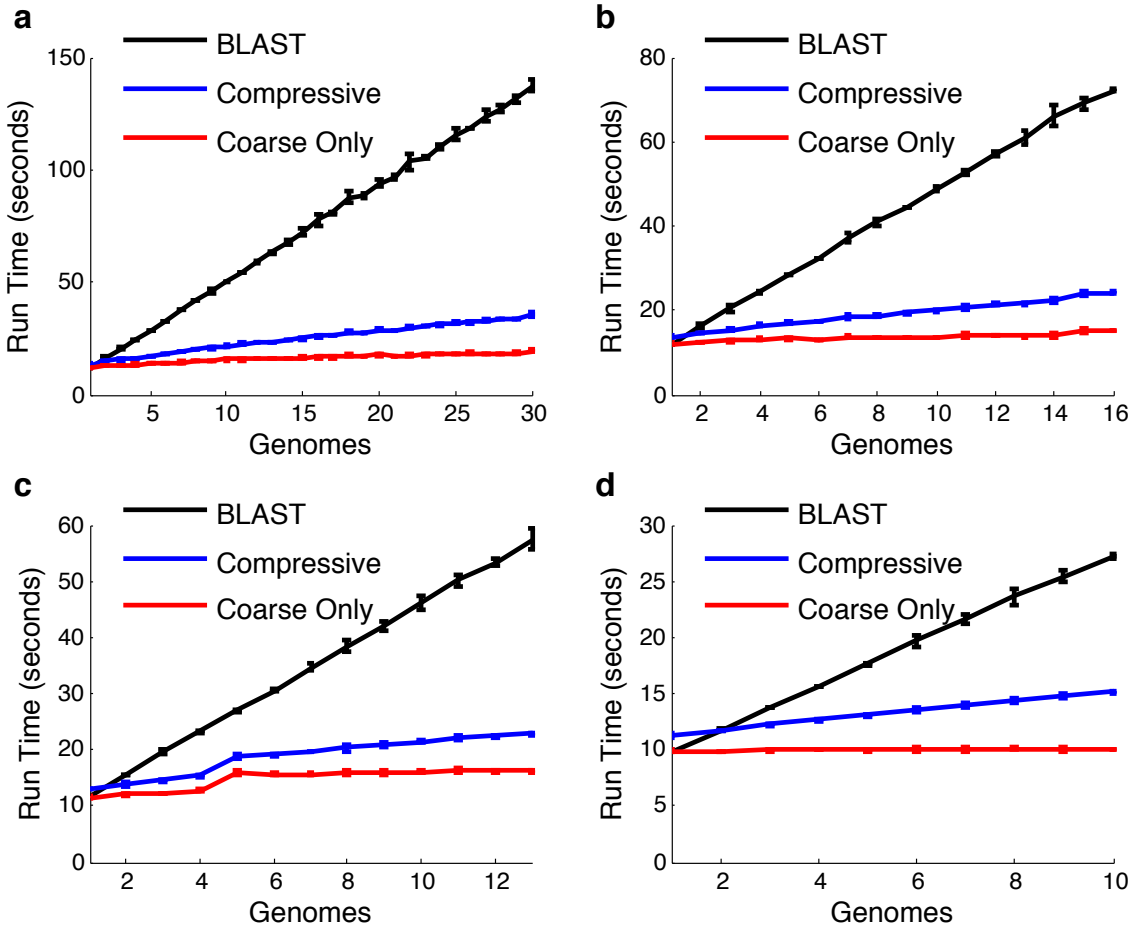


Figure S2: Performance of Compressive BLAST on databases of four bacterial genera using a single search set derived from the combined library of bacterial and yeast sequences. Parameters are the same (default) as in the primary manuscript. (a) Escherichia; (b) Salmonella; (c) Yersinia; (d) Brucella.



Nature Biotechnology: doi:10.1038/nbt.2241

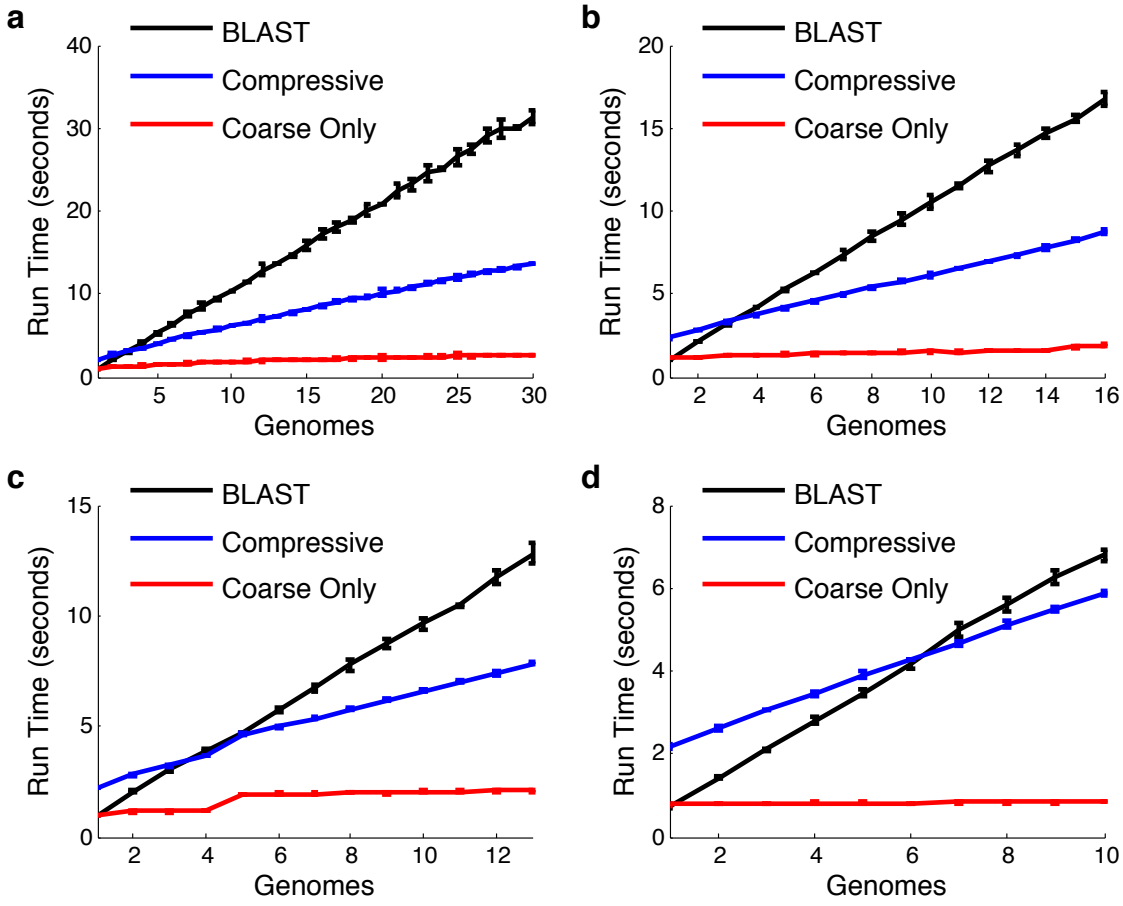


Figure S3: Performance of Compressive BLAST on databases of four bacterial genera using distinct search sets derived from each bacterial genus individually. Parameters are the same (default) as in the primary manuscript. Results, while still showing improvement over BLAST, are slower than the test set of Fig. S2, representing the increased time spent in fine search owing to increased hit rates. (a) Escherichia; (b) Salmonella; (c) Yersinia; (d) Brucella.



Nature Biotechnology: doi:10.1038/nbt.2241

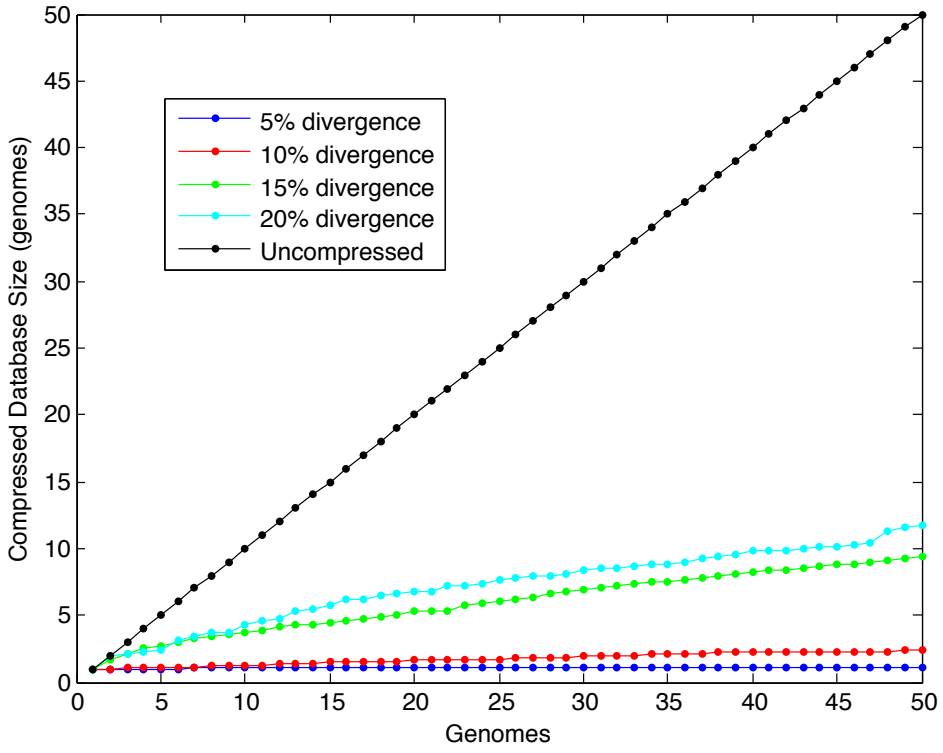


Figure S4: Performance of the Compressive BLAST preprocessing phase on simulated genera. Databases consist of sets of 50 simulated genomes (at 5%, 10%, 15%, and 20% divergence) generated with INDELible v1.03 [3].



Nature Biotechnology: doi:10.1038/nbt.2241

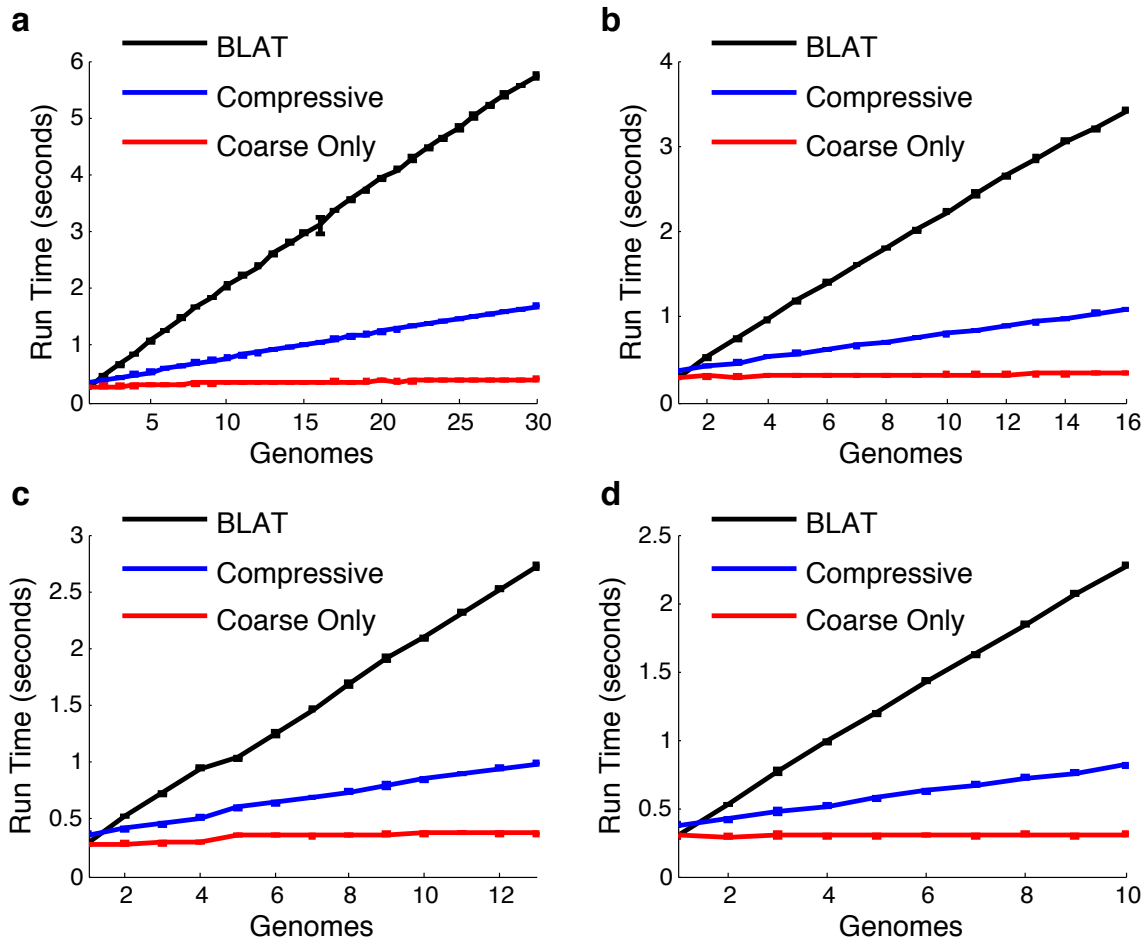


Figure S5: Performance of Compressive BLAT on databases of four bacterial genera using distinct search sets derived from each bacterial genus individually. Parameters are the same (default) as in the primary manuscript. (a) Escherichia; (b) Salmonella; (c) Yersinia; (d) Brucella.



Nature Biotechnology: doi:10.1038/nbt.2241

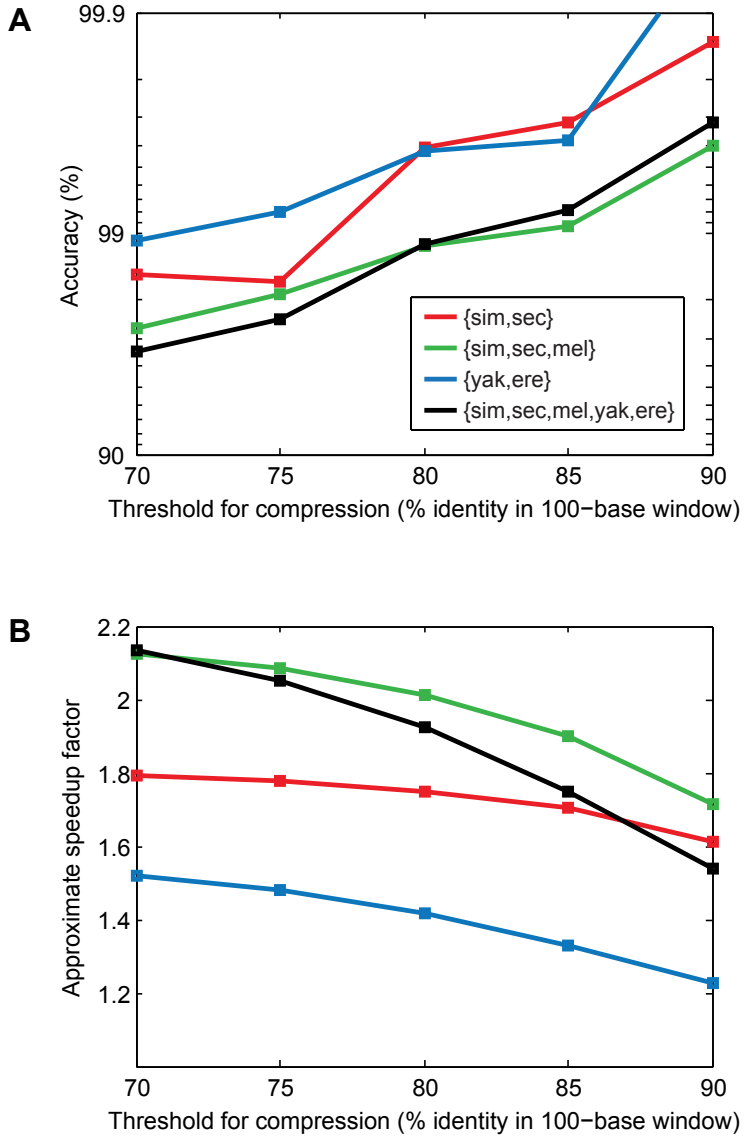


Figure S6: Trade-off between preprocessing compression and search accuracy for simulated queries on *Drosophila* subtrees. As the threshold required for compression is increased from 70 to 90% identity within each 100-base window, accuracy improves (*A*) while search speedup decreases (*B*).



Nature Biotechnology: doi:10.1038/nbt.2241

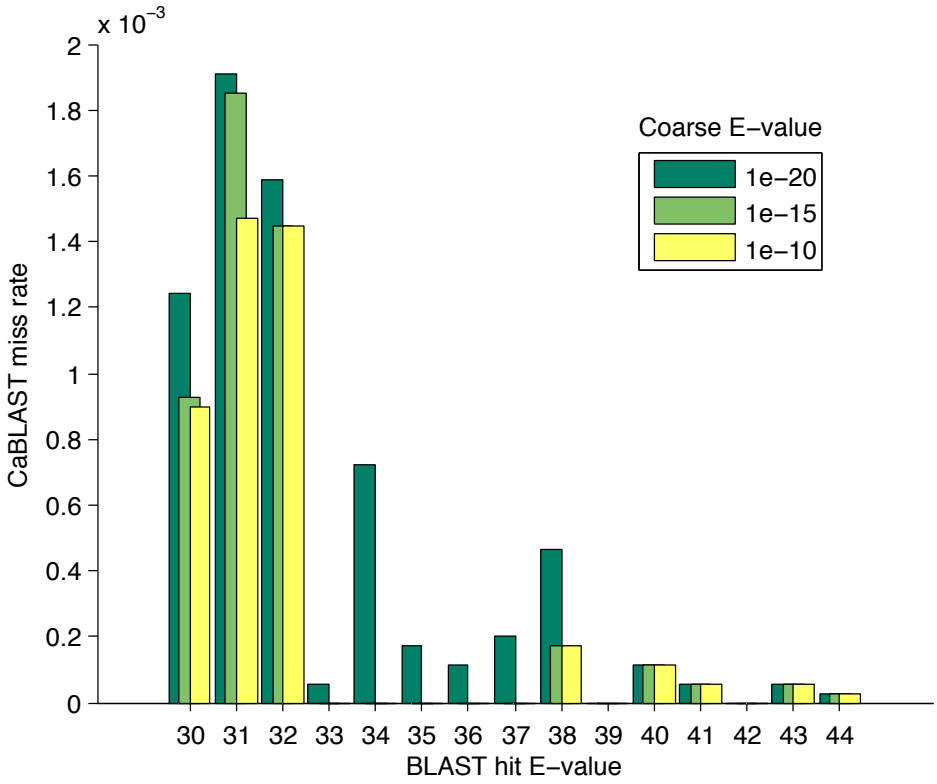


Figure S7: Analysis of missed BLAST hits. Ten thousand queries were run on the combined microbial dataset (yeast plus four bacterial genera) at three different coarse E-values and a fine E-value of 1e-30. The overwhelming majority of misses are at the margin of significance; in total these represent less than 0.5% of the hits.



Nature Biotechnology: doi:10.1038/nbt.2241

# Supplementary Table

| **Species set** | BLAST runtime (sec) | Coarse E-value | CaBLAST runtime (sec) | | Total % of BLAST runtime | Accuracy (%) |
|---|---|---|---|---|---|---|
| | | | Coarse BLAST | Fine BLAST | | |
| {**sim,sec**} | 97 | 1e-20 | 54 | 3.1 | 59 | 99.6 |
| Original size: 304 Mb | | 1e-15 | 53 | 3.6 | 58 | 100.0 |
| Unique size: 174 Mb (57.1%) | | 1e-10 | 55 | 8.0 | 65 | 100.0 |
| {**sim,sec,mel**} | 147 | 1e-20 | 72 | 4.8 | 53 | 98.9 |
| Original size: 473 Mb | | 1e-15 | 73 | 6.4 | 54 | 99.7 |
| Unique size: 235 Mb (49.7%) | | 1e-10 | 74 | 13.7 | 60 | 99.8 |
| {**yak,ere**} | 96 | 1e-20 | 67 | 2.5 | 73 | 99.6 |
| Original size: 318 Mb | | 1e-15 | 67 | 3.2 | 73 | 99.8 |
| Unique size: 225 Mb (70.5%) | | 1e-10 | 69 | 8.8 | 80 | 99.9 |
| {**sim,sec,mel,yak,ere**} | 243 | 1e-20 | 126 | 6.7 | 55 | 98.9 |
| Original size: 792 Mb | | 1e-15 | 127 | 9.0 | 56 | 99.6 |
| Unique size: 411 Mb (57.3%) | | 1e-10 | 130 | 20.6 | 62 | 99.8 |

Table S1: Detailed preprocessing and search results from CaBLAST runs on four sets of *Drosophila* genomes using final E-value 1e-30 and three choices of coarse E-value. CaBLAST runtime is dominated by the coarse BLAST step, achieving speedup relative to BLAST almost exactly proportional to the reduction from the original to the unique database (size ratio in parentheses). Accuracies nearly match those of BLAST; note that specificity is always 100% by virtue of the algorithm design: hits picked up by the fine search are by definition true BLAST hits.



Nature Biotechnology: doi:10.1038/nbt.2241

# Supplementary References


[1] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *Journal of molecular biology* **215**, 403–410 (1990).

[2] Kent, W. J. BLAT—The BLAST-Like Alignment Tool. *Genome Research* **12**, 656–664 (2002).

[3] Fletcher, W. & Yang, Z. Indelible: A flexible simulator of biological sequence evolution. *Molecular Biology and Evolution* **26**, 1879–1888 (2009).




Nature Biotechnology: doi:10.1038/nbt.2241