Clustering Near-Identical Sequences for Fast Homology Search

Michael Cameron[1], Yaniv Bernstein[1], and Hugh E. Williams[2]

[1] School of Computer Science and Information Technology
RMIT University, GPO Box 2476V, Melbourne, Australia, 3001
{mcam, ybernste}@cs.rmit.edu.au

[2] Microsoft Corporation
One Microsoft Way, Redmond, Washington 98052, USA
hughw@microsoft.com

Abstract. We present a new approach to managing redundancy in sequence databanks such as GenBank. We store clusters of near-identical sequences as a representative *union-sequence* and a set of corresponding edits to that sequence. During search, the query is compared to only the union-sequences representing each cluster; cluster members are then only reconstructed and aligned if the union-sequence achieves a sufficiently high score. Using this approach in BLAST results in a 27% reduction is collection size and a corresponding 22% decrease in search time with no significant change in accuracy. We also describe our method for clustering that uses *fingerprinting*, an approach that has been successfully applied to collections of text and web documents in Information Retrieval. Our clustering approach is ten times faster on the GenBank nonredundant protein database than the fastest existing approach, CD-HIT. We have integrated our approach into FSA-BLAST, our new Open Source version of BLAST, available from `http://www.fsa-blast.org/`. As a result, FSA-BLAST is twice as fast as NCBI-BLAST with no significant change in accuracy.

1 Introduction

Comprehensive genomic databases such as the GenBank non-redundant protein database contain a large amount of internal redundancy. Although exact duplicates are removed from the collection, there remain large numbers of near-identical sequences. Such near-duplicate sequences can appear in protein databases for several reasons, including the existence of closely-related homologues or partial sequences, sequences with expression tags, fusion proteins, post translational modifications, and sequencing errors. These minor sequence variations lead to the over-representation in databases of certain protein domains, particularly those that are under intensive research. For example, the GenBank database contains several thousand near-identical protein sequences from the human immunodeficiency virus.

Database redundancy has several pernicious effects. First, a larger database takes longer to query; as sequencing efforts continue to outpace improvements in computer hardware, this is a problem that will continue to worsen. Second, redundancy can lead to highly repetitive search results for any query that matches closely with an over-represented sequence. Third, large-scale redundancy has the effect of skewing the statistics used for determining alignment significance, ultimately leading to decreased search effectiveness. Fourth, the PSI-BLAST algorithm (Altschul et al., 1997) can be misled by redundant matches during iteration, causing it to bias the profile towards over-represented domains; this can result in a less sensitive search or even profile corruption (Li et al., 2002; Park et al., 2000).

Redundancy has been managed in the past by the creation of representative-sequence databases (RSDBs), culled collections in which no two sequences share more than a given level of identity. Such databases have been shown to significantly improve profile training in iterative search tools such as PSI-BLAST by reducing the amount of over-representation of certain protein domains and consequently reducing profile corruption. However, they are less suitable for regular search algorithms such as BLAST (Altschul et al., 1990, 1997) and FASTA (Pearson and Lipman, 1988, 1985) because, by definition, RSDBs are not comprehensive. This leads to search results that are both less accurate—the representative sequence for a cluster may not be the one that aligns best with a given query—and less authoritative because the user is only shown one representative sequence from a family of similar sequences.

In this paper, we describe a sequence clustering methodology that efficiently and effectively identifies and manages redundancy. Importantly, it lacks the drawbacks of previous representative-sequence databases. Previous approaches choose one sequence from each near-duplicate cluster as a representative to the database and delete the other sequences. In contrast, we generate for each cluster a special union-sequence that—through use of wildcard characters—represents all of the sequences in the cluster simultaneously. Through careful choice of wildcards, we are able to achieve near-optimal alignments while still substantially reducing the number of sequences against which queries need to be matched. Further, we store all sequences in a cluster as a set of edits against the union-sequence. This achieves a form of compression and allows us to retrieve cluster members for more precise alignment against a query should the union-sequence achieve a good alignment score. Thus, both space and time are saved with no significant loss in accuracy or sensitivity.

Our method supports two modes of operation: users can choose to see all alignments or only the best alignment from each cluster. In the former mode, the clustering is transparent and the results comparable to searches on an unclustered collection. In the latter mode, the search output is similar to the result of searching a culled representative database, except that our approach is guaranteed to display the best alignment from each cluster and is also able to report the number of similar alignments that have been suppressed.

Our work also improves on previous approaches by reducing the time and resources required to create clusters. The most successful existing algorithms use a form of all-against-all comparison that is quadratic in the number of sequences in the database. Our innovative clustering approach uses a technique known as *fingerprinting* that leads to significantly faster clustering; we are able to process the entire GenBank collection in one hour on a commodity workstation. By contrast, the fastest previously available system, CD-HIT (Li et al., 2001b), takes almost ten hours on the same machine.

To investigate the effectiveness of our clustering approach we have integrated it with our freely available open-source software package, FSA-BLAST. When applied to the GenBank non-redundant (NR) database, our method reduces the size of sequence data in the NR database by 27% and improves search times by 22% with no significant effect on accuracy.

2 Existing approaches

Reducing redundancy in a sequence database is essentially a two-stage process: first, redundancy within the database must be identified by grouping similar sequences into clusters; then, the clusters must be managed in some way. In this section we describe past approaches to these two stages.

The first stage of most clustering algorithms involves identifying pairs of similar sequences. An obvious approach to this is to align each sequence with every other sequence in the collection using a pairwise alignment scheme such as Smith-Waterman local alignment (Smith and Waterman, 1981). This is the approach taken by several existing clustering algorithms, including d2_cluster (Burke et al., 1999), OWL (Bleasby and Wootton, 1990), and KIND (Kallberg and Persson, 1999). However, this approach is impractical for any collection of significant size; each pairwise comparison is computationally intensive and the number of pairs is quadratic in the number of sequences.

Several schemes, including CLEANUP (Grillo et al., 1996), NRDB90 (Holm and Sander, 1998), RSDB (Park et al., 2000), CD-HI (Li et al., 2001a) and CD-HIT (Li et al., 2001b), use fast clustering approaches based on greedy incremental algorithms. In general, each proceeds as follows. To begin, the collection sequences are sorted by decreasing order of length. Then, each sequence is extracted in turn and used as a query to search an initially-empty representative database for high-scoring matches. If a similar sequence is found, the query sequence is discarded; otherwise, it is added to the database as the representative of a new cluster. When the algorithm terminates, the database consists of the representative (longest) sequence of each cluster. This greedy approach reduces the number of pairwise comparisons but has three drawbacks: first, a match is only identified when one sequence is a substring of another; second, cases where the prefix of one sequence matches the suffix of another are neglected; and, third, clusters form around longer sequences instead of natural centroids, potentially leading to a suboptimal set of clusters.

Existing greedy incremental algorithms also use a range of BLAST-like heuristics to quickly identify high-scoring pairwise matches. The CLEANUP algorithm (Grillo et al., 1996) builds a rich inverted index of short substrings or *words* in the collection and uses this structure to score similarity between sequence pairs. NRDB90 (Holm and Sander, 1998) and RSDB (Park et al., 2000) use in-memory hashtables of decapeptides and pentapeptides for fast identification of possible high-scoring sequence pairs before proceeding with an alignment. CD-HI (Li et al., 2001a) and CD-HIT (Li et al., 2001b) use lookup arrays of very short subsequences to more efficiently identify similar sequences. However, despite each scheme having fast methods for comparing sequence pairs, the algorithms still operate on a pairwise basis and remain $O(n^2)$ in the size of the database. Indeed, we show in Section 7 that CD-HIT — the fastest of the greedy incremental algorithms mentioned and the most successful existing approach — scales poorly, with superlinear complexity in the size of the collection.

One way to avoid an all-against-all comparison is to pre-process the collection using an index that can efficiently identify high-scoring candidate pairs. Malde et al. (2003) and Gracy and Argos (1998) investigated the use of suffix structures such as suffix trees (Gusfield, 1997) and suffix arrays (Manber and Myers, 1993) to identify groupings of similar sequences in linear time. However, traditional suffix structures consume large amounts of memory and are not suitable for processing large sequence collections such as GenBank on desktop workstations. Malde et al. (2003) report results for only a few thousand EST sequences. The algorithm described by Gracy and Argos (1998) requires several days to process a collection of around 60,000 sequences. External suffix structures, which record information on disk, are also unsuitable; they use a large amount of disk space, are extremely slow for searching, or have slow construction times (Cheung et al., 2005). Nonetheless, we believe that investigating data structures for identifying all pairs of similar sequences in a fixed number of passes is the correct approach.

Once a set of clusters have been identified, most existing approaches retain a single representative sequence from each cluster and delete the rest (Holm and Sander, 1998; Park et al., 2000; Li et al., 2001a,b). The result is a representative database with fewer sequences and less redundancy. However, purging near-duplicate sequences can significantly reduce the quality of results returned by search tools such as BLAST. There is no guarantee that the representative sequence from a cluster is the sequence that best aligns with a given query. Therefore, some queries will fail to return matches against a cluster that contains sequences of interest, which reduces sensitivity. Further, results of a search lack authority because they do not show the best alignment from each cluster. Also, the existence of highly-similar alignments, even if strongly mutually redundant, may be of interest to a researcher.

3 Clustering using wildcards

In this section we describe our approach to representing and searching clusters of highly-similar sequences using union-sequences and special-purpose wildcard characters to represent clusters.

Let us define $E = \{e_1, ..., e_n\}$ as the set of sequences in a collection where each sequence is a string of residues $e_i = r_1...r_n \mid r \in R$. Our approach represents the collection as a set of clusters C, where each cluster contains a union-sequence U and edit information for each member of the cluster. The union-sequence is a string of residues and wildcards $U = u_1...u_n | u_i \in R \cup W$ where $W = \{w_1, ..., w_n \mid w_i \subseteq R\}$ is the set of available wildcards. Each wildcard represents a set of residues and is able to act as a substitute for any of these residues. By convention, w_n is assumed to be the *default wildcard* w_d that can represent any residue; that is, $w_n = R$.

```
KNQVAMN * QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV * QAEVDV * RFRSNT * ER  (union-seq)
        P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEV                  (gi 156103)
KNQVAMN P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEV                  (gi 156105)
          QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV V QAEVDV L RFRSNT K ER  (gi 156121)
KNQVAMN P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV V QAEVDV L RFRSNT K     (gi 552059)
KNQVAMN P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEVDV Q RFRSNT R     (gi 552055)
KNQVAMN P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEVDV Q RFRSNT R E   (gi 552057)
        P QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV V QAEVDV L RFRSNT K ER  (gi 156098)
          QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV V QAEVDV L RFRS         (gi 156100)
              VFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEVDV Q RFRSNT R E   (gi 156111)
        N QNTVFDAKRLIGRKFDEPTVQADMKHWPFKV I QAEVDV Q RFRSNT R     (gi 552056)
```

Fig. 1. Example cluster of heat shock proteins from the GenBank NR database. The union-sequence is shown at the top, followed by the ten member sequences.

Figure 1 shows an example cluster constructed using our approach. The union-sequence is shown at the top and cluster members are aligned below. Columns where the member sequences differ from each another and a wildcard has been inserted are shown in bold face. In this example, $W = \{w_d\}$ — that is, only the default wildcard is used and it is represented by an asterisk.

When a cluster is written to disk, the union-sequence — shown at the top of the figure — is stored in its complete form, and each member of the cluster is recorded using edit information. The edit information for each member sequence includes start and end offsets that specify a range within the union-sequence, and a set of residues that replace the wildcards in that range. For example, the first member of the cluster with GI accession 156103 would be represented by the tuple (8,44,PI); the member sequence can be reconstructed by copying the substring between positions 8 and 44 of the union-sequence and replacing the wildcards at union-sequence positions 8 and 40 with characters P and I respectively. Note that we do not permit gaps; insertions and deletions are heavily penalised during alignment and any scheme that allows gaps in representative sequences is likely to reduce search accuracy. A more complex cluster representation such as a partial-order graph (Lee et al., 2002) could tolerate gaps; while the increased complexity of such a representation leads inevitably to larger on-disk footprint and longer alignment times, the potential increase in cluster size that gapping would allow means that such a technique merits future investigation.

Our clustering method is designed so that each union-sequence aligns to the query with a score that is—with high probability—equal to or higher than the best score for aligning the query to members of the cluster (see Section 5). During search, the query is compared to the union-sequence of each cluster; if the union-sequence produces a statistically significant alignment, then the members of the cluster are restored from their compressed representations and aligned to the query. Our approach supports two modes of operation: users can choose to see all high-scoring alignments, or only the best alignment from each cluster. The latter mode reduces redundancy in the results.

4 Clustering algorithm

In this section we briefly describe our approach to efficiently clustering large sequence collections. A more detailed description of the algorithm is given in Bernstein and Cameron (2006).

In our approach, we use a largely linear-time algorithm that has low main-memory overheads for identifying candidate pairs. Document fingerprinting (Manber, 1994; Heintze, 1996; Brin et al., 1995; Broder et al., 1997; Bernstein and Zobel, 2004) has been used for grouping highly similar documents in extremely large collections and has been successfully applied to text and web data for several applications including plagiarism detection, copyright protection, and search-engine optimisation. Fingerprinting operates by selecting fixed-length subsequences — known as *chunks* — from each document. This set of chunks is known as the document *fingerprint* and acts as a compact surrogate for the document. As highly similar documents are expected to share a large number of chunks, fingerprints are used to efficiently detect similar documents in a collection.

The basic process of fingerprinting can be applied to biological sequence data by substituting sequences for documents, although some alterations in approach are necessary because genomic sequences do not contain natural word-delimiters such as punctuation and whitespace. We have modified our DECO fingerprinting package (Bernstein and Zobel, 2004, 2005) for use with sequence data and it is used as the first stage of our clustering algorithm.

The fingerprinting process identifies chunks that occur in the collection more than once. In the context of sequence data we use subsequences or *words* of length W as our chunks. For each word, DECO outputs a *postings list* of sequences that contain the word and the offset into each sequence where the word occurs. Our clustering algorithm uses these lists to calculate the number of identical words shared by each pair of sequences in the collection. The number of matching words is normalised by the length of the overlapping region between the two sequences; this provides a good quality estimate of the degree of mutual redundancy between the sequences. If this measure exceeds a threshold then the two sequences are aligned using the similarity score measure we describe next. Highly similar candidate pairs with a score below threshold T are then recorded.

Given the list of candidate pairs, we use a variation on single-linkage hierarchical clustering (Johnson, 1967) to identify clusters. Each sequence is initially considered as a cluster with one member. Candidate pairs are processed in increasing order of similarity score, from most- to least-similar, and clusters are merged. To merge a pair of candidate clusters C_X and C_Y with union-sequences X and Y respectively, the overlapping regions of X and Y are aligned. A new union-sequence U is then created by replacing each mismatched residue in the overlap region with a suitable wildcard w. The clusters will only be merged if the mean alignment score increase $\bar{Q}$ in the overlap region is below a specified threshold T — this prevents union-sequences from containing too many wildcards and reducing search performance.

If the clusters are merged, a new cluster C_U is created consisting of all members of C_X and C_Y. When inserting wildcards into the union-sequence, if more than one wildcard is suitable then the one with the lowest expected match score $e(w) = \sum_R s(w,r)p(r)$ is selected, where $p(r)$ is the background probability of residue r (Robinson and Robinson, 1991) and $s(w,r)$ is the alignment score for matching wildcard w to residue r. We discuss how alignment vectors $s(w,\cdot)$ are constructed in Section 5 and how wildcards are chosen in Section 6.

The alignment score increase Q for a wildcard w is calculated as

$$Q(w) = \sum_{R} s(w,r)p(r) - \sum_{R \times R} s(r_1,r_2)p(r_1)p(r_2)$$

where $s(r_1, r_2)$ is the score for matching a pair of residues as defined by a scoring matrix such as BLOSUM62 (Henikoff and Henikoff, 1992). This value estimates the increase in alignment score one can expect against arbitrary query residues by aligning against w instead of against the actual residue at that position.

The above approach has a quadratic complexity in the length of each postings list. While most lists remain quite short even in large databases, a small proportion of words can appear a large number of times. As the database being processed grows, common words can come to dominate overall processing time. For example, a postings lists with 500 entries produces 124,750 potential sequence pairs, which will take a very long time to process. We therefore process frequently occurring words —those with more than M occurrences in the collection, where we use $M = 100$ by default—in a different, top-down manner before proceeding to the standard hierarchical clustering approach described above.

Given a list of sequences l containing a particular frequently occurring word, the top-down approach extracts all sequences in l and selects an exemplar; this is the sequence with the highest percentage identity to the other sequences in the list. The exemplar is then aligned against each sequence in l and used to create a cluster as defined above. All sequences in the new cluster are removed from l and the process is repeated until $|l| < M$. The shortened list is then processed using the hierarchical clustering method. This process still has an $O(n^2)$ worst-case in the length of the list, but is significantly quicker than the hierarchical approach when processing long lists in practice.

5 Scoring wildcards

We have modified BLAST to work with our clustering algorithm. Instead of comparing the query sequence to each member of the database, our approach compares the query only to the union-sequence representing each cluster, where the union-sequence may contain wildcard characters. If a high-scoring alignment between the union-sequence and query is identified, the members of the cluster are reconstructed and aligned to the query. In this section we discuss how, given a set of wildcards W, we determine the scoring vectors $s(w_i, \cdot)$ for each $w_i \in W$.

Ideally, we would like the score between a query sequence Q and a union-sequence U to be precisely the highest score that would result from aligning Q against any of the sequences in cluster C_U. This would result in no loss in sensitivity as well as no false positives. Unfortunately, such a scoring scheme is not likely to be achievable without aligning against each sequence in every cluster, defeating much of the purpose of clustering in the first place.

To maintain the speed of our approach, scoring of wildcards against residues must be on the basis of a standard scoring vector $s(w, \cdot)$ and cannot take into consideration any data about the sequences represented by the cluster. Thus, scoring will involve a compromise between sensitivity (few false negatives) and speed (few false positives). We describe two such compromises below, and finally show how to combine them to achieve a good balance of sensitivity and speed.

During clustering, wildcards are inserted into the union-sequence to denote residue positions where the cluster members differ. Let us define $S = s_1 ... s_x \mid s_i \in W$ as the ordered sequence of x wildcards substituted into union-sequences during clustering. Each occurrence of a wildcard is used to represent a set of residues that appear in its position in the members of the cluster. We define $o \subseteq R$ as the set of residues represented by an occurrence of a wildcard in the collection and $O = o_1 ... o_x \mid o_i \subseteq R$ as the ordered sequence of substituted residue sets. The k^{th} wildcard s_k that is used to represent the set of residues o_k must be chosen such that $o_k \subseteq s_k$.

Our first scoring scheme, s_{exp}, builds the scoring vector by considering the actual occurrence pattern of residues represented by the wildcard in the collection. Formally, we calculate the expected best score s_{exp} as:

$$s_{exp}(w, r) = \frac{\sum\limits_{k \in P_i} \max\limits_{f \in o_k} s(r, f)}{|P_i|}$$

where P_i is the set of ordinal numbers of all substitutions using the wildcard w_i: $P_i = \{j \mid j \ \in \mathbb{N},\ j \leq x,\ s_j = w_i\}$.

This score can be interpreted as the mean score we would get by aligning residue r against the actual residues represented by the wildcard w. This score has the potential to reduce search accuracy; however, it distributes the scores well, and provides an excellent tradeoff between sensitivity and speed.

The second scoring scheme, s_{opt}, calculates the optimistic alignment score of the wildcard w against each residue. The optimistic score is the highest score for aligning residue q to any of the residues represented by wildcard w. This is calculated as follows:

$$s_{opt}(w, r) = \max_{f \in w} s(r, f)$$

The optimistic score guarantees no loss in sensitivity: the score for aligning against a union-sequence U using this scoring scheme is at least as high as the score for any of the sequences represented by U. The problem is that in many cases the score for U is significantly higher, leading to false-positives where the union-sequence is flagged as a match despite none of the cluster members being sufficiently close to the query. The result is substantially slower search.

The expected and optimistic scoring schemes represent two different compromises between sensitivity and speed. We can adjust this balance by combining the two approaches using a mixture model. We define a mixture parameter, λ, such that $0 \leq \lambda \leq 1$. The mixture-model score for aligning wildcard w to residue r is defined as:

$$s_\lambda(w, r) = \lambda s_{opt}(w, r) + (1 - \lambda) s_{exp}(w, r)$$

The score $s_\lambda(w, r)$ for each w, r pair is calculated when the collection is being clustered and then recorded on disk. During a BLAST search, the wildcard scores are loaded from disk and used to perform the search. We report experiments with varying values of λ in Section 7.

6 Selecting wildcards

Having defined a system for assigning a scoring vector to an arbitrary wildcard, we now describe a method for selecting a set of wildcards to be used during the clustering process. Each wildcard w represents a set of residues $w \subseteq R$ and can be used in a union-sequence to substitute for any set of residues of which it is a superset. A set of wildcards, $W = \{w_1, ..., w_n\}$ is used during clustering. We assume that one of these wildcards w_n is the default wildcard that can be used to represent any of the 24 residue and ambiguous codes, that is $w_n = R$. The remaining wildcards must be selected carefully; large residue sets can be used more frequently but provide poor discrimination with higher average alignment scores and more false positives. Small residue sets can be used less frequently, increasing the use of larger residue sets such as the default wildcard.

The first aspect of choosing a set of wildcards to use for substitution is to decide on the size of this set. It would be ideal to use as many wildcards as necessary, so that each substitution $s_i = o_i$. However, each wildcard must be encoded as a different character leading to an extremely large alphabet. An enlarged alphabet would in turn lead to inefficiencies in BLAST due to larger lookup and scoring data structures. Thus, a compromise is required. BLAST uses a set of 20 character codes to represent residues, as well as 4 IUPAC-IUBMB ambiguous residue codes and an end-of-sequence code, for a total of 25 distinct codes. Each code is represented using 5 bits, permitting a total of 32 codes. This leaves 7 unused character codes. We have therefore chosen to use $|W| = 7$ wildcards.

We treat the task of selecting a good set of wildcards as an optimisation problem. To do this, we first cluster the collection as described in Section 4 using only the default wildcard, *ie.* $W = \{w_d\}$. We use the residue-substitution sequence O from this clustering to create a set W^* of candidate wildcards. Our goal can then be defined as follows: we wish to select the set of wildcards $W \subseteq W^*$ such that the total average alignment score $A = \sum\limits_{w \in S} \sum\limits_{r \in R} s(w,r)p(r)$ for all substitutions S is minimised. A lower A implies a reduction in the number of high-scoring matches between a typical query sequence and union-sequences in the collection, thereby reducing the number of false-positive situations in which cluster members are fruitlessly recreated and aligned to the query.

In selecting the wildcard set W that minimises A we use the following greedy approach: first, we initialize W to contain only the default wildcard w_d. We then scan through W^* and select the wildcard that leads to the greatest overall reduction in A. This process is repeated until the set W is filled, at each iteration considering the wildcards already in W in the calculation of A. Once W is full we employ a hill-climbing strategy where we consider replacing each wildcard with a set of residues from W^* with the aim of further reducing A.

A set of wildcards was chosen by applying this strategy to the GenBank NR database described in Section 7. The following wildcards were identified and are used for all reported experiments: `LVIFM, GEKRQH, AVTIX, SETKDN, LVTPRFYMHCW, AGSDPH, LAGSVETKDPIRNQFYMHCWBZXU`.

We also considered defining wildcards based on groups of amino acids with similar physico-chemical properties by using the amino acid classifications described in Taylor (1986). However, a preliminary investigation of this approach resulted in 3% slower search times and reduced search accuracy compared to the approach we have described.

7 Results

The Structural Classification of Proteins (SCOP) database (Murzin et al., 1995; Andreeva et al., 2004) is widely used to evaluate the accuracy of sequence search tools (Brenner et al., 1998; Park et al., 1998). For our own assessments, we used version 1.65 of the ASTRAL Compendium (Chandonia et al., 2004) that uses information from the SCOP database to classify sequences with fold, superfamily, and family information. The database contains a total of 67,210 sequences classified into 1,538 superfamilies.

A set of 8,759 test queries were extracted from the ASTRAL database such that no two of the queries shared more than 90% identity. To measure search accuracy, each query was searched against the ASTRAL database and the Receiver Operating Characteristic (ROC) score (Gribskov and Robinson, 1996) was calculated. A match between two sequences was considered positive if they came from the same superfamily, otherwise it was considered negative. The ROC_{50} score provides a measure between 0 and 1, where a higher score represents better sensitivity (detection of true positives) and selectivity (ranking true positives ahead of false positives).

The SCOP database is too small to provide an accurate measure of search time, so we use the GenBank non-redundant (NR) protein database to measure search times. The GenBank collection was downloaded August 18, 2005 and contains 2,739,666 sequences in around 900 megabytes of sequence data. Performance was measured using 50 queries randomly selected from GenBank NR. Each query was searched against the entire collection three times with the best runtime recorded and the results averaged. Experiments were conducted on a Pentium 4 2.8GHz machine with two gigabytes of main memory.

We used FSA-BLAST[1]—our open-source version of BLAST—with default parameters as a baseline. To assess the clustering scheme, the GenBank and ASTRAL databases were clustered and FSA-BLAST was configured to report all high-scoring alignments, rather than only the best alignment from each cluster. All reported collection sizes include sequence data and edit information but exclude sequence descriptions. CD-HIT version 2.0.4 beta was used for experiments with 95% clustering threshold and maximum memory set to 1.5 Gb. We also report results for NCBI-BLAST version 2.2.11 and our own implementation of Smith-Waterman that uses the exact same scoring functions and statistics as BLAST (Karlin and Altschul, 1990). No sequence filtering was performed.

The overall results of our clustering method are shown in Table 1. When used with default settings of $\lambda = 0.2$ and $T = 0.25$, our clustering approach reduces the overall size of the NR database by 27% and improves search times by 22%. Importantly, the ROC score indicates that there is no significant effect on search accuracy, with the highly redundant SCOP database reducing in size by 80%

[1] Available from: `http://www.fsa-blast.org`

Scheme	GenBank NR		ASTRAL
	Time	Sequence data	
	secs (% baseline)	Mb (% baseline)	ROC$_{50}$
FSA-BLAST			
No clustering (baseline)	28.75 (100%)	900 (100%)	0.398
Cluster $\lambda = 0.2, T = 0.25$	22.54 (78%)	655 (73%)	0.398
Cluster $\lambda = 0, T = 0.3$	20.97 (73%)	650 (72%)	0.397
NCBI-BLAST	45.75 (159%)	898 (100%)	0.398
Smith-Waterman	—	—	0.415

Table 1. Average runtime for 50 queries searched against the GenBank NR database, and SCOP ROC$_{50}$ scores for the ASTRAL collection.

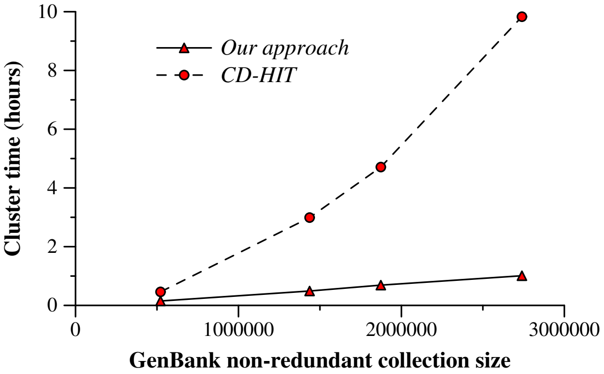

Fig. 2. Clustering performance for GenBank NR databases of varying sizes.

when clustered. If users are willing to accept a small loss in accuracy, then the parameters $\lambda = 0$ and $T = 0.3$ improve search times by 27% and reduce the size of the sequence collection by 28% with a decrease of 0.001 in ROC score when compared to our baseline. Since we are interested in improving performance with no loss in accuracy we do not consider these non-default settings further. Overall, our clustering approach with default parameters combined with improvements to the gapped alignment (Cameron et al., 2004) and hit detection (Cameron et al., 2005) stages of BLAST more than double the speed of FSA-BLAST compared to NCBI-BLAST with no significant effect on accuracy. Both versions of BLAST produce ROC scores 0.017 below the optimal Smith-Waterman algorithm.

Figure 2 shows a comparison of clustering times between CD-HIT and our clustering approach for four different releases of the GenBank NR database; details of the collections used are given in Table 2. The results show that the clustering time of our approach is linear with the collection size and the CD-HIT approach is superlinear (Figure 2). On the recent GenBank non-redundant collection, CD-HIT is almost 10 times slower than our approach; we expect this ratio to further increase with collection size.

Table 2 shows the amount of redundancy in the GenBank NR database as it has grown over time, measured using our clustering approach. We observe

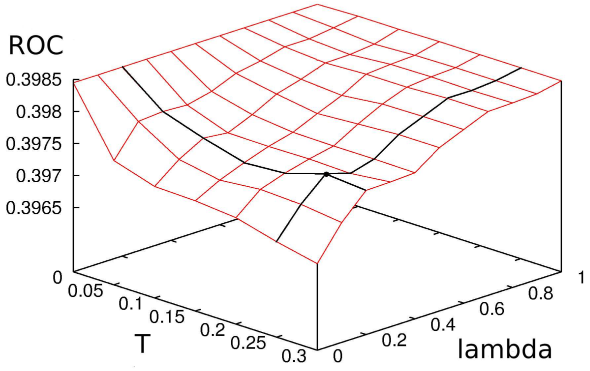

Fig. 3. Search accuracy for collections clustered with varying values of λ and T. Default values of $\lambda = 0.2, T = 0.25$ are highlighted.

Release date	Number of sequences	Collection Size (Mb)	Overall size reduction (Mb)	Percentage of collection
16 July 2000	521,662	157	45	28.9%
22 May 2003	1,436,591	443	124	28.1%
30 June 2004	1,873,745	597	165	27.4%
18 August 2005	2,739,666	900	245	27.3%

Table 2. Redundancy in GenBank NR database over time.

that the degree of redundancy is not changing significantly with the percentage reduction through clustering remaining between 27% and 29% across versions of the collection tested.

Figure 3 shows the effect on accuracy for varying values of λ and T. We have chosen $\lambda = 0.2$ as a default value because smaller values of λ result in a larger decrease in search accuracy, and larger values reduce search speed. We observe that for $\lambda = 0.2$ there is little variation in search accuracy for values of T between 0.05 and 0.3.

Figure 4 shows the effect on search times for varying values of T where $\lambda = 0.2$. As T increases the clustered collection becomes smaller, leading to faster search times. However, if T is too large then union-sequences with a high percentage of wildcards are permitted, leading to an increase in the number of cluster members that are recreated and a corresponding reduction in search speed. We have chosen the value $T = 0.25$ that maximises search speed.

8 Conclusion

Sequence databanks such as GenBank contain a large number of redundant sequences. Such redundancy has several negative effects including larger collection size, slower search, and difficult-to-interpret results. Redundancy within a collection can lead to over-representation of alignments within particular protein domains, distracting the user from other potentially important hits.

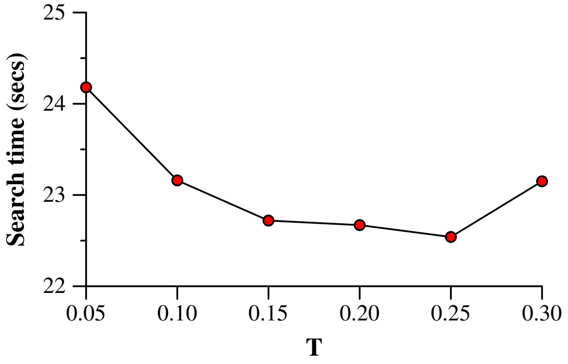

Fig. 4. Average BLAST search time using $\lambda = 0.2$ and varying values of T.

We have proposed a new scheme for managing redundancy. Instead of discarding near-duplicate sequences, our approach identifies clusters of redundant sequences and constructs a special union-sequence that represents all members of the cluster through the careful use of wildcard characters. We present a new approach for searching clusters that, when combined with a well-chosen set of wildcards and a system for scoring matches between wildcards and query residues, leads to faster search times without a significant loss in accuracy. Moreover, by recording the differences between the union-sequence and each cluster member using edit information our approach compresses the collection. Our scheme is general and can be adapted to most homology search tools.

We have integrated our algorithm into FSA-BLAST, a new version of BLAST that is substantially faster than NCBI-BLAST and freely available for download at `http://www.fsa-blast.org/`. Our results show that our clustering scheme reduces BLAST search times against the GenBank non-redundant database by 22% and compresses sequence data by 27% with no significant effect on accuracy. We have also described a new system for identifying clusters that uses fingerprinting, a technique that has been successfully applied to duplicate-document detection in information retrieval. Our implementation can cluster the entire GenBank NR protein database in one hour on a standard workstation and scales linearly in the size of the collection. We propose that pre-clustered copies of the GenBank collection be made publicly available for download.

We have confined our experimental work to protein sequences and plan to investigate the effect of our clustering scheme on nucleotide data as future work. We also plan to investigate the effect of our approach on iterative search algorithms such as PSI-BLAST, and how our scheme can be used to improve the current measure of the statistical significance of BLAST alignments.

Acknowledgements

The authors thank Peter Smooker and Michelle Chow for valuable suggestions. This work was supported by the Australian Research Council.

References

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

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 Research*, 25(17):3389–3402.

Andreeva, A., Howorth, D., Brenner, S., Hubbard, T., Chothia, C., and Murzin, A. (2004). SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Research*, 32:D226–D229.

Bernstein, Y. and Cameron, M. (2006). Fast discovery of similar sequences in large genomic collections. In *Proc. European Conference on Information Retrieval*. To appear.

Bernstein, Y. and Zobel, J. (2004). A scalable system for identifying co-derivative documents. In Apostolico, A. and Melucci, M., editors, *Proc. String Processing and Information Retrieval Symposium (SPIRE)*, pages 55–67, Padova, Italy. Springer.

Bernstein, Y. and Zobel, J. (2005). Redundant documents and search effectiveness. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 736–743, New York, NY, USA. ACM Press.

Bleasby, A. J. and Wootton, J. C. (1990). Construction of validated, non-redundant composite protein sequence databases. *Protein Engineering*, 3(3):153–159.

Brenner, S., Chothia, C., and Hubbard, T. (1998). Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences USA*, 95(11):6073–6078.

Brin, S., Davis, J., and García-Molina, H. (1995). Copy detection mechanisms for digital documents. In Carey, M. and Schneider, D., editors, *Proceedings of the ACM SIGMOD Annual Conference*, pages 398–409, San Jose, California, United States. ACM Press.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166.

Burke, J., Davison, D., and Hide, W. (1999). d2_cluster: A validated method for clustering EST and full-length DNA sequences. *Genome Research*, 9(11):1135–1142.

Cameron, M., Williams, H. E., and Cannane, A. (2004). Improved gapped alignment in BLAST. *IEEE Transactions on Computational Biology and Bioinformatics*, 1(3):116–129.

Cameron, M., Williams, H. E., and Cannane, A. (2005). A deterministic finite automaton for faster protein hit detection in BLAST. *Journal of Computational Biology*. To appear.

Chandonia, J., Hon, G., Walker, N., Conte, L. L., Koehl, P., Levitt, M., and Brenner, S. (2004). The ASTRAL compendium in 2004. *Nucleic Acids Research*, 32:D189–D192.

Cheung, C.-F., Yu, J. X., and Lu, H. (2005). Constructing suffix tree for gigabyte sequences with megabyte memory. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):90–105.

Gracy, J. and Argos, P. (1998). Automated protein sequence database classification. i. integration of compositional similarity search, local similarity search, and multiple sequence alignment. *Bioinformatics*, 14(2):164–173.

Gribskov, M. and Robinson, N. (1996). Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers & Chemistry*, 20:25–33.

Grillo, G., Attimonelli, M., Liuni, S., and Pesole, G. (1996). CLEANUP: a fast computer program for removing redundancies from nucleotide sequence databases. *Computer Applications in the Biosciences*, 12(1):1–8.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press.

Heintze, N. (1996). Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, pages 191–200, Oakland, California, USA.

Henikoff, S. and Henikoff, J. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA*, 89(22):10915–10919.

Holm, L. and Sander, C. (1998). Removing near-neighbour redundancy from large protein sequence collections. *Bioinformatics*, 14(5):423–429.

Johnson, S. (1967). Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254.

Kallberg, Y. and Persson, B. (1999). KIND — a non-redundant protein database. *Bioinformatics*, 15(3):260–261.

Karlin, S. and Altschul, S. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences USA*, 87(6):2264–2268.

Lee, C., Grasso, C., and Sharlow, M. F. (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464.

Li, W., Jaroszewski, L., and Godzik, A. (2001a). Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17(3):282–283.

Li, W., Jaroszewski, L., and Godzik, A. (2001b). Tolerating some redundancy significantly speeds up clustering of large protein databases. *Bioinformatics*, 18(1):77–82.

Li, W., Jaroszewski, L., and Godzik, A. (2002). Sequence clustering strategies improve remote homology recognitions while reducing search times. *Protein Engineering*, 15(8):643–649.

Malde, K., Coward, E., and Jonassen, I. (2003). Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19(10):1221–1226.

Manber, U. (1994). Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA.

Manber, U. and Myers, G. (1993). Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948.

Murzin, A., Brenner, S., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540.

Park, J., Holm, L., Heger, A., and Chothia, C. (2000). RSDB: representative sequence databases have high information content. *Bioinformatics*, 16(5):458–464.

Park, J., Karplus, K., Barrett, C., Hughey, R., Haussler, D., Hubbard, T., and Chothia, C. (1998). Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4):1201–1210.

Pearson, W. and Lipman, D. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441.

Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences USA*, 85(8):2444–2448.

Robinson, A. and Robinson, L. (1991). Distribution of glutamine and asparagine residues and their near neighbors in peptides and proteins. *Proceedings of the National Academy of Sciences USA*, 88(20):8880–8884.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

Taylor, W. (1986). The classification of amino-acid conservation. *Journal of Theoretical Biology*, 119:205–218.