Compressive genomics for protein databases

BIOINFORMATICS

Vol. 29 ISMB/ECCB 2013, pages i283–i290
doi:10.1093/bioinformatics/btt214

Compressive genomics for protein databases

Noah M. Daniels[1], Andrew Gallant[1], Jian Peng[2,3], Lenore J. Cowen[1], Michael Baym[2,3,4] and Bonnie Berger[2,3,*]

[1]Department of Computer Science, Tufts University, Medford, MA 02451, [2]Department of Mathematics, [3]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 and [4]Department of Systems Biology, Harvard Medical School, Boston, MA 02115, USA

ABSTRACT

Motivation: The exponential growth of protein sequence databases has increasingly made the fundamental question of searching for homologs a computational bottleneck. The amount of unique data, however, is not growing nearly as fast; we can exploit this fact to greatly accelerate homology search. Acceleration of programs in the popular PSI/DELTA-BLAST family of tools will not only speed-up homology search directly but also the huge collection of other current programs that primarily interact with large protein databases via precisely these tools.

Results: We introduce a suite of homology search tools, powered by compressively accelerated protein BLAST (CaBLASTP), which are significantly faster than and comparably accurate with all known state-of-the-art tools, including HHblits, DELTA-BLAST and PSI-BLAST. Further, our tools are implemented in a manner that allows direct substitution into existing analysis pipelines. The key idea is that we introduce a local similarity-based compression scheme that allows us to operate directly on the compressed data. Importantly, CaBLASTP's runtime scales almost linearly in the amount of unique data, as opposed to current BLASTP variants, which scale linearly in the size of the full protein database being searched. Our compressive algorithms will speed-up many tasks, such as protein structure prediction and orthology mapping, which rely heavily on homology search.

Availability: CaBLASTP is available under the GNU Public License at http://cablastp.csail.mit.edu/

Contact: bab@mit.edu

1 INTRODUCTION

Identification of homologous sequences is of fundamental importance in computational biology. Sequence search tools, such as BLASTP and PSI-BLAST (Altschul *et al.*, 1997), have played important roles in various tasks arising in protein science, including secondary and tertiary structure prediction (Rost *et al.*, 2004; Söding *et al.*, 2005), functional annotation (Kosloff and Kolodny, 2008; Loewenstein *et al.*, 2009) and orthology mapping (Singh *et al.*, 2008; Tatusov *et al.*, 2000). The runtimes of the most popular methods [e.g. BLASTP, PSI-BLAST and DELTA-BLAST (Boratyn *et al.*, 2012)] scale nearly linearly in the size of protein databases. With the exponential increase in protein sequence data, this is becoming a major bottleneck to computation. Thus, it is imperative to design algorithms that scale sub-linearly in the size of the databases.

The recent exponential growth in genomic sequence data (Kahn, 2011; Kircher and Kelso, 2010), which is outpacing growth of computing power (Gross, 2011; Huttenhower and Hofmann, 2010; Kahn, 2011; Schatz *et al.*, 2010), has spurred an interest in *compressive genomics* (Loh *et al.*, 2012) and the need to compress sequence data for efficient storage (Brandon *et al.*, 2009; Cameron *et al.*, 2007; Chen *et al.*, 2002). Protein sequence data, although on a slower growth curve than genomic data, nonetheless increase at an exponential rate (Fig. A1), doubling roughly every 2 years, for now just keeping pace with Moore's law for computational power.

A key observation from compressive genomics is that much of the new data are actually similar to existing data, which was used to accelerate *nucleotide* sequence search without loss of accuracy (Loh *et al.*, 2012).

Despite its name, even NCBI's non-redundant protein sequence database (NR) contains a great deal of redundancy; it is non-redundant only at the level of entire sequences; highly *similar* sequences are represented separately. Thus, even NR lends itself to a compression scheme that takes advantage of this redundancy. Although NR has already eliminated exact duplicates at the global sequence level, we take advantage of *local* sequence similarity to achieve compression.

We introduce a compressive algorithm, CaBLASTP, along with an implementation that allows direct computation on the compressed data. CaBLASTP boosts the runtime performance of any search tool in the protein BLAST (Altschul *et al.*, 1997) family, while maintaining accuracy. Specifically, we show that compressive versions of BLASTP, PSI-BLAST (Altschul *et al.*, 1997) and DELTA-BLAST (Boratyn *et al.*, 2012) scale nearly linearly in the size of the unique data, as well as sub-linearly in the size of the complete protein database.

Notably, any program that relies on protein BLAST can take advantage of our compressive software with virtually no effort. Thus, we expect CaBLASTP to be of great use to the community.

2 METHODS

We introduce a framework for compressing protein sequences and performing a variety of homology search techniques in compressed space. We have designed this 'CaBLASTP' framework primarily to be compatible with the NCBI-BLAST family of software (Altschul *et al.*, 1997).

The key observation underlying CaBLASTP is that when sequences are sufficiently similar—yet not necessarily identical—tasks such as approximate search can initially operate on just one representative of the similar set. The remainder of the set need only be analyzed if a representative sequence is found to be of interest.

The basic approach of CaBLASTP thus consists of two phases. First, a pre-processing (or compression) phase identifies similarities among sequences in a protein database. This phase is computationally intensive, yet it need be done only once for a given database. After compression is

*To whom correspondence should be addressed.

complete, CaBLASTP can then translate the decreased redundancy of the database into a speed-up when performing search, which is the second phase.

CaBLASTP compresses a protein sequence database to identify regions of high similarity (Fig. 1a). This is done by first scanning through the database and categorizing sequences as either new or redundant. Owing to both the amino acid alphabet size and the sheer database size, to become tractable, this step required the development of new computationally efficient methods (detailed later in the text).

Novel sequences are stored in a 'coarse' database, whereas sequence segments that align well to previously seen sequences are not. The coarse database essentially represents only the *unique* data from the original database. Instead, records for these alignments are added to a link index. Our approach can be viewed as a hybrid between traditional data-compression algorithms, which create a dictionary for exact sequences encountered in the data, and sequence alignment algorithms, such as BLAST (Altschul *et al.*, 1997).

The search phase applies a two-stage approach (Fig. 1b). First, the query is searched against the coarse database. To maintain accuracy, this 'coarse search' uses a more permissive E-value threshold than the threshold specified for final results. For each hit from the coarse search, CaBLASTP then reconstructs any additional hit candidates by following the links in the link index. Final results are then obtained by a 'fine search' against these candidate sequences.

We have implemented a compression tool, which converts a protein sequence database to a CaBLASTP compressed database, as well as three compressive search tools that operate on this database, implementing compression-space versions of NCBI-BLAST, PSI-BLAST and the recently released DELTA-BLAST (Boratyn *et al.*, 2012). Our software is written in the publicly available Go programming language (Griesemer, 2009; Kortschak, 2011).

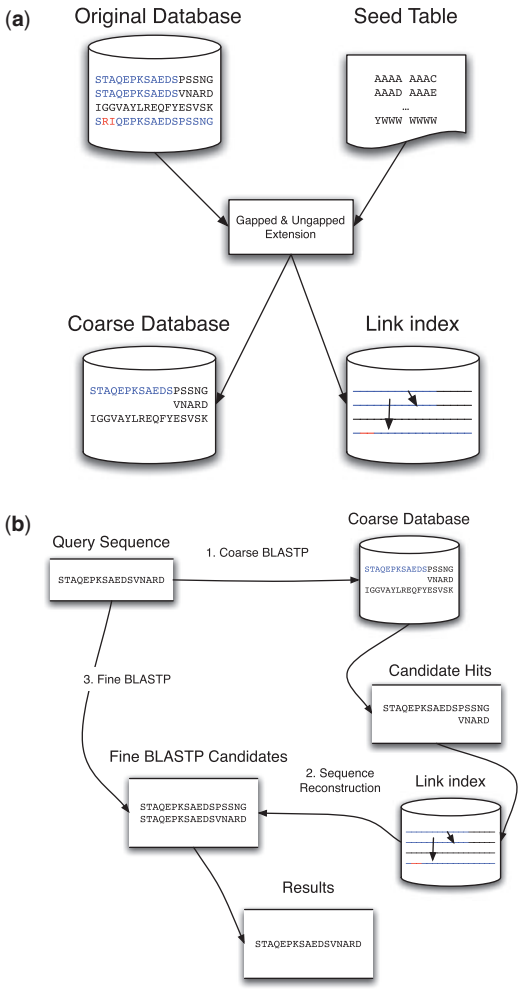

Fig. 1. (**a**) Novel sequences are stored in a 'coarse' database, whereas sequence segments that align well to previously seen sequences are not. Instead, records for these alignments are added to a link index. Our approach can be viewed as a hybrid between traditional data-compression algorithms, which create a dictionary for exact sequences encountered in the data and sequence alignment algorithms, such as BLAST. Links point from entries in the seed table to entries in the coarse database. Blue text indicates matching subsequences; red indicates differences. (**b**) The search phase applies a two-stage approach. First, the query is searched against the coarse database. To maintain accuracy, this 'coarse search' uses a more permissive E-value threshold than the threshold specified for final results. For each hit from the coarse search, CaBLASTP then reconstructs any additional candidate hits by following the links in the link index. Final results are then obtained by a 'fine search' against these candidate sequences

2.1 Compression

The compressive phase takes a protein sequence database and produces a compressed data structure amenable to the search step described in Section 2.2. This pipeline is illustrated in Figure 1a. This is implemented in the program cablastp-compress, which takes a standard FASTA file as input.

Given an input sequence database, compression proceeds as follows:

(1) First, initialize a table of all possible k-mer 'seeds' of amino acids, and a 'coarse' database of amino acid sequences, initially containing the first sequence in the input database (empirically, the best compression runtime performance occurs with k set to 4).

(2) For each k-mer of the first sequence, then create a pointer from the corresponding entry in the seed table to the position of that k-mer in the first sequence.

(3) For each sequence s in the input after the first, slide a window of size $k + k'$, where k' may be zero (empirically, best performance is achieved with k' set to 2).

(4) Low-complexity regions (single-residue repeats) of length >10 are skipped.

(5) Look up the first k residues of this window in the seed table. For every pointer corresponding to that k-mer in the seed table, follow it to a subsequence in the coarse database. If a resulting subsequence s' in the coarse database further matches the window by the additional k' residues, then attempt *extension* (see below). If no subsequences from this window can be extended, move the window by one residue. The separation of the window size into k and k' is simply an optimization to reduce the memory footprint of compression; it allows, for example, an effective window size of 6 while only requiring a seed table with 20^4 rather than 20^6 entries.

(6) If a match was found via extension, move the k-mer window to the first k-mer in s after the match, and the extension process repeats with this new seed.

Extension. Given a $k + k'$ match between the sequence s and subsequence s' pointed to by the seed table, first attempt *ungapped* extension:

(1) Greedily extend the match into an ungapped alignment as far as possible.

(2) Within each window of 10 residues, if identical 4mers in s and s' can be found, and at least two additional matching residues can be

found, then there is an ungapped match within that 10mer window between s and s' that exhibits at least 60% sequence identity.

(3) Continue ungapped matching using 10mer windows until no more 60% identity 10mers are found.

(4) The result of ungapped extension is that there is an alignment between sequences s and s' with no insertions or deletions, only matches and substitutions, and at least 60% of the positions contain exact matches.

When ungapped extension can no longer proceed, switch to *gapped* extension. From the end of the ungapped alignment, align 25mer windows of both s and s' using the Needleman–Wunsch (Needleman and Wunsch, 1970) algorithm with BLOSUM62 as a cost matrix. We use a variant of Needleman–Wunsch, implementing constrained dynamic programming, prohibiting more than six gaps in the alignment, reducing the search space by a factor of ~4. Global alignment is chosen because we wish to attempt to align the entire 25mer from each sequence. After gapped extension on a window length of 25, attempt ungapped extension again.

When neither gapped nor ungapped extension can continue, terminate extension. Realign the resulting extension of s and s', again using Needleman–Wunsch. If the resulting alignment has <70% sequence identity or is <40 residues, discard it, instead attempt extension on the next link in the seed table for the original k-mer; if there are no more links for that k-mer, then consider the next k-mer. If, however, the resulting alignment has at least 70% sequence identity and is at least 40 residues long, then create a link from the entry for s' in the coarse database to the subsequence of s beginning with the original k-mer and corresponding to the extended region. If there are 'dangling' ends to s <30 residues that did not satisfy the extension criteria, append them to the match. Longer 'dangling' ends that did not match any subsequences reachable from the seed table are added into the coarse database themselves, with links from the relevant seeds in the seed table to their constituent k-mers. The requirement to deal with protein sequences being discrete represents a difference from Loh *et al.* (2012).

Any sequence or subsequence in the input that cannot be matched to earlier sequences in the coarse database will itself become an entry in the coarse database, with pointers from the k-mer seed table linking to it, and similar sequences seen later in the input may be matched to it.

In addition, a *difference script* is associated with this link. The difference script is simply a representation of the insertions, deletions and substitutions resulting from the overall Needleman–Wunsch alignment. Applying the difference script to a representative sequence in the coarse database (s' above) will return the sequence s; it is effectively decompressed. Similarly, applying the difference script to s will return its representative s'.

After all sequences have been compressed, the sequences in the coarse database are written out in FASTA format; the resulting coarse FASTA file, which is smaller than the original input file, is used by all search implementations described later in the text. In addition, the set of links between coarse sequences and original sequence identifiers and their difference scripts is written to disk in a binary format. An index file is also produced, which maps the sequence identifiers from the coarse database to entries in the compressed database. These formats are documented in the Go source code for CaBLASTP. It is worth noting that the compression format is lossless and completely invertible; it is possible to exactly reconstruct the original FASTA source from the compressed database.

When compressing a large amino acid data set such as NCBI's 'NR', memory usage can grow large. As a memory and runtime performance optimization, the seed table can be reset when it reaches a user-specified size, 8 GB by default. For our experiments, we used a maximum seed table size of 20 GB. When no limit was imposed, the seed table could grow to >40 GB on NR, but we saw negligible difference in compression ratio between these two limits.

On the compressed database described here, we have implemented three search techniques, BLASTP, PSI-BLAST and DELTA-BLAST. All three follow the same basic two-step technique (Fig. 1b): first, they search the compressed database with a relaxed threshold to find candidate matches, and then the closely related sequences to the candidate hits are more closely examined. The fundamental speed-up introduced by this two-step approach is that the initial step rules out the vast majority of the original database without ever having to examine it.

2.2 Search

2.2.1 Compressive BLASTP Compressive accelerated BLASTP, or cablastp-search, requires a compressed database produced by our compression method as described earlier in the text. Given a query sequence and a compressed database, this search method calls the BLASTP program to search the coarse FASTA file, which is typically much smaller than the original FASTA file. This step is called *coarse search*, as suggested by Loh *et al.* (2012). Coarse search uses a relaxed E-value threshold compared with what would be desired if the entire original database was searched using standard BLASTP. The idea behind coarse search is to identify *possible* hits, which may be rejected by the later *fine* search. Because the coarse FASTA file is a subset of the original, uncompressed FASTA file, potential hits may be subsequences that are shorter than or slightly different from the original sequences they represent. Thus, a more permissive E-value must be used. Command-line arguments to be passed to BLASTP itself may be specified by the user. The results of the coarse search are sequences from the coarse FASTA file; thus, they are actually sequences or subsequences from the original FASTA file. Based on the compressed database's search index, each of these sequences is then *reconstructed* into all corresponding sequences from the original database, by following the links to original sequence matches, and applying their difference scripts. Note that the coarse FASTA file need not ever be decompressed in its entirety, although it is possible to do so. The resulting set of sequences, larger than the resulting set from the coarse search, is then provided to BLASTP as the subject for a second query, which again uses the query sequence provided to cablastp-search. This step is called *fine search*, and it produces a set of final results, based on an E-value threshold specified by the user (or the BLASTP default). These results are provided in an identical format to BLASTP. This implementation of cablastp-search relies on the BLAST+ implementation (developed and tested against BLAST+ 2.2.6 and 2.2.7).

2.2.2 Compressive PSI-BLAST Compressively accelerated PSI-BLAST, or cablastp-psisearch, operates much like compressively accelerated BLASTP. PSI-BLAST builds a position-specific scoring matrix, or PSSM, iteratively, by running BLAST searches for a query against a database. Instead of just using the BLOSUM-62 matrix to compute alignment scores, PSI-BLAST computes substitution scores column-by-column, based on an initial alignment and subsequent refinements. cablastp-psisearch takes advantage of the PSI-BLAST program's ability to save a *checkpoint* of its PSSM to a file. Given a user-specified number of iterations, the program performs both a coarse and a fine search for each iteration. Every iteration, except the first, relies on a PSSM file output by the previous iteration, whereas every iteration, except the final, writes a PSSM file for the next iteration to use. Each iteration comprises a coarse and a fine search identical to cablastp-search, but using the PSI-BLAST executable.

2.2.3 Compressive DELTA-BLAST Domain-enhanced look up time accelerated BLAST, or DELTA-BLAST (Boratyn *et al.*, 2012), uses a library of pre-computed PSSMs based on NCBI's Conserved Domain Database. The DELTA-BLAST executable is included with BLAST+ 2.2.6 and later versions. Compressively accelerated DELTA-BLAST, or cablastp-deltasearch, operates similarly to compressively accelerated BLASTP, performing a single iteration of search comprising

a coarse and a fine search step. We did not implement an iterative version of this algorithm, as Boratyn *et al*. (2012) showed decreased accuracy with iteration.

2.3 Accuracy validation

To verify that compressive acceleration does not significantly harm the accuracy of BLASTP, PSI-BLAST and DELTA-BLAST, we performed 100 random searches against the NR database, for each of these three tools. For each tool, we treated the results from the standard version (e.g. BLASTP) as a gold standard, and computed the true positive rate and false positive rate for compressive versions of the same search (e.g. cablastp-search) with respect to this gold standard. We performed this search with an E-value threshold of 10^{-3}, for both the coarse and fine threshold for the compressive versions of each search, and for CaBLASTP, PSI-BLAST and DELTA-BLAST. Because of the design of the algorithm, false positives with respect to the non-compressively accelerated tools are not possible.

We were also interested in homology detection performance of our compressive implementations of PSI-BLAST and DELTA-BLAST with respect to HHblits (McDonnell *et al*., 2006). We identified all 1123 sequences from the ASTRAL subset of release 1.75A of the Structural Classifications of Proteins (SCOP) (Murzin *et al*., 1995) database that were not present in HHblits' 'NR20' database or the August 2010 NCBI NR database, but whose SCOP families contained other homologous sequences that were present in those databases. We chose the August 2010 NCBI NR database to more fairly compare with the August 2011 HHblits NR20, which is the most recent available. We then performed searches using one iteration of HHblits, one iteration of cablastp-deltasearch and two iterations of cablastp-psisearch against these databases. We chose these numbers of iterations because a single iteration of PSI-BLAST is effectively just BLASTP, whereas Boratyn *et al*. (2012) showed decreased accuracy with more than one iteration of DELTA-BLAST. Multiple iterations of HHblits would have resulted in slower runtime performance. We considered results from the same SCOP superfamily (and by extension, the same SCOP family) as the query to be true positives, and results from different SCOP folds to be false positives. We removed results from the same SCOP fold but different superfamilies, as it is not consistent across the SCOP fold classifications whether those sequences are homologs. We also removed results that were not identifiable in SCOP. We plotted ROC curves based on these homology predictions. We also report the mean running times of these searches.

3 RESULTS

3.1 Scalability on simulated data

We first compared the performance of our compressive accelerated versions of BLAST with their original implementations. We constructed a simulated dataset to mimic the expected growth of a protein sequence database into the future, to demonstrate CaBLASTP's ability to scale to large datasets. We began with all known and putative proteins in the *Saccharomyces* Genome Database (Cherry *et al*., 2012), which contains the proteomes of 21 strains of yeast. To simulate clades of recently diverged species, we used a tool for simulating protein mutation (Daniels *et al*., 2012; Kumar and Cowen, 2009, 2010). For each original sequence in the database, we added 5, 10, 20, 30 or 40 similar sequences by substituting residues with a mutation rate of 20%, based on the BLOSUM62 substitution matrix. The original dataset contained 6717 sequences; with 40 mutated copies of each sequence, the database contained 275 397 sequences. In this way, we essentially 'simulate' an evolutionary process to build a number of 'putative' proteomes from *Saccharomyces* proteomes. Performance of sequence search on these augmented databases should be comparable with the performance on future databases where closely related species have now been sequenced, producing increasing numbers of orthologous sequences. We benchmarked sequence search on these augmented databases.

Figure 2a demonstrates the superior runtime of CaBLASTP over BLASTP for large datasets. The results are averaged over all sequences from the native *Saccharomyces* proteome. The runtime of BLASTP increases almost linearly in the number of 'simulated' proteomes, or the size of the full database. In contrast, CaBLASTP scales sub-linearly with database size, even when there are 40 times as many proteomes. Notably, CaBLASTP achieves roughly constant runtime regardless of database size. These results show that our compressive scheme is able to exploit data redundancy, thereby avoiding redundant searches. Finally, we have performed similar comparisons on datasets with different mutation rates (e.g. 5, 10 and 30%), and the results are similar. This benchmark was performed on a quad-core Intel Core i7 with 16 GB random access memory and a solid-state disk.

3.2 Homology search on real data

3.2.1 Speed We evaluated the homology-search performance of both the original and our compressive BLAST versions on the widely used NR database. We randomly chose 100 sequences from the December 2012 NR database. Five runs for each query sequence were performed on three early versions of NR built on June 2010, July 2012 and December 2012, with a coarse E-value of 10^{-5} and a fine E-value of 10^{-10} (we selected these three NR datasets because we do not have access to any other versions). The average runtime for each method is shown in Figure 2b. This benchmark was run on a system with dual six-core AMD Opteron 2427 processors and 32 GB random access memory, equipped with a RAID-10 disk array.

Although on each NR dataset, BLASTP takes 120, 200 and 240 s, respectively, CaBLASTP takes only 50, 70 and 75 s, respectively. Given that the NR datasets each contain 11.6, 19.1 and 22 million sequences, BLASTP scales almost exactly linearly in database size, whereas the runtime of CaBLASTP grows much more slowly. CaBLASTP is faster than BLASTP by factors of 2.4, 2.7 and 3.1 on these NR datasets, respectively. These results fit with the observation that the uncompressed NR databases are 6.1, 11 and 13 GB in size, respectively, whereas their compressed counterparts are 1.4, 2.4 and 2.7 GB in size. Considering that the NR databases already have 100% global sequence-identity redundancy removed, CaBLASTP takes advantage of the local similarity within the databases to speed-up homology search. It is worth noting that on the NR databases, the 'coarse' search step of CaBLASTP dominates the running time; the 'fine' step requires <1 s in all cases.

Similar to the comparison between BLASTP and CaBLASTP, the compressive accelerated versions of both PSI-BLAST and DELTA-BLAST are much faster than their original versions. We performed two iterations of PSI-BLAST and one iteration of DELTA-BLAST, as suggested in the latter's original article. The acceleration ratio increases as the size of NR grows (Fig. 2b).

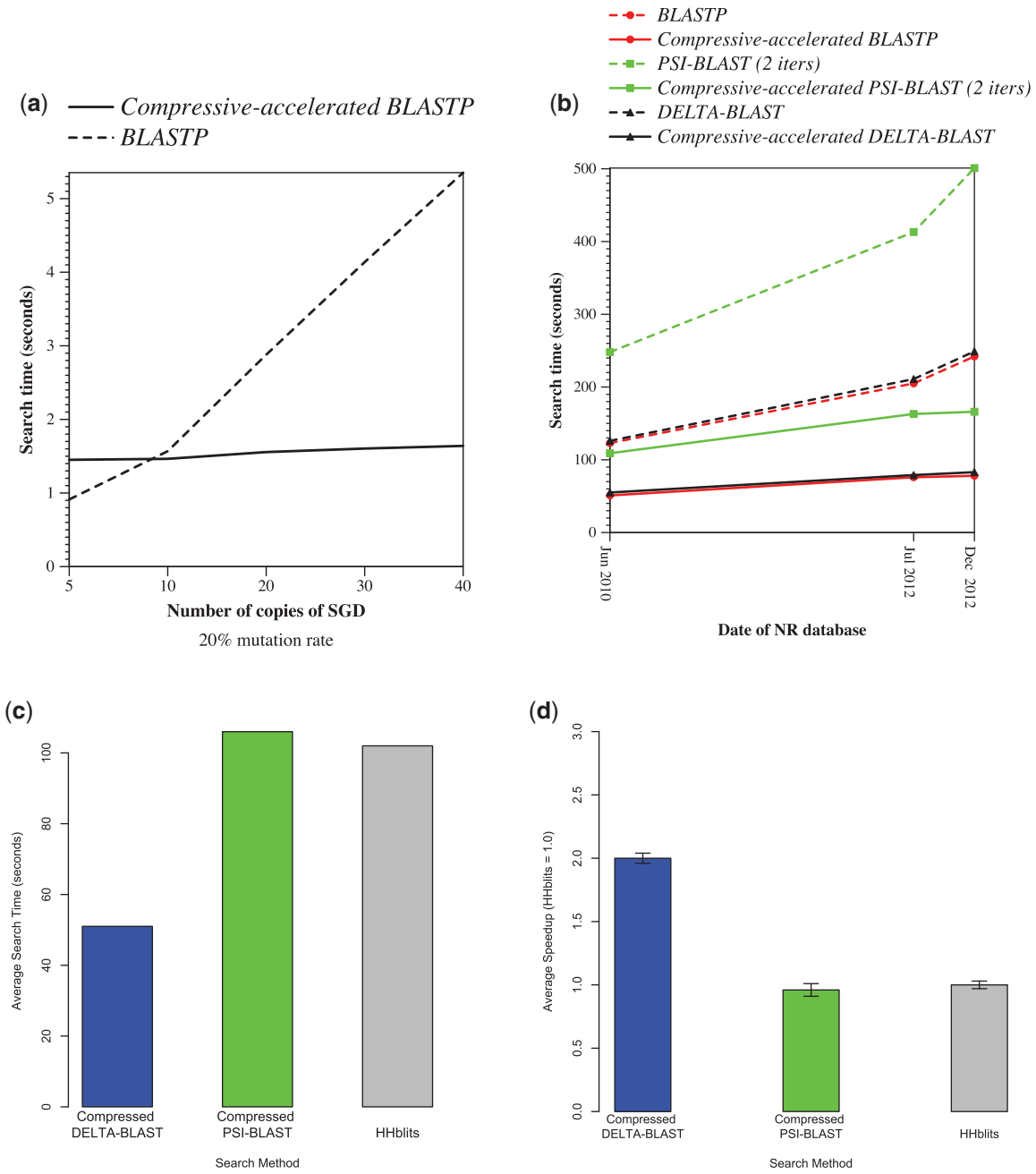

Fig. 2. (**a**) Runtime of CaBLASTP versus BLASTP as datasets grow because of simulated mutation. Below 20% mutation rate, CaBLASTP run time is virtually constant. (**b**) Runtime of cablastp-search versus BLASTP on three historical versions of NCBI's 'NR' database. Times are the mean of five runs each for 100 randomly chosen queries. (**c**) Runtime of cablastp-deltasearch versus cablastp-psisearch (two iterations) on NR from August 2010 and HHblits on NR20 from August 2011. Times are the mean of five runs each for 100 queries from NR from December 2012. (**d**) Relative speed-up of cablastp-deltasearch and cablastp-psisearch (two iterations) versus HHblits (one iteration) on NR from August 2010 and HHblits on NR20 from August 2011

3.2.2 Accuracy To verify that compressive acceleration does not decrease the accuracy of BLASTP, PSI-BLAST and DELTA-BLAST, we also compared the differences between the sequence hits from the above random query searches with the NR databases for each tool. Specifically, we compared the overlap between the sequence hits found by the compression-accelerated versions and those identified by the original versions. It is worth noting that because of the boosting compressive scheme we have designed, our algorithms will not find any sequences that do not appear in the hits of their original counterparts. We then calculated the overlap between the alignments generated by our compression-accelerated tools and their original versions. Table 1 depicts that the overlap of sequence hits is >99% and that of alignments is 100%. In other words, when a

Table 1. Accuracy of compressive tools

Program	TPR (%)	FPR (%)	Alignment accuracy (%)
Compressive BLASTP	99.4	0	100
Compressive PSI-BLAST	99.3	0	100
Compressive DELTA-BLAST	99.4	0	100

Note: TPR is the fraction of hits from standard versions of each tool that were also found by the compressive versions. FPR is the fraction of hits from the compressive versions that were not found by the standard versions. Note: because of the algorithm design, false positives with respect to the standard uncompressed tools are not possible.

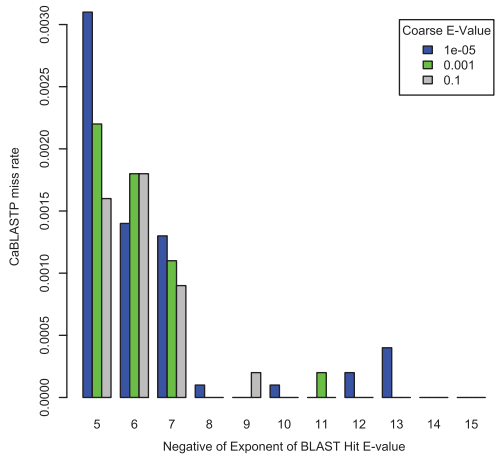

Fig. 3. Analysis of missed BLASTP hits. One thousand queries were run on the yeast genome database at three different coarse E-values and a fine E-value of 1E-5. The majority of misses are at the margin; in total, these represent <0.5% of the hits

hit is found, the alignment perfectly matches the standard BLASTP alignment. An analysis of the differences in the search results suggests that short query sequences (<40 residues) may in some cases return no hits in the coarse search. Changing the minimum match length in the compression phase would likely address this issue, yet likely at the expense of a significant fraction of the runtime performance gains.

To better gauge the impact of coarse search E-value on accuracy, we performed 1000 random queries against the yeast database, with a fine E-value of 10^{-5} and three different coarse E-values: 10^{-1}, 10^{-3} and 10^{-5}. We compared these results with standard BLASTP queries with an E-value of 10^{-5}. Figure 3 illustrates the results of this analysis; CaBLASTP is robust to choice of coarse E-value, as long as the coarse E-value is more permissive than the fine E-value.

3.2.3 Comparison with HHblits Finally, we compared the performance of homology detection of our compressively accelerated implementations of PSI-BLAST and DELTA-BLAST with a recently introduced profile-based search tool, HHblits (Remmert *et al.*, 2012). By partitioning sequences into clusters based on global sequence similarity, HHblits pre-computes discretized hidden Markov models (HMMs) on each cluster and

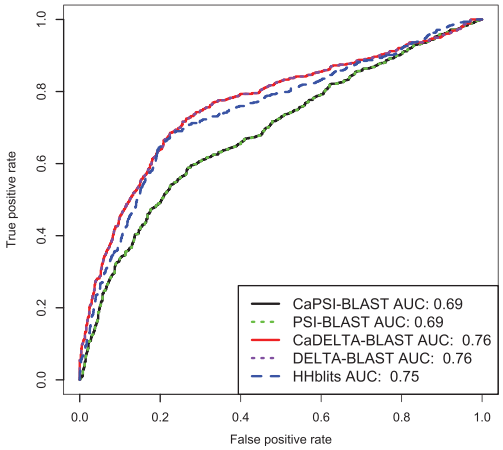

Fig. 4. Accuracy: ROC curves for homology detection performance of cablastp-psisearch versus cablastp-deltasearch and HHblits, as well as standard PSI-BLAST and DELTA-BLAST

only searches a query against those HMMs. In contrast, our compression-accelerated algorithms take the local similarity into account to speed-up sequence search. For comparison, we identified all 1123 sequences from the ASTRAL subset of release 1.75A of the SCOP (Murzin *et al.*, 1995) database that are not present in HHblits' 'NR20' database or the August 2010 NCBI NR database, and which were in SCOP families that *did* contain other non-identical sequences in those older NR databases. We chose the August 2010 NCBI NR database to more fairly compare with the August 2011 HHblits NR20, which is the most recent available. We then performed searches using one iteration of HHblits, one iteration of cablastp-deltasearch and two iterations of cablastp-psisearch. The numbers of iterations were chosen to ensure the performance of these tools is similar according to previous reports (Boratyn *et al.*, 2012; Remmert *et al.*, 2012). We considered top sequence hits from the same SCOP superfamily (and by extension, the same SCOP family) as the query to be true positives, and hits from different SCOP folds to be false positives. We removed sequence hits from the same SCOP fold but different superfamilies, as it is questionable whether those sequences are homologous. We also removed results that were not identifiable in SCOP. We reported the mean running times of these searches and plotted ROC curves based on the homology predictions. Figure 2c illustrates these results. Finally, we reported the *speed-up* of cablastp-deltasearch and cablastp-psisearch with respect to HHblits. Speed-up is calculated as the mean, over all queries, of the mean HHblits time for a given query divided by the mean time for the specified search for that query. Error bars represent a 95% confidence interval based on the distribution of search times for each query sequence. Figure 2d illustrates these results.

HHblits takes an average of 102 s for one iteration. cablastp-deltasearch takes an average of 51 s for one iteration. cablastp-psisearch needs 52 s for one iteration and 106 s for two iterations. Compression-accelerated DELTA-BLAST is twice as fast as HHblits on this test; CaBLASTP-PSI-search is slightly slower than HHblits. The result is notable considering that the clustered NR20 by HHblits is much smaller than the NR database we used. Moreover, as shown in Figure 4, compressive DELTA-

BLAST achieves an area under the ROC curve of 0.76, compared with 0.75 for HHblits and 0.69 for compressive PSI-BLAST. In a ROC_5 analysis (Fig. A2), where only the area under the curve up to the fifth false positive is considered, and the area is normalized, compressive DELTA-BLAST achieves a ROC_5 score of 0.82, compared with 0.71 for HHblits and 0.63 for compressive PSI-BLAST.

We also ran the original versions of DELTA-BLAST and PSI-BLAST on the same set of query sequences. Their results are identical to our compression-accelerated versions, but their runtimes are roughly three times slower.

4 DISCUSSION

We have introduced a compression-accelerated search algorithm that boosts the speed while maintaining accuracy of tools in the protein BLAST family. Our approach scales sub-linearly with the size of the database being searched, and linearly with the size of the *unique* data. We expect that as the NR database continues to grow exponentially, the benefits of this compressive approach will become more pronounced.

In contrast to genomic sequence compression (Loh *et al.*, 2012), which appears on its surface to be similar, subtle differences make protein sequence compression a different problem. The primary difference is that proteins have a larger alphabet, and thus, random sequences will have less similarity. This results in different parameters and compression ratios, but it also increases the computational complexity of compression, as the number of k-mers is exponential in the alphabet size. Another difference is that protein sequences are discrete; therefore, our compression algorithm must handle sequence beginnings and ends.

We have demonstrated that our compressive approach provides significant gains as the redundancy of the data increases, but we also see future challenges. As the NCBI's NR database continues to grow in the coming years, the size of each cluster of similar subsequences will also grow. We expect that for compression to remain tractable, further algorithmic and software-engineering improvements, for example, a hierarchical compression scheme, will be required.

Many sophisticated homology search and protein structure prediction tools require BLAST searches of one type or another to incorporate sequence profiles or structural information to improve performance (Moult *et al.*, 2011). For example, when we introduced the BetaWrapPro method (McDonnell *et al.*, 2006), which requires a BLASTP search at query time, NCBI's NR database contained <4.5 million sequences; today it contains >22 million sequences; thus, search requires approximately five times the running time.

Although the original motivation for developing our compressive approach was the growing running time of BLASTP searches on NR, the results described in Figure 2a suggest that our approach may also be useful for orthology mapping across organisms, performing an all-against-all search between a query proteome and a set of well-studied proteomes (Chen *et al.*, 2007; Hachiya *et al.*, 2009; Moreno-Hagelsieb,G. and Latimer, 2008), which takes an inordinate amount of time.

Our tools can be readily incorporated into these applications to accelerate their search, pre-processing or library construction. Our software can be easily interfaced with any programs that use protein BLAST search tools. Another important advantage of our methods is that the compressed database can be incrementally maintained to keep current with new proteomic sequence data.

ACKNOWLEDGEMENTS

The authors thank Norman Ramsey and Po-Ru Loh for helpful discussions about the compression approach.

Funding: This work was partially supported by a grant from the Simons Foundation and the NIH (to B.B.). N.D., A.G. and L.C. were funded in part by NIH grant (R01GM080330). M.B. was funded in part by an NSF MSPRF grant.

Conflict of Interest: none declared.

REFERENCES

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

Boratyn,G.M. *et al.* (2012) Domain enhanced lookup time accelerated BLAST. *Biol. Direct.*, **7**, 12.

Brandon,M.C. *et al.* (2009) Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, **25**, 1731–1738.

Cameron,M. *et al.* (2007) Clustered sequence representation for fast homology search. *J. Comput. Biol.*, **14**, 594–614.

Chen,F. *et al.* (2007) Assessing performance of orthology detection strategies applied to eukaryotic genomes. *PloS One*, **2**, e383.

Chen,X. *et al.* (2002) DNACompress: fast and effective DNA sequence compression. *Bioinformatics*, **18**, 1696–1698.

Cherry,J.M. *et al.* (2012) Wong. Saccharomyces Genome Database: the genomics resource of budding yeast. *Nucleic Acids Res.*, **40**, D700–D705.

Daniels,N.M. *et al.* (2012) Cowen. SMURFLite: combining simplified Markov random fields with simulated evolution improves remote homology detection for beta-structural proteins into the twilight zone. *Bioinformatics*, **28**, 1216–1222.

Griesemer,R. *et al.* (2009) The GO Programming Language. http://golang.org/ (3 June 2013, date last accessed).

Gross,M. (2011) Riding the wave of biological data. *Curr. Biol.*, **21**, R204–R206.

Hachiya,T. *et al.* (2009) Accurate identification of orthologous segments among multiple genomes. *Bioinformatics*, **25**, 853–860.

Huttenhower,C. and Hofmann,O. (2010) A quick guide to large-scale genomic data mining. *PLoS Comput. Biol.*, **6**, e1000779.

Kahn,S.D. (2011) On the future of genomic data. *Science*, **331**, 728–729.

Kircher,M and Kelso,J. (2010) High-throughput DNA sequencing–concepts and limitations. *BioEssays*, **32**, 524–536.

Kortschak,R.D. (2011) BioGo. http://github.org/kortschak/biogo (3 June 2013, date last accessed).

Kosloff,M. and Kolodny,R. (2008) Sequence-similar, structure-dissimilar protein pairs in the PDB. *Proteins*, **71**, 891–902.

Kumar,A. and Cowen,L. (2009) Augmented training of hidden Markov models to recognize remote homologs via simulated evolution. *Bioinformatics*, **25**, 1602–1608.

Kumar,A. and Cowen,L. (2010) Recognition of beta-structural motifs using hidden Markov models trained with simulated evolution. *Bioinformatics*, **26**, i287–i293.

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

Loh,P.R. *et al.* (2012) Compressive genomics. *Nat. Biotechnol.*, **30**, 627–630.

McDonnell,A. *et al.* (2006) Fold recognition and accurate sequence-structure alignment of sequences directing -sheet proteins. *Proteins*, **63**, 976–985.

Moreno-Hagelsieb,G. and Latimer,K. (2008) Choosing BLAST options for better detection of orthologs as reciprocal best hits. *Bioinformatics*, **24**, 319–324.

Moult,J. *et al.* (2011) Critical assessment of methods of protein structure prediction (CASP)—round IX. *Proteins*, **79** (**Suppl. 10**), 74–90.
Murzin,A. *et al.* (1995) SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, **247**, 536–540.
Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.
Remmert,M. *et al.* (2012) HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nat. Methods*, **9**, 173–175.
Rost,B. *et al.* (2004) The PredictProtein server. *Nucleic Acids Res.*, **32**, W321–W326.
Schatz,M.C. *et al.* (2010) Cloud computing and the DNA data race. *Nat. Biotechnol.*, **28**, 691–693.
Singh,R. *et al.* (2008) Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci. USA*, **105**, 12763–12768.
Söding,J. *et al.* (2005) The HHpred interactive server for protein homology detection and structure prediction. *Nucleic Acids Res.*, **33**, W244–W248.
Tatusov,R.L. *et al.* (2000) The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, **28**, 33–36.

APPENDIX

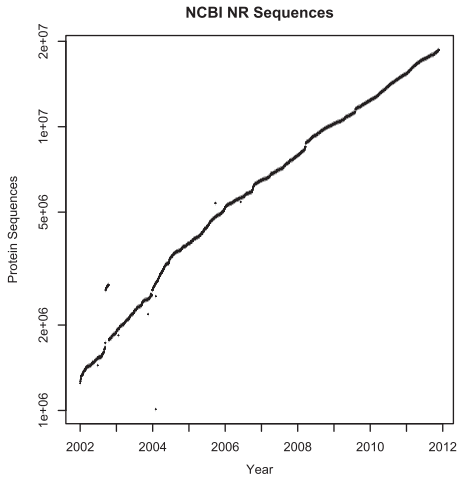

Fig. A1. Number of sequences in NCBI's 'NR' non-redundant protein sequence database from 2002 to 2012. The *y*-axis is logarithmic; doubling time is ∼2 years

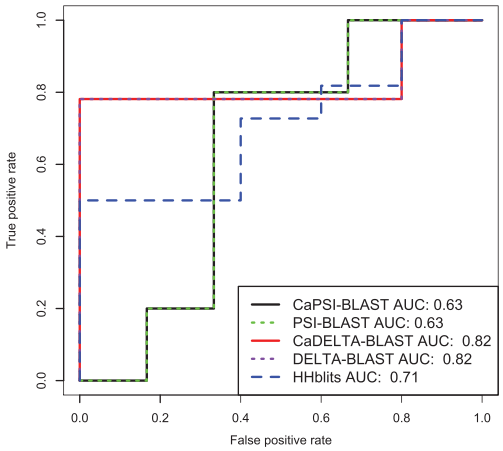

Fig. A2. ROC_5 analysis of homology detection performance