DivergentSet, a Tool for Picking Non-redundant Sequences from Large Sequence Collections*

Jeremy Widmann‡§, Micah Hamady§¶, and Rob Knight‡¨

DivergentSet addresses the important but so far neglected bioinformatics task of choosing a representative set of sequences from a larger collection. We found that using a phylogenetic tree to guide the construction of divergent sets of sequences can be up to 2 orders of magnitude faster than the naive method of using a full distance matrix. By providing a user-friendly interface (available online) that integrates the tasks of finding additional sequences, building and refining the divergent set, producing random divergent sets from the same sequences, and exporting identifiers, this software facilitates a wide range of bioinformatics analyses including finding significant motifs and covariations. As an example application of DivergentSet, we demonstrate that the motifs identified by the motif-finding package MEME (Motif Elicitation by Maximum Entropy) are highly unstable with respect to the specific choice of sequences. This instability suggests that the types of sensitivity analysis enabled by DivergentSet may be widely useful for identifying the motifs of biological significance. Molecular & Cellular Proteomics 5:1520–1532, 2006.

The problem of picking a representative non-redundant set of sequences in a convenient manner is critical for many bioinformatics analyses. Many sequence analysis methods assume that protein or nucleic acid sequences have had sufficient time to reach equilibrium such that unimportant residues or associations have mutated away, and only functionally important sites remain intact. These methods include identifying functional motifs (1) and detecting correlated evolution in functionally related residues (2). Sequences that resemble each other primarily because of shared ancestry rather than because of functional constraints must be excluded from these analyses.

The restriction on sequence identity arises because methods that detect similar patterns against a random background model cannot determine whether sequences are similar only because they are conserved from sequences that have had insufficient time to evolve in different directions. This effect is especially important when using shared motifs to define superfamilies (3, 4). Phylogenetic analyses of large numbers of sequences can also be extraordinarily time-consuming even with efficient algorithms (5). Choosing a smaller but representative set of sequences often provides the same phylogenetic tree far more efficiently (6, 7).

Because picking a divergent set manually is laborious, little attention has been paid to the reproducibility of programs that rely on divergent sets. How much does the taxon sampling or the precise choice of sequences from the same protein or RNA families affect the apparent functional motifs or relationships? In this study, we demonstrate one use of DivergentSet by comparing the motifs found by the popular motif-finding program MEME1 (1) using different sets of divergent sequences from the same initial alignment.

Divergent sets of sequences have typically been chosen manually. In this procedure, the distance between each pair of sequences is calculated. Any sequence that is too similar to a sequence already in the set is discarded. Taxonomic information can also be used, e.g. by taking one sequence from each genus. However, both methods based on taxonomic annotations and existing methods based on sequence similarity have substantial drawbacks. Thus, an automated method to choose sets of sequences based on sequence similarity is highly desirable.

To our knowledge, no fully automated system for choosing a divergent set based on sequence distances exists. Several programs, including BLASTCLUST, part of the BLAST package (8); nrdb90.pl, a script for removing nearly identical protein sequences from a database (9); and DOTUR (10) and FastGroup (11), two programs for choosing “operational taxonomic units” (OTUs) by sequence similarity, can assist to some extent in choosing divergent sets by identifying groups of related sequences. However, DOTUR and FastGroup were designed for community analysis and do not facilitate the task of randomly choosing representative sequences from each OTU. This task must still be completed by a time-consuming manual process that can take an experienced researcher many days in part because it is often desirable to include sequences from particular, well studied organisms. The ability to include sequences manually is especially desirable if rep-

---

1 The abbreviations used are: MEME, Motif Elicitation by Maximum Entropy; BLAST, Basic Local Alignment Search Tool; KEGG, Kyoto Encyclopedia for Genes and Genomes; LysRS, lysyl-tRNA synthetase; PSI-BLAST, Position-Specific Iterated Basic Local Alignment Search Tool; OTU, operational taxonomic unit; UPGMA, unweighted pair group method with arithmetic mean; CPU, central processing unit; PBS, portable batch system.
representative crystal structures are available in the Protein Data Bank (12).

Choosing sequences based on taxonomy is especially problematic because a “genus” or a “family” represents a very different level of sequence divergence in different taxa. For example, genera of mammals and birds contain notoriously little sequence divergence, whereas bacteria are especially diverse. The mitochondrial small subunit rRNA in humans and chimpanzees is over 99% identical at the sequence level, but these species are in separate genera; in contrast, small subunit rRNA from species within the bacterial genus *Mycoplasma* can differ by more than 10% (representative sequences from GenBank™, data not shown).

Ideally an automated system for choosing a divergent set of sequences should support manual adjustment of the divergent set. For example, disease-causing mutants might cause problems for motif analysis because the deleterious changes are likely to be at the active site. We identified seven essential design requirements for the system. The system should:

- Pick different sets of divergent sequences from an existing set of nucleotide or protein sequences.
- Find additional related sequences given a seed sequence or set of seed sequences to ensure the set is unbiased. Existing tools, such as DOTUR (10) and FastGroup (11) are designed for picking OTUs from a set of sequences known to be homologous and are not suited to this task.
- Refine divergent sets of sequences and pick additional sets of divergent sequences using a phylogenetic tree as a guide. Viewing the candidate divergent set in the context of a phylogeny is a major advantage over existing tools.
- Be fast and scalable even with thousands of sequences. This is particularly important for sensitivity analyses where users can iteratively refine the set of divergent sequences.
- Easily control the sequences that are included in the final divergent set (e.g. to ensure that user-selected sequences or sequences with crystal structures are preferentially included in the divergent set). This includes the ability to highlight sequences that contain or lack specific keywords in their annotations and to highlight on the tree the original sequences that were entered.

Take as input and produce as output accessions (including Enzyme Commission (EC) numbers and KEGG orthology codes), GIs, or full sequences in FASTA format.

Choose both random divergent sets (i.e. sets chosen at random from all possible sets in which no sequence could be added without being too similar to an existing sequence) and maximal divergent sets (i.e. divergent sets that contain the maximum possible number of sequences).

DivergentSet (available at bmf.colorado.edu/divergentset) is a web-based tool that meets these requirements. DivergentSet combines an easy to use user interface with a powerful Linux cluster back-end. To optimize the performance of the system, we compared several exact and approximate methods of calculating pairwise distances between sequences. This was critical because algorithms for clustering sequences spend most of their time performing pairwise comparisons. We also tested several methods of gathering additional homologs of the sequences in the initial set.

The overall workflow that DivergentSet supports is shown in Fig. 1. This workflow starts with a single identifier or sequence and produces a divergent set of sequences that can be used for downstream analyses such as motif finding. The user begins with a single sequence, identifier, or set of sequences. Additional sequences are recovered by BLAST
(8), PSI-BLAST (13), or Shotgun (14). A tree relating the sequences is then used to choose a candidate divergent set. To ensure that the set is valid, all pairs of sequences in the candidate set are compared, and non-divergent sequences are discarded. The resulting divergent set can then be recovered either as database identifiers or as sequences and used for many different tasks including motif finding, phylogeny, and covariation analysis. Alternatively the set can be used for another round of refinement, or another divergent set can be chosen from the same initial set of sequences. For each step, we tried several different approaches. Here we compare the performance of these different approaches and describe the best practice methods available in the DivergentSet interface.

**EXPERIMENTAL PROCEDURES**

**Measuring Distances between Sequences**—We tested three different methods to measure distances between sequences. These methods were: (a) Needleman-Wunsch global alignments on all pairs of sequences (15), (b) Dice coefficients (16) of “k-words” in the sequences of different sizes, and (c) the clustering algorithm implemented in MUSCLE (17). Only the Needleman-Wunsch method provides the exact pairwise sequence identity: the other two approaches were designed to reduce the time spent on pairwise sequence comparisons.

Dice coefficients (16) are used to compare strings without aligning them. The comparisons are instead performed by examining the fraction of shorter strings that the two strings have in common. The Dice coefficient between two strings A and B is defined as \(2|A \cap B|/|A \cup B|\) (Fig. 2), where \(A\) and \(B\) are the sets of unique substrings of length \(k\) contained in the strings \(A\) and \(B\), respectively. Dice coefficients provide a very fast method of comparing strings because they do not require that the sequences be aligned. The algorithm thus takes time proportional to sum of the lengths of the two strings rather than the product of the lengths.

MUSCLE (17) also contains an approximate clustering and tree-building method that does not require alignment (it is used to build the initial guide tree for alignment). This method is also substantially faster than building a tree using distances from all pairwise alignments.

We tested the approximate methods (Dice coefficients and MUSCLE) by comparing the tree distances from these techniques with the Needleman-Wunsch distances. Tree distances were the leaf-to-leaf distances between sequences using the UPGMA tree (18) calculated from each kind of distance matrix or, for MUSCLE, either the UPGMA tree or the neighbor-joining tree (19).

**Gathering Additional Sequences**—We tested two methods for gathering additional sequences. First, we performed one round of the Shotgun algorithm (14), which consolidates the results of many BLAST searches by keeping all sequences that appear in a minimum number of BLAST results from sequences in the input set. We used a BLAST E-value threshold of \(10^{-10}\) and a minimum Shotgun score of 1 or 3. These parameters can be easily modified in the user interface. Second, we performed PSI-BLAST (13), a method that increases the sensitivity of BLAST searches by iteratively building up a profile of conserved regions on each sequence for three iterations using an inclusion threshold of \(10^{-20}\). We tested these methods on alignments of real sequences and on simulated data consisting of several disjoint protein families that evolved separately.

**Using a Phylogenetic Tree to Reduce the Number of Comparisons**—To minimize the number of comparisons, we tried several methods that examine the sequences in an order defined by a phylogenetic tree. The tree allows us to eliminate some sequences from the analysis early, saving the time that would otherwise be required to compare them with all other sequences. Drawing the tree with the root (the ancestor) at the left and the leaves (the modern sequences) at the right, the children of each node are placed in an arbitrary order from top to bottom. The tree order can also conveniently produce different, randomly sampled divergent sets.

The first method assumes that the phylogenetic tree provides a complete picture of sequence evolution and that the rates of evolution are constant in every lineage. In this “trust tree” method, sequences are compared in tree order (Fig. 3). Starting with the first sequence \(i\), we compare each successive sequence \(j\) until a sequence more divergent than the percent identity threshold is encountered. We measure divergence in terms of “percent identity,” which is the number of bases or residues that are the same in the optimal alignment of both sequences divided by the length of the shorter sequence. All sequences that appear later in the tree order \((j + 1, j + 2, \ldots, j + n)\) are assumed to be at least as divergent as \(i\) from \(j\) and are not examined.

This trust tree method reduces the number of comparisons greatly. However, it also introduces many false positives by excluding sequences from families of which they are members primarily because of differences in evolutionary rate between sequences. False positives are introduced when sequences that appear later in the tree order after the first divergent sequence in the tree are actually more similar to the sequence currently being evaluated than is the divergent sequence. These comparisons can occur when the tree is inaccurate or when the rates of evolution are unequal in different parts of the tree. False negatives, or sequences that are supposed to be in the set but were inadvertently dropped, are not possible with this approach or with any of the approaches we used. False negatives are impossible.
In other words, groups of sequences whose common ancestor is to
tility than the threshold (successive sequence. Each sequence that has a higher percent iden-
ty threshold (divergent sequence is found (method is used (shown here), the comparison stops as soon as a
divergent set. It provides large time savings when few sequences are in the
divergent set but more modest savings when the percent identity threshold is low, and therefore most sequences must be compared with
most other sequences. One disadvantage of this method is that it is
difficult to parallelize because the comparisons performed during
each pass through the tree determine which other comparisons need
to be performed and thus cannot be known at the start of the
procedure. This method is theoretically the most efficient of the
methods we tested when running on a single CPU without
parallelization.

Finally we used a method that we call "candidate pairs." In this
method, we use the trust tree approach to pick a candidate divergent
set and then perform all pairwise comparisons within this reduced set
(not all pairs in this candidate set will be divergent because the trust
tree method is approximate). This method also produces no false
positives and reduces the number of comparisons considerably. It
also has the advantage that it was easily parallelized (as described
under "Implementation" below). This natural parallelism arises be-
cause all pairwise comparisons in the reduced set must be per-
formed, and thus blocks of comparisons can be distributed among
multiple CPUs without synchronization. This method is theoretically
the most efficient of the methods we tested when running in parallel
on a cluster if considering elapsed time for the user rather than total
CPU time.

We chose to implement these methods using a phylogenetic tree
rather than a distance matrix for two reasons. First, the tree clusters
together the sequences that are most closely related to one another. Thus,
when comparing each sequence with the sequences that are
already in the divergent set, similarities that would cause the new
sequence to be rejected are encountered earlier in the process,
reducing the total number of comparisons. Second, we found during
user testing that displaying the divergent set in the context of a
phylogenetic tree makes the data much easier to interpret than dis-
playing the full distance matrix especially when the number of se-
quencies is large (a tree with 1000 taxa is much easier for most
researchers to interpret than the equivalent distance matrix with
1,000,000 elements). In practice, constructing the tree takes a rela-
tively small fraction of the total run time (about 1%), so the benefits of
using it outweigh the cost of its construction.

Maximal Versus Random Divergent Sets—One important issue
when choosing divergent sets is whether the application requires a
random divergent set or a maximal divergent set. A random divergent
set ensures that the sample of sequences included in the divergent
set is unbiased, which can be important for phylogenetic applications.
A maximal divergent set contains as many sequences from the original
set as possible, which can be important when the amount of data
is limiting.

We produce both random and maximal divergent sets using a
tree-based approach. To generate a random divergent set, we ran-
randomize the order of the branches at each level of the tree using the
Mersenne Twister random number generator (20) as implemented in
the Python standard library. Then for each sequence starting with the

\[
\begin{align*}
(a) & : \text{In other words, groups of sequences whose } \text{common ancestor is} \\
(b) & : \text{to} \\
(c) & : \text{than the threshold (successive sequence. Each sequence that has a higher percent identity threshold (divergent sequence is found (method is used (shown here), the comparison stops as soon as a} \\
\text{divergent set. It provides large time savings when few sequences are in the} \\
\text{divergent set but more modest savings when the percent identity threshold is low, and therefore most sequences must be compared with} \\
\text{most other sequences. One disadvantage of this method is that it is} \\
\text{difficult to parallelize because the comparisons performed during} \\
\text{each pass through the tree determine which other comparisons need} \\
\text{to be performed and thus cannot be known at the start of the} \\
\text{procedure. This method is theoretically the most efficient of the} \\
\text{methods we tested when running on a single CPU without} \\
\text{parallelization. Finally we used a method that we call "candidate pairs." In this} \\
\text{method, we use the trust tree approach to pick a candidate divergent} \\
\text{set and then perform all pairwise comparisons within this reduced set} \\
\text{(not all pairs in this candidate set will be divergent because the trust} \\
\text{tree method is approximate). This method also produces no false} \\
\text{positives and reduces the number of comparisons considerably. It} \\
\text{also has the advantage that it was easily parallelized (as described} \\
\text{under "Implementation" below). This natural parallelism arises be-} \\
\text{cause all pairwise comparisons in the reduced set must be per-} \\
\text{formed, and thus blocks of comparisons can be distributed among} \\
\text{multiple CPUs without synchronization. This method is theoretically} \\
\text{the most efficient of the methods we tested when running in parallel} \\
\text{on a cluster if considering elapsed time for the user rather than total} \\
\text{CPU time. We chose to implement these methods using a phylogenetic tree} \\
\text{rather than a distance matrix for two reasons. First, the tree clusters} \\
\text{together the sequences that are most closely related to one another. Thus, when comparing each sequence with the sequences that are} \\
\text{already in the divergent set, similarities that would cause the new} \\
\text{sequence to be rejected are encountered earlier in the process,} \\
\text{reducing the total number of comparisons. Second, we found during} \\
\text{user testing that displaying the divergent set in the context of a} \\
\text{phylogenetic tree makes the data much easier to interpret than dis-} \\
\text{playing the full distance matrix especially when the number of se-} \\
\text{quences is large (a tree with 1000 taxa is much easier for most} \\
\text{researchers to interpret than the equivalent distance matrix with} \\
\text{1,000,000 elements). In practice, constructing the tree takes a rela-} \\
\text{tively small fraction of the total run time (about 1%), so the benefits of} \\
\text{using it outweigh the cost of its construction.} \\
\end{align*}
\]
Picking Non-redundant Sequences

first sequence (i.e. the sequence found by following the first child in each successive node from the root), we compare the current sequence with each other sequence that is below it in the tree, ignoring already deleted sequences. Any sequence that is too similar to the current sequence is deleted. A new, random set can be produced by randomizing the branch order. The branch order determines which of each set of topologically equivalent sequences is encountered first and therefore which sequence will be kept when there is an equal choice between groups of sequences.

To generate maximal sets of sequences, we build a fully connected graph (each sequence is connected to each other sequence). In this graph, the vertices represent the sequences, and the edges represent the pairwise percent identity scores. We iteratively remove the vertices with greatest number of edges that are above the percent identity threshold until all edges are below the percent identity threshold. This process yields a maximal set of divergent sequences.

Keeping Specific Sequences in the Divergent Set—It is often important to keep specific sequences in the divergent set. Sequences may be preferred because their crystal structures are known or because the organism in which they are found is genetically tractable. Because the order in which each sequence appears in the tree determines whether it will be kept in preference to other sequences in the same subtree, we can conveniently "promote" user-specified sequences to be preferentially kept by moving each of these sequences to the first position in its parent's list of children (Fig. 3, topmost in the tree shown). We mark sequences as "preferred" by checking whether they are in a user-supplied list of identifiers to keep or whether they match specified criteria for preferential inclusion. These criteria include the presence of associated crystal structures (Protein Data Bank records). Randomization can be achieved by permuting the preferred and non-preferred children of each node separately such that all the preferred children of a node always appear before all the non-preferred children. Inclusion in the preferred list is propagated from the leaves to the root such that any node that has a preferred descendant is itself preferred. This method allows random sampling from the distribution of divergent sets that include as many of the desired sequences as possible (if a pair of preferred sequences are similar to one another, only one sequence from that pair will be kept in the divergent set).

Data Sources—To test the performance of the different algorithms, we used two test data sets: one downloaded from KEGG (21) on August 25, 2005 and one simulated. We chose these sets to represent a variety of possible applications.

The KEGG set, "LysRS," consists of the 497 non-redundant sequences that match a text search in KEGG for the core translational enzyme "lysine tRNA synthetase" consisting of both class I and class II LysRS sequences (the two classes are not homologous to one another and represent two independent origins of the same biochemical activity (22)).

The simulated set, "sim," consists of 1000 sequences families, each with 1–50 sequences evolved using a random breakpoint phylogeny, also known as the equiprobable model (23). Each sequence family started from a single ancestral sequence with equal amino acid frequencies. Sister taxa could evolve a total sequence divergence in the range 1–50%. This set was designed to test false positive rates and false negative rates for inclusion in the divergent sets. False positives are sequences that are included in the set inappropriately because they are too similar to one of the other sequences in the set; false negatives are sequences that were inappropriately excluded from the divergent set. Measuring false positive and false negative rates requires simulated data because the true protein families for the biological sequences in GenBank™ are not precisely known. This uncertainty makes it difficult to tell whether BLAST hits to "hypothetical proteins" are finding new family members or are hitting arbitrary sequences by chance. This sequence set is ~1% of the current size of GenBank™ and models the situation of finding genes in a discrete taxonomic group (e.g. the vertebrates).

Implementation—Most of the source code described here was written in Python 2.4 (Python Software Foundation) and tested on MacOSX and on a Linux Beowulf cluster (Beowulf Project). The exceptions are the Needleman-Wunsch algorithm, which we implemented in C for performance, and the BLAST, PSI-BLAST, MUSCLE, and MEME programs for which we used the published implementations. The web interface uses Apache (Apache Software Foundation) and mod_python. Parallelization is achieved via PBS/TORQUE (Cluster Resources Inc.) running on our Beowulf cluster (bmf.colorado.edu/divergentset). The following is a high level overview of the algorithm used by DivergentSet to pick divergent sets of sequences.

Take as user input a list of GIs, accessions, KEGG Ortholog (KO) identifiers, KEGG Enzyme Commission (EC) numbers, or FASTA sequences. Retrieve sequences from the selected sequence database using the National Center for Biotechnology Information fastacmd program.

Optionally search for additional related sequences. For each input sequence, generate a BLAST or PSI-BLAST job and submit it to the PBS server to run in parallel on a Beowulf cluster.

While the BLAST or PSI-BLAST jobs are still running, update the status of running jobs and amount of time jobs have been running on the cluster.

When the BLAST or PSI-BLAST jobs have finished running, collect the results using the Shotgun algorithm (14). Create new jobs to build a MUSCLE tree with the resulting sequences and submit to the PBS server to run in parallel.

Once the MUSCLE tree has been built, pick a set of candidate divergent sequences using the tree as a guide (see “Experimental Procedures”).

While the MUSCLE jobs are still running, update the status of running jobs and amount of time jobs have been running on the cluster.

Pick set of candidate divergent sequences from the MUSCLE tree.

Verify all pairs of sequences in the candidate set. During this step, build an upper triangular matrix of all pairwise comparisons and split up the number of comparisons such that no processor gets more than some maximum number of pairs of sequences to analyze. Submit these jobs to the PBS server and run in parallel on the cluster.

While all pairs of candidate sequences are being analyzed, update the status of running jobs and amount of time jobs have been running on the cluster.

Once all pairs of candidate sequences have been analyzed, pick the final set of divergent sequences using either the maximal set or tree order method (see “Experimental Procedures”).

Finally generate a text representation of the final phylogenetic tree. Highlight in gray original (input) sequences, and highlight in yellow all preferentially kept sequences (e.g. those with crystal structures available). If the user has requested a keyword search, indicate which sequences have these keywords. Select all divergent sequences. Cache the results of this search in the session to allow the user to rapidly refine the results of the search by manually adding or deleting sequences from the divergent set or by changing the BLAST thresholds.
This task-farming approach allowed us to obtain nearly linear speed-up for the BLAST and pairwise comparison steps, greatly reducing the overall analysis time.

RESULTS

Speed and Accuracy of Approximate Methods for Measuring Distance between Sequences—A significant amount of time is spent computing distance matrices from all pairs of sequence alignments. This step has a time complexity of $O(N^2)$ both in the number of sequences and in the length of the sequences. The amount of time spent computing the distance matrices becomes prohibitive for the large numbers of homologous sequences that are now available in public databases. We explored two methods of reducing the time complexity: reducing the time spent performing each comparison and reducing the total number of comparisons.

Speed of Distance Calculation Methods—Fig. 4 shows the performance characteristics of the different distance methods. All these comparisons use the distances computed to build UPGMA trees. We initially prototyped the Dice coefficients and Needleman-Wunsch distance comparisons in Python. Although the performance of the Python implementation of Dice coefficients was acceptable, we were able to increase the performance of the Needleman-Wunsch algorithm by a factor of almost 10,000 by recoding it in C using arrays of small integers rather than character strings. Results shown are for the C implementation. Using MUSCLE and using Dice coefficients were both much faster than performing all-pairs Needleman-Wunsch alignments even using our optimized C implementation. MUSCLE uses a heuristic method to cut down the number of comparisons.

As expected, the time increased roughly in proportion to the square of the number of sequences: for Needleman-Wunsch, this time ranged from 0.3 s to align 10 sequences to 920 s to
Comparison between different methods for building the divergent set, in terms of speed and accuracy, reveals large differences in performance between algorithms. All data shown use the LysRS set as input. The x axis always shows the divergence threshold in terms of percent identity. “tree dice 5” and “tree muscle 5” refer to iterating the trust tree procedure five times to rerandomize the
align about 500 sequences, whereas for Dice and MUSCLE the times with 500 sequences were 13 and 3.3 s, respectively. Thus, using approximate methods such as Dice coefficients to reduce the cost of the comparisons over Needleman-Wunsch can provide large time savings, but the heuristic methods used by MUSCLE perform even better. The results were similar for both real and simulated sequences, suggesting that the highly conserved regions in the biological sequences do not greatly contribute to the result.

**Speed and Accuracy of Set Picking Methods**—Fig. 5 shows time and accuracy characteristics of the different set picking methods we used. All points in this figure are averages of 20 independent runs in which a different random divergent set was chosen. The total time (Fig. 5a) is, as expected, constant for all pairs using either Dice or Needleman-Wunsch because all sequences must be compared. The good pairs method can be almost 2 orders of magnitude faster than all pairs when few sequences are more divergent than the threshold but quickly slows down to within a factor of 4 of all pairs when the threshold is high and more sequences are in the divergent set and must be compared with one another. The Dice coefficient method was about 2 orders of magnitude faster than even our optimized Needleman-Wunsch implementation.

The number of comparisons (Fig. 5b, note log scale) was greatly reduced by using the phylogenetic tree as a guide. The series on this graph are paired because a given set picking method requires the same number of comparisons whether sequences are compared using the Dice or Needleman-Wunsch method. Trusting the tree reduced the number of comparisons by 2 orders of magnitude but, unfortunately, increased the false positive rate to unacceptable levels. Relative to all pairs, the good pairs method reduced the number of comparisons required by more than an order of magnitude (3912 comparisons instead of 123,256 at 50% divergence, resulting in a reduction of run time from 820 to 20 s), thus resulting in a substantial savings. Note that in Fig. 5b for the good pairs methods Needleman-Wunsch was always used to evaluate the methods in the tree (dice versus nw refers to the method used to build the tree). Because building the tree takes little of the overall time, these two methods perform in a very similar fashion.

The number of sequences found (Fig. 5c, note log scale) is similar using each method except that Dice sometimes returns either too few (when iterated) or too many (when all-pairs comparisons are performed) sequences depending on the threshold and on whether all pairs are compared or the tree is used. This poor performance of the Dice method is due to the nonlinear relationship between the Dice coefficients and the percent identity calculated from pairwise alignments.

![Fig. 6. Nonlinear relationships between pairwise distances between sequences measured by Needleman-Wunsch (x axis) and Dice coefficients (y axis) for word sizes of 2 (black points) and 3 (gray points). The dashed line shows x = y. The scatter in the Dice coefficients leads to the occasional inclusion of very similar sequences that have very dissimilar Dice coefficients. Averaging the distances for k = 2 and k = 3 did not provide a solution to this problem in part because using this technique to smooth out the nonlinear relation also increased the variance. Results shown are for 1000 randomly chosen sequence pairs from the LysRS set.](image)

The worst false positive for each method (Fig. 5d), which measures the maximum sequence identity in the set returned, is alarmingly high for all methods except using the good pairs and “all pairs nw” methods (the results for candidate pairs, not shown, are identical to those for good pairs). In fact, all other
Fig. 7. Comparison of different methods for finding additional sequences using the simulated data set (a–d) and the LysRS set, including both class I and class II LysRS (e and f). The color denotes the search algorithm (black, PSI-BLAST; red, Shotgun with a threshold of 1; blue, Shotgun with a threshold of 3); the symbol denotes the data set (for sim, diamonds indicate sequences sampled from all families, and crosses indicate sequences sampled from a single family). The x axis always shows the number of sequences used as a seed for the search. The y axis shows the number of sequences returned in the divergent set (a), the fraction of sequences in the relevant family or families that were detected (b), the false discovery rate defined as the fraction of sequences returned in the set that were false positives (c), the amount of CPU time required (d), the number of known LysRS hits that were missed in the analysis (e), and the number of matches to known LysRS sequences for the LysRS set (f). Each point is the average of 20 runs on independent sets of starting sequences. Error bars show one standard deviation. All BLAST thresholds were set at $10^{-3}$. 
methods invariably return some sequences that are far above 
threshold, in some cases exceeding the desired similarity by 
almost 50%, which would be devastating for motif-finding 
algorithms.

The false positive rate (Fig. 5e) shows that using Dice co-

efficients, although fast, is not an accurate method, whereas 
using the good pairs method does not ever produce false 
positives. The false positive rate (Fig. 5f) shows that neither 
"all pairs" nor good pairs ever produces false positives, a 
highly desirable property. Even with five rounds of iteration 
to randomize the tree order, all of the methods that trust the 
tree and all pairs using Dice coefficients produce alarming high 
false positive rates (often exceeding 50%). Overall although 
Dice coefficients and trusting the tree can be extremely fast, 
the good pairs is accurate and is substantially faster than 
calculating the Needleman-Wunsch alignment scores for all 
pairs of sequences.

Speed and Accuracy of Methods for Finding Additional 
Sequences—We tested the accuracy of the methods for find-
ing additional sequences (PSI-BLAST and Shotgun) on both 
the LysRS and the simulated data. For the simulated set 
(Fig. 7, a–d), we were able to measure false negative and false 
discovery rates because we knew the membership of each 
sequence in each family in advance. For the real sequences 
(Fig. 7, e and f), we could only measure the number of known 
LysRS sequences recovered and missed. For the simulated 
data, at a given BLAST threshold (data shown are for an 
E-value of $10^{-3}$), PSI-BLAST was able to recover a much 
larger fraction of the sequences (Fig. 7, a and b), although it 
typically took 5–6 times longer to run (Fig. 7d). For example, 
PSI-BLAST took 242 s to run on a set of 30 seed sequences 
as opposed to 65 s for BLAST.

Because most of the time was spent performing the BLAST 
searches, Shotgun took about the same time to run regard-
less of similarity threshold. Mixing seed sequences from mul-
tiple families greatly decreased the performance of all algo-

rithms (Fig. 7b). For example, Shotgun with a threshold of 1 
could recover over 90% of the hits when at least 20 se-
quencies from the same family were used as seeds but only 
10% of the hits when the same number of sequences were 
chosen from different families. PSI-BLAST was somewhat 
less adversely affected by using sequences from multiple 
families. The false discovery rate (the fraction of sequences in 
the result set that were false positives) was below 1%, and 
therefore negligible, for almost all methods. The one method 
with a high false positive rate was Shotgun when used with a 
threshold of 1. The false positive rate for this method reached 
8% under some conditions (Fig. 7d). On the LysRS set, PSI-
BLAST returned many more sequences and missed far fewer 
sequences than did Shotgun at either threshold, finding al-
most all sequences when at least 20 sequences were used as 
input (Fig. 7e) and returning on average 2–4-fold more se-
quencies (Fig. 7f).

The poor performance of Shotgun on these data is probably 
due to the stringent E-value threshold of $10^{-3}$. The strength of 
Shotgun is that it can find matches by combining multiple 
weak hits (14). With the E-value we used, the hits were indi-

cividually significant, and relatively few sequences matched 
more than one of the initial query sequences. Running Shot-
gun with lower E-values, e.g. 1, can result in much better 
sensitivity with little loss of specificity (data not shown). We 
kept the E-value the same for all analyses in the results shown 
so that the techniques could be compared directly when used 
with exactly the same BLAST parameter settings.

Effect of Set Choice on Motifs Identified by MEME—Finally 
we tested the effect of the set choice on the stability of the 

motifs recovered by MEME (1). For this analysis, we used the 
default MEME settings except that the maximum number of 

toys was raised to 20, and a significance threshold was set 
at 0.01 to minimize noise from non-significant motifs (MEME 
will otherwise return as many motifs as are needed to reach 
the number specified).

From the LysRS set, we chose random divergent sets of 
sequences at divergence thresholds ranging from 30 to 90% 
identity (Fig. 8). We found that the motifs that MEME identified 
from the same set of initial sequences varied considerably 
depending on which divergent set of sequences we used. The 
average length of the motifs recovered was 31 amino acids.

The results shown in Fig. 8 use a range of similarity thresh-
holds to decide whether each of the pairs of motifs counts as 
the "same" motif in two independent runs of MEME using 
different, randomly chosen divergent sets. We aligned each 
motif with each other motif using the Needleman-Wunsch 
algorithm. We then chose the best match for each motif as the 
motif in the other run that had the highest alignment score 
when aligned with that motif. Results are shown for perfect 
matches (zero mismatches) and for two, five, and 10 
mismatches.

For very similar motifs with no more than two mismatches, 
the average recovery of motifs (as defined by the 

fraction of pairs of runs in which the same motif was recov-
ered in both runs) never exceeded 50%. For exact matches, 
this recovery fell below 10% for exact matches for interme-
diate divergences (50–60% sequence identity). The recover-
y was substantially higher when the criterion for similarity 
was relaxed, averaging 50–60% when five mismatches or 
gaps were allowed and reaching 90–100% when 10 mis-
matches or gaps were allowed. However, at these levels, 
the motifs differ substantially, and the core motifs that 
match between the samples would typically not have been 
significant by themselves.

Interestingly the approximate match using the reduced al-
phabet made very little difference to the results. In the re-
duced alphabet, similar amino acids (e.g. isoleucine, valine, 
and leucine) are coded as the same character, resulting in five 
groups of amino acids. On average, using the reduced alpha-
bet led to less than a 10% increase in recovery (Fig. 8, 
compare a and b), suggesting that the differences are not
primarily due to interchange between similar amino acids. Fig. 9 gives examples of some of the motif pairs with different numbers of changes on the normal and reduced alphabets. This lack of consistency underscores the importance of using many different divergent sets for motif finding and for using only those motifs and parts of motifs that are reliably recovered. We repeated the analysis on another data set, isoleucyl-tRNA synthetase, in which all sequences were homologous rather than falling into two classes, but the results were essentially identical (data not shown).

Additional Analyses—In addition to the analyses described here and above under “Experimental Procedures,” we tried several variations on the distance metric and the tree-building algorithm. We tested Dice coefficients based on substrings of lengths 2, 3, and 4 and the average of the Dice coefficients for substrings of lengths 2 and 3 (results similar to those shown as “Dice”, which was calculated with words of length 3). We also tried an alternative tree-based comparison algorithm that compared each of the sequences to the sequences in the “good” set rather than deleting “bad” sequences from the tree (results very similar to those shown as good pairs). Finally we tried alternative methods of building the tree with results very similar to those shown in all cases; we tried UPGMA clustering of the Dice coefficients and pairwise Needleman-Wunsch.
alignment scores, but our implementations were typically slower than the MUSCLE output, and no improvement in accuracy was obtained (data not shown).

**DISCUSSION**

We found large differences in both speed and accuracy in the methods we tested for finding additional sequences, measuring distances between sequences, and choosing the divergent set. We made the best of these methods (Shotgun and PSI-BLAST, Needleman-Wunsch alignments, and candidate pairs, respectively) available in the DivergentSet web interface. Candidate pairs provides results identical to the good pairs method but is easier to parallelize.

Dice coefficients, although very fast to calculate, are unsuitable for choosing divergent sets. Dice coefficients did not map accurately onto the overall percent identity between aligned pairs of sequences. Consequently a given threshold in terms of Dice coefficients both excluded pairs of sequences less similar than the threshold and included more similar than the threshold. In particular, sequences that are very similar overall often differ radically in their Dice coefficients for $k > 2$, leading to the inclusion of nearly identical sequences in the final result. Because the main point of choosing divergent sets is to eliminate such sequences, we concluded that they were unsuitable for this application. Although it would be possible to fit a function to relate Dice coefficients and pairwise distances for a random sample of pairs of sequences from a given input set, the statistical error in the relationship is likely to be so large that the false positive and false negative rates remain unacceptably high.

The ability to find additional sequences is greatly improved when all the sequences come from the same sequence family. Both PSI-BLAST and Shotgun performed poorly in cases where many unrelated sequences are used as seeds. DivergentSet itself could be used to assist in such analyses by screening a set of input sequences for those that are significantly different, rather than those that are significantly similar, to identify the families of sequences that are similar enough to be used as seeds for subsequent searches. When the number of input sequences was large, Shotgun found as many sequences as PSI-BLAST in a much shorter time. The Shotgun results can also be improved by lowering the $E$-value threshold, which imposes essentially no time penalty but may affect specificity. We kept the $E$-value threshold across methods in this analysis to permit a direct comparison using the same BLAST parameters, but we should note that the strength of Shotgun is in using very low $E$-values, e.g. 1, to find very distant homologies using a combination of hits that are individually weaker than threshold (14).

The particular divergent set chosen can greatly influence the motifs found. Under some conditions, less than 10% of the motifs were the same between two different randomly chosen divergent sets. These results underscore the importance of using many different divergent sets from the same sequences for analyses such as motif finding. Such a striking degree of inconsistency calls into question the biological significance of motifs that are identified from a single divergent set of proteins, although it also suggests that the false positive rate of motif-finding algorithms may be greatly reduced by using multiple divergent sets chosen from the same larger set of sequences. This need to repeat the analysis using different divergent sets underscores the need for an efficient method for set choice: if each divergent set requires days of manual effort to put together, confirming motif analyses by using many different sets is infeasible.

We achieved a dramatic speed improvement both over choosing the divergent set manually and over the simple algorithm of computing the full distance matrix and choosing sequences below the percent identity threshold. The process of choosing divergent sets by hand has traditionally been extremely tedious, taking days for an experienced researcher, which, along with the inability to randomize the sets, has discouraged repetition of analyses to check for stability. Using the full distance matrix and Needleman-Wunsch alignments automated this process to take a few hours of computer time (on a set of ~1000 sequences) without manual intervention.
By taking advantage of a phylogenetetic tree built from the data and using less time-consuming comparison methods, we were able to further reduce this time to a few minutes. By parallelizing the slow steps (the pairwise comparisons and BLAST searches), we were able to further reduce the time to a minute or two from the user’s perspective.

The ability to sample random divergent sets coupled with the ability to conveniently retrieve additional sequences from the database in minutes rather than hours will greatly improve our ability to test the robustness of a wide range of bioinformatics analyses to the choice of divergent set. DivergentSet will also facilitate the development of improved methods for drawing reliable conclusions from many of these analyses.

Acknowledgments—We thank Shelley Copley and Michael Yarus for providing input into the feature set and writing and members of the Copley laboratory for beta testing. We also thank Sandra Smit, Catherine Lozupone, and other members of the Knight laboratory for comments on the manuscript.

* This work was supported in part by a seed grant from the W. M. Keck RNA Bioinformatics Initiative and by Air Force Office of Scientific Research Proposal 0705.01.1102b. The costs of publication of this article were defrayed in part by the payment of page charges. This article must therefore be hereby marked “advertisement” in accordance with 18 U.S.C. Section 1734 solely to indicate this fact.

§ Both authors contributed equally to this work and should be considered joint first authors.

To whom correspondence should be addressed. Tel.: 303-492-1984; Fax: 303-492-7744; E-mail: rob@spot.colorado.edu.

REFERENCES

1. Bailey, T. L., and Elkan, C. (1994) in ISMB-94: Proceedings, Second International Conference on Intelligent Systems for Molecular Biology (Altman, R., Brutlag, D., Karp, P., and Searls, D., eds) pp. 28–36, AAAI Press, Menlo Park, CA

2. Lockless, S., and Ranganathan, R. (1999) Evolutionarily conserved pathways of energetic connectivity in protein families. Science 286, 299–299

3. Copley, S., and Dhillion, J. (2002) Lateral gene transfer and parallel evolution in the history of glutathione biosynthesis genes. Genome Biol. 3, Research0025

4. Copley, S., Novak, W., and Babbitt, P. (2004) Divergence of function in the thioredoxin fold superfamly: evidence for evolution of peroxiredoxins from a thioredoxin-like ancestor. Biochemistry 43, 13981–13995

5. Tamura, K., Nei, M., and Kumar, S. (2004) Prospects for inferring very large phylogenies by using the neighbor-joining method. Proc. Natl. Acad. Sci. U. S. A. 101, 11030–11035

6. Poe, S. (1998) Sensitivity of phylogeny estimation to taxonomic sampling. Syst. Biol. 47, 18–31

7. Rosenberg, M., and Kumar, S. (2001) Incomplete taxon sampling is not a problem for phylogenetic inference. Proc. Natl. Acad. Sci. U. S. A. 98, 10751–10756

8. Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990) Basic local alignment search tool. J. Mol. Biol. 215, 403–410

9. Holm, L., and Sander, C. (1998) Removing near-neighbour redundancy from large protein sequence collections. Bioinformatics 14, 423–429

10. Schloss, P., and Handelsman, J. (2005) Introducing DOTUR, a computer program for defining operational taxonomic units and estimating species richness. Appl. Environ. Microbiol. 71, 1501–1508

11. Seguritan, V., and Rowher, F. (2001) Fastgroup: a program to dereplicate libraries of 16S rDNA sequences. BMC Bioinformatics 2, 9

12. Bernstein, F., Koetzle, T., Williams, G., Meyer, E. F., J., Brice, M., Rodgers, J., Kennard, O., Shimanouchi, T., and Tasumi, M. (1977) The Protein Data Bank: a computer-based archival file for macromolecular structures. J. Mol. Biol. 112, 535–542

13. Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402

14. Pegg, S., and Babbitt, P. (1999) Shotgun: getting more from sequence similarity searches. Bioinformatics 15, 729–740

15. Needleman, S., and Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453

16. Dice, L. (1945) Measures of the amount of ecologic association between species. Ecology 26, 297–302

17. Edgar, R. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 32, 1792–1797

18. Sneath, P., and Sokal, R. (1973) Numerical Taxonomy, Freeman, San Francisco

19. Saitou, N., and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. 4, 406–425

20. Matsumoto, M., and Nishimura, T. (1998) Mersenne Twister: a 623-dimensional equidistributed uniform pseudorandom number generator. ACM Trans. Model. Comput. Simul. 8, 3–30

21. Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., and Kanehisa, M. (1999) KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Res. 27, 29–34

22. Iba, M., Morgan, S., Curnow, A., Priddore, D., Vothknecht, U., Gardner, W., Lin, W., Woese, C., and Soll, D. (1997) A euryarchaeal lysyl-tRNA synthetase: resemblance to class I synthetases. Science 278, 1119–1122

23. Page, R. (1993) On describing the shape of rooted and unrooted trees. Cladistics 9, 93–99