Fast Discovery of Similar Sequences in Large Genomic Collections

Yaniv Bernstein Michael Cameron

School of Computer Science and Information Technology
RMIT University, Melbourne, Australia
{ybernste, mcam}@cs.rmit.edu.au

Abstract. Detection of highly similar sequences within genomic collections has a number of applications, including the assembly of expressed sequence tag data, genome comparison, and clustering sequence collections for improved search speed and accuracy. While several approaches exist for this task, they are becoming infeasible — either in space or in time — as genomic collections continue to grow at a rapid pace. In this paper we present an approach based on document fingerprinting for identifying highly similar sequences. Our approach uses a modest amount of memory and executes in a time roughly proportional to the size of the collection. We demonstrate substantial speed improvements compared to the CD-HIT algorithm, the most successful existing approach for clustering large protein sequence collections.

1 Introduction

Similarity between genomic sequences is a strong predictor of functional or phylogenetic relatedness, and thus the identification of sequence similarity is a very important application in bioinformatics. The first step towards identifying a new sequence typically involves searching a large sequence databank for similar sequences using an alignment algorithm such as Smith-Waterman (Smith & Waterman 1981), FASTA (Pearson & Lipman 1988) or BLAST (Altschul et al. 1990, 1997, Cameron et al. 2004, 2005). These algorithms compare a single query sequence against a collection of sequences, and are notable for their accuracy and sensitivity. In many cases, however, it is useful to know not just the similarity between a single sequence and a collection, but between *all* sequences in the collection. The identification of similar sequence pairs is useful for clustering EST (expressed sequence tag) data (Burke et al. 1999, Malde et al. 2003), genome comparison (Kurtz et al. 2004), and reducing redundancy in a collection to improve search speed (Holm & Sander 1998, Li et al. 2001*b*,*a*) and accuracy (Park et al. 2000, Li et al. 2002). In such cases, a straightforward application of Smith-Waterman or BLAST is not appropriate: an all-against-all comparison of a 100 Mb collection takes several days with BLAST.

Previous studies have investigated a range of approaches to identifying all pairs of highly similar sequences in a collection. Most of the proposed techniques execute significantly faster than a naïve application of query-based algorithms

such as BLAST. However, a majority of these algorithms still have a fundamental $O(n^2)$ complexity in the size of the collection, rendering them increasingly infeasible as genomic databases continue their exponential growth. Malde et al. (2003) have investigated the use of suffix structures such as suffix trees (Gusfield 1997) or suffix arrays (Manber & Myers 1993) to efficiently identify high-scoring pairs in a single pass of the collection. This approach does not suffer from the quadratic complexity problem, however suffix structures have significant memory overheads and long construction times, making them unsuitable for large genomic collections such as GenBank.

In this paper we describe and apply *document fingerprinting* to isolate candidate pairs of highly similar sequences. Our approach is fast, scales linearly with collection size, and has modest memory requirements. We describe our method for applying fingerprinting to genomic sequences and find that it is remarkably accurate and sensitive for this task. We also apply fingerprinting to the creation of representative-sequence databases (Holm & Sander 1998, Park et al. 2000, Li et al. 2001*b*). We are able to process the GenBank non-redundant database in around 1.5 hours, while the fastest existing approach, CD-HIT (Li et al. 2001*b*), requires over 9 hours for the same task. Importantly, there is no significant change in accuracy.

2 Similarity detection: techniques and applications

Query-based similarity detection in sequence collections is a fundamental task in bioinformatics. It is often the first step in the identification, classification and comparison of new sequence data. A number of well-researched and well-established techniques exist for this task; alignment algorithms such as BLAST compare a single query sequence to every sequence in a collection and are generally considered satisfactory solutions for this task.

For some important applications, however, there is no notion of a query sequence; rather, it is necessary to identify similarity between arbitrary pairs of sequences in a collection. For example, the assembly of EST (expressed sequence tag) data involves arranging a collection of overlapping sequences into a longer consensus sequence. For this application there is no apparent query sequence: rather, we are interested in similarity between any pair of sequences in the collection. Another application where an all-against-all comparison is required is the construction of a representative-sequence database (RSDB), where highly redundant sequences are removed from a collection resulting in faster, more sensitive search for distant homologies using search algorithms such as PSI-BLAST (Altschul et al. 1997).

Several past solutions — including Holm & Sander (1998) and Li et al. (2001*b*) — use a simple pairwise approach to identify pairs of similar sequences. These schemes use fast BLAST-like heuristics to compare each sequence in the collection to the entire collection. The representative-sequence database tool CD-HIT (Li et al. 2001*b*) is the fastest approach based on this method. However, despite fast methods for comparing each sequence pair, such approaches require

time that is quadratic in the size of the collection and are increasingly infeasible as genomic collections continue to grow. The CD-HIT tool requires over 9 hours to process the current GenBank non-redundant protein database.

An alternative approach to identifying similar sequences involves using a suffix structure such as a suffix tree or suffix array. This approach is taken by Malde et al. (2003), where suffix arrays are used to cluster EST sequences in linear time. While the approach is highly effective for this application — in which collections are typically quite small — suffix structures are known to consume large amounts of main-memory. In our experiments with the freely available XSACT software, this was confirmed: the software required more than 2 Gb of main memory to process a 10 Mb collection of uncompressed nucleotide data. Although more compact suffix structures exist (Grossi & Vitter 2000) they have longer construction and search times.

In the following sections we describe a novel, alternative approach called document fingerprinting with linear time complexity and modest memory requirements.

3 Document fingerprinting and SPEX

Document fingerprinting (Manber 1994, Brin et al. 1995, Heintze 1996, Broder et al. 1997, Shivakumar & García-Molina 1999) is an effective and scalable technique for identifying pairs of documents within large text collections that share portions of identical text. Document fingerprinting has been used for several applications, including copyright protection (Brin et al. 1995), document management (Manber 1994) and web search optimisation (Broder et al. 1997, Fetterly et al. 2003, Bernstein & Zobel 2005).

The fundamental unit of document fingerprinting techniques is the *chunk*, a fixed-length unit of text such as a series of consecutive words or a sentence. The full set of chunks for a given document is formed by passing a sliding window of appropriate length over the document; this is illustrated below for a chunk length of six words:

	`[the quick brown fox jumped over]`
the quick brown fox jumped	`[quick brown fox jumped over the]`
over the lazy dog	`[brown fox jumped over the lazy]`
	`[fox jumped over the lazy dog]`

The set of all chunks in a collection can be stored in an inverted index (Witten et al. 1999) and the index can be used to calculate the number of shared chunks between pairs of documents in a collection. Two identical documents will naturally have an identical set of chunks. As the documents begin to diverge, the proportion of chunks they share will decrease. However, any pair of documents sharing a run of text as long as the chunk length will have at least one chunk in common. Thus, the proportion of common chunks is a good estimator of the quantity of common text shared by a pair of documents. The quality of this estimate is optimised by choosing a chunk length that is long enough so that

```
for chunkLength = 1 to finalLength
   foreach sequence in the collection
     foreach chunk of length chunkLength in sequence
       if chunkLength = 1
         increment lookup[chunk]
       else
         subchunk1 = chunk prefix of length chunkLength - 1
         subchunk2 = chunk suffix of length chunkLength - 1
         if lookup[subchunk1] = 2+ and lookup[subchunk2] = 2+
           increment lookup[chunk]
```

Fig. 1. The SPEX algorithm

two identical chunks are unlikely to coincidentally occur, but not so long that it becomes too sensitive to minor changes. In the DECO package, for example, the default chunk length is eight words (Bernstein & Zobel 2004).

For practical document fingerprinting, chunks are generally hashed before storage in order to make their representation more compact. Further, some sort of *selection heuristic* is normally applied so that only some chunks from each document are selected for storage. The choice of selection heuristic has a very significant impact on the general effectiveness of the fingerprinting algorithm. Most fingerprinting algorithms have used simple feature-based selection heuristics, such as selecting chunks only if their hash is divisible by a certain number, or selecting chunks that begin with certain letter-combinations. These heuristics are obviously lossy: if two documents share chunks, but none of them happen to satisfy the criteria of the selection heuristic, the fingerprinting algorithm will not identify these documents as sharing text.

Bernstein & Zobel (2004) introduced the SPEX chunk selection algorithm, which allows for *lossless* selection of chunks, based on the observation that singleton chunks (chunks that only occur once and represent a large majority in most collections) do not contribute to identifying text reuse between documents. The SPEX algorithm takes advantage of the fact that, if any subchunk (subsequence) of a chunk is unique, the chunk as a whole is unique. Using a memory-efficient iterative hashing technique, SPEX is able to select only those chunks that occur multiple times in the collection. Using SPEX can yield significant savings over selecting every chunk without any degradation in the quality of results.

Figure 1 provides a pseudocode sketch of how SPEX identifies duplicate chunks of length `finalLength` within a collection of documents or genomic sequences. The algorithm iterates over chunk lengths from 1 to `finalLength`, the final chunk length desired. At each iteration, SPEX maintains two hashtables (referred to as `lookup` in the figure): one recording the number of occurrences of each chunk for the previous iteration, and one for the current iteration. As we are only interested in knowing whether a chunk occurs multiple times or not, each entry in `lookup` takes one of only three values: zero, one, or more than one (2^+). This allows us to fit four hashtable entries per byte; collisions are not resolved. A chunk is only inserted into `lookup` if its two subchunks of length `chunkLength - 1` both appear multiple times in the hashtable from the previous iteration. The iterative process helps prevent the hashtables from being flooded. The SPEX algorithm is able to process quite large collections of text and indicate whether a given chunk

occurs multiple times in a reasonable time, and consuming a relatively modest amount of memory. For a full description of how the SPEX algorithm works, we refer the reader to Bernstein & Zobel (2004).

4 Fingerprinting for genomic sequences

The SPEX algorithm (and, indeed, any fingerprinting algorithm) can be trivially adapted for use with genomic sequences by simply substituting documents with sequences. However, the properties of a genomic sequence are quite different from those of a natural language document. The most significant difference is the lack of any unit in genomic data analogous to natural language words. The protein sequences we consider in this paper are represented as an undifferentiated string of amino-acid characters with no natural delimiters such as whitespace, commas or other punctuation marks.

The lack of words in genomic sequences has a number of immediate impacts on the operation and performance of the SPEX algorithm. First, the granularity of the sliding window must be increased from word-level to character-level. An increased granularity means that there will be far more chunks in a genomic sequence than in a natural-language document of similar size. As a result, the SPEX algorithm is less efficient and scalable for genomic data than for natural language documents.

The distribution of subsequences within genomic data is also less highly skewed than the distribution of words in English text. Given a collection of natural language documents, we expect some words (such as 'and' and 'or') to occur extremely frequently, while other words (such as perhaps 'alphamegamia' and 'nudiustertian') will be *hapax legomena*: words that occur only once. This permits the SPEX algorithm to be effectual from the first iteration by removing word-pairs such as 'nudiustertian news'. In contrast, given a short string of characters using the amino acid alphabet of size 20, it is far less likely that the word will occur only once in any collection of nontrivial size. Thus, the first few iterations of SPEX are likely be entirely ineffectual.

One simple solution to these problems is to introduce 'pseudo-words', effectively segmenting each sequence by moving the sliding window several characters at a time. However, this approach relies on sequences being aligned along segment boundaries. This assumption is not generally valid and makes the algorithm highly sensitive to insertions and deletions. Consider, for example, the following sequences given a chunk length of four and a window increment of four:

	Sequence	Chunks
Sequence 1	AB**CDEFGHIJKLM**NOP	ABCD EFGH IJKL MNOP
Sequence 2	AAB**CDEFGHIJKLM**NOP	AABC DEFG HIJK LMNO
Sequence 3	GHAA**CDEFGHIJKLM**Q	GHAA CDEF GHIJ KLMQ

Despite all three of these sequences containing an identical subsequence of length 11 (in bold above), they do not share a single common chunk. This strong correspondence between the three sequences will thus be overlooked by the algorithm.

```
chunkLength = finalLength - Q × (numIterations - 1)
for iteration = 1 to numIterations
   foreach sequence in the collection
     foreach chunk of length chunkLength in sequence
       if lookup[chunk] ≠ 0
         increment lookup[chunk]
       else
         count number of subchunks of length chunkLength - Q
               where lookup[subchunk] = 2+
         if (count ≥ 2 or iteration = 1) and
            (number of chunks processed since increment lookup ≥ Q)
             increment lookup[chunk]
   increment chunkLength by Q
```

Fig. 2. The slotted SPEX algorithm

We propose a hybrid of regular SPEX and the pseudo-word based approach described above that we call slotted SPEX. Slotted SPEX uses a window increment greater than one but is able to 'synchronise' the windows between sequences so that two highly-similar sequences are not entirely overlooked as a result of a misalignment between them.

Figure 2 describes the slotted SPEX algorithm. As in standard SPEX, we pass a fixed-size window over each sequence with an increment of one. However, unlike SPEX, slotted SPEX does not consider inserting every chunk into the hashtable. In addition to decomposing the chunk into subchunks and checking that the subchunks are non-unique, slotted SPEX also requires that one of two initial conditions be met. First, that it has been at least `Q` window increments since the last insertion; or second, that the current chunk already appears in the hashcounter. The parameter `Q` is the *quantum*, which can be thought of as the window increment used by the algorithm. Slotted SPEX guarantees that at least every $\texttt{Q}^{th}$ overlapping substring from a sequence is inserted into the hashtable. The second precondition — that the chunk already appears in the hashcounter — provides the synchronisation that is required for the algorithm to work reliably.

The operation of slotted SPEX is best illustrated with an example. Using the same set of sequences as above, a quantum `Q` = 4 and a chunk length of four, slotted SPEX produces the following set of chunks:

	Sequence	Chunks
Sequence 1	AB**CDEFGHIJKL**MNOP	ABCD **EFGH IJKL** MNOP
Sequence 2	AAB**CDEFGHIJKL**MNOP	AABC ABCD **EFGH IJKL** MNOP
Sequence 3	GHAA**CDEFGHIJKL**MQ	GHAA CDEF **EFGH IJKL**

For the first sequence, the set of chunks produced does not differ from the naïve pseudo-word technique. Let us now follow the process for the second sequence. The first chunk — `AABC` — is inserted as before. When processing the second chunk, `ABCD`, the number of chunks processed since the last insertion is one, fewer than the quantum `Q`. However, the condition `lookup[chunk]` ≠ `0` on line 5 of Figure 2 is met: the chunk has been previously inserted. The hashcounter is therefore incremented, effectively synchronising the window of the sequence with that of the earlier, matching sequence. As a result, every $\texttt{Q}^{th}$ identical

chunk will be identified across the matching region between the two sequences. In this example, the slotted SPEX algorithm selects two chunks of length four that are common to all sequences. Slotted SPEX also differs from regular SPEX by incrementing the word length by `Q` rather than 1 between iterations.

In comparison to the ordinary SPEX algorithm, slotted SPEX requires fewer iterations, consumes less memory and builds smaller indexes. This makes it suitable for the higher chunk density of genomic data. While slotted SPEX is a lossy algorithm, it does offer the following guarantee: for a window size `finalLength` and a quantum `Q`, any pair of sequences with a matching subsequence of length `finalLength + Q - 1` or greater will have at least one identical chunk selected. As the length of the match grows, so will the guaranteed number of common chunks selected. Thus, despite the lossiness of the algorithm, slotted SPEX is still able to offer strong assurance that it will reliably detect highly similar pairs of sequences.

5 Fingerprinting for identity estimation

In this section, we analyze the performance of slotted SPEX for distinguishing sequence pairs with a high level of identity from those that do not.

Following Holm & Sander (1998) and Li et al. (2001*b*), we calculate the percentage identity between a pair of sequences by performing a banded Smith-Waterman alignment (Chao et al. 1992) using a band width of 20, match score of 1, and no mismatch or gap penalty. The percentage identity I for the sequence pair s_i, s_j is calculated as $I = S(s_i, s_j)/L(s_i, s_j)$ where $S(s_i, s_j)$ is the alignment score and $L(s_i, s_j)$ is the length of the shorter of the two sequences. This score can be functionally interpreted as being the proportion of characters in the shorter sequence that match identical characters in the longer sequence. We define similar sequence pairs as those with at least 90% identity ($I \geq 0.9$); this is the same threshold used in Holm & Sander (1998) and is the default parameter used by CD-HIT (Li et al. 2001*b*).

For experiments in this section we use version 1.65 of the ASTRAL Compendium (Chandonia et al. 2004), because it is a relatively small yet complete database that allows us to experiment with a wide range of parameterisations. The ASTRAL database contains 24,519 sequences, equating to 300,578,421 unique sequence-pair combinations. Of these, 139,716 — less than 0.05% — have an identity of 90% or higher by the above measure; this is despite the fact that the database is known to have a high degree of internal redundancy. A vast majority of sequence pairs in any database can be assumed to be highly dissimilar.

Although we do not expect fingerprinting to be as sensitive and accurate as a computationally intensive dynamic-programming approach such as Smith-Waterman, we hope that the method will effectively distinguish sequence-pairs with a high level of identity from the large number of pairs that have very low identity. Our aim is to use document fingerprinting to massively reduce the search space within which more sensitive analysis must be pursued. For example, even if fingerprinting identifies three times as many false positives (dissimilar sequences)

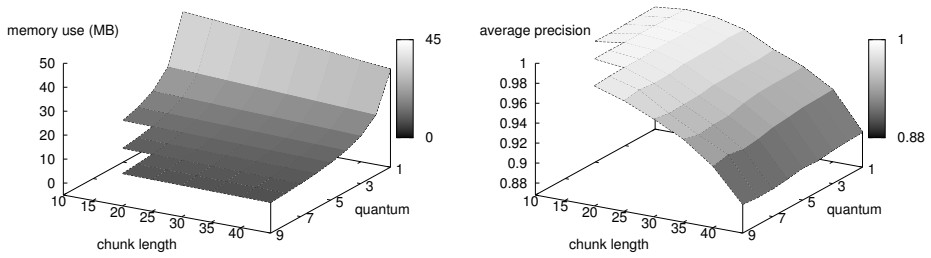

Fig. 3. SPEX index size as a function of final chunk length and quantum (left) and average precision as a function of final chunk length and quantum (right)

as true positives (similar sequences), less than 0.2% of all sequence pairs in the ASTRAL collection would need to be aligned.

In order to find a good compromise between resource consumption and effectiveness, we have experimented with different parameter combinations. Figure 3 (left) shows the SPEX index size for varying chunk lengths and quanta. The results show that increasing the word length does not result in a large reduction in index size, but increasing the quantum results in a marked and consistent decrease in the size of the index.

Figure 3 (right) plots the average precision (Buckley & Voorhees 2000) as a function of chunk length and quantum. The average precision measure was calculated by sorting pairs in decreasing order of SPEX score — the number of matching chunks divided by the length of the shorter sequence — and using sequence pairs with an identity of 90% or above as the set of positives. We observe that increasing the chunk length results in a small loss in accuracy, however increasing the quantum has almost no effect on average precision. This indicates that slotted SPEX — even with a high quantum — is able to estimate sequence identity nearly as well as the regular SPEX algorithm with reduced costs in memory use, index size and index processing time.

The result in Figure 3 make a strong case for using a shorter word length; however, shorter words place a greater loading on the hashcounter. With larger collections, memory bounds can lead to the hashtable flooding and a consequent blowout in index size. Thus, shorter word lengths are less scalable. Similarly, longer quanta are in general beneficial to performance. However, a larger quantum reduces the number of iterations possible in slotted SPEX. Thus, a very high quantum can result in more collisions in the hashcounter due to fewer iterations, suggesting once again that a compromise is required. Guided by these observations along with the other data, chunk lengths of 25 or 30 with a quantum of 5 to 9 appear to provide a good compromise between the various considerations in all-against-all identity detection for large collections.

The high average precision results indicate that slotted SPEX provides an accurate and sensitive prediction of whether sequence pairs have a high level of identity. What is particularly surprising is that the average precision stays high even with reasonably long chunk lengths and high quanta. Earlier efforts by

Holm & Sander (1998) and Li et al. (2001*b*), and Li et al. (2001*a*) are extremely rigorous and rely upon short matching chunks, typically less than ten characters in length, between sequence-pairs before proceeding with alignment. Our results indicate that longer chunk lengths have only minor impact on result quality.

In our experiments we have focused on identifying sequence-pairs with greater than 90% identity, and we have shown that fingerprinting is effective at this task. However, it is probable that fingerprinting will prove less useful as the identity threshold is lowered.

6 Removing redundant sequences: an application

Holm & Sander (1998), Park et al. (2000) and Li et al. (2001*b*) have all investigated techniques for creating *representative-sequence databases* (RSDBs), culled collections where no two sequences share more than a given level of identity. RSDBs are typically constructed by identifying clusters of similar sequences and retaining only one sequence from each cluster, the cluster representative. Such databases are more compact, resulting in faster search times. More significantly, they have been demonstrated to improve the sensitivity of distant-homology search algorithms such as PSI-BLAST (Li et al. 2002).

The most recent and efficient technique for constructing an RSDB, CD-HIT (Li et al. 2001*b*), uses a greedy incremental approach based on an all-against-all comparison. The algorithm starts with an empty RSDB. Each sequence is processed in decreasing order of length and compared to every sequence already inserted into the RSDB. If a high-identity match is found, where I exceeds a threshold, the sequence is discarded; otherwise it is added to the RSDB. To reduce the number of sequence pairs that are aligned, CD-HIT first checks for short matching chunks — typically of length four or five — between sequences before aligning them. The approach is still fundamentally quadratic in complexity.

We have replicated the greedy incremental approach of CD-HIT, but use fingerprinting with slotted SPEX as a preprocessing step to dramatically reduce the number of sequence comparisons performed. A list of candidate sequence-pairs, for which the SPEX score exceeds a specified threshold, is constructed. We only perform alignments between sequence pairs in this candidate list. This is significantly faster than comparing each sequence to all sequences in the RSDB.

We measured the performance and scalability of our approach by comparing it to CD-HIT — which is freely available for download — using several releases of the comprehensive Genbank non-redundant (NR) protein database over time[1]. We used the CD-HIT default threshold of $T = 90\%$ and the four releases of GenBank NR database from July 2000 until August 2005 described in Table 1. For tests with CD-HIT we used default parameters except for `max_memory` which we increased to 1.5 Gb. For our approach, we used a final chunk length `finalLength` of 25, a quantum of 9 and 3 iterations. Our threshold for identifying a candidate

[1] Ideally, we would have had more datapoints for this experiment. However, old releases of the NR database are not officially maintained, and thus we could only find four different releases of the database.

Table 1. Reduction in collection size for CD-HIT and our approach for various releases of the GenBank NR database.

	Original	Size reduction	
Release date	Size (Mb)	CD-HIT	Our approach
16 July 2000	157	61.71 Mb (39.56%)	61.72 Mb (39.57%)
22 May 2003	443	164.38 Mb (37.33%)	165.07 Mb (37.48%)
30 June 2004	597	217.80 Mb (36.71%)	218.76 Mb (36.87%)
18 August 2005	900	322.98 Mb (36.08%)	324.92 Mb (36.30%)

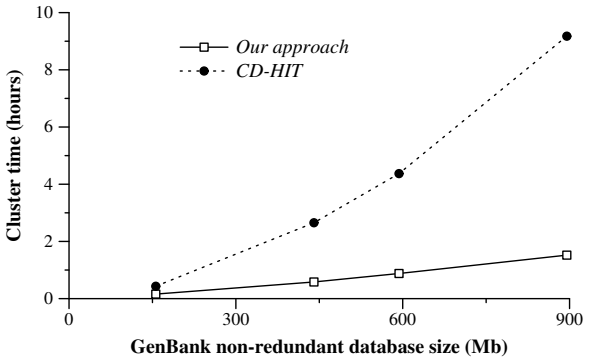

Fig. 4. Time required to identify and remove redundant sequences from various releases of the GenBank NR database.

pair is one matching chunk between the pair. We use this low threshold because we have found that it provides improved accuracy with a negligible increase in execution time. In our experiments with the ASTRAL database described previously and our chosen default parameters, slotted SPEX identifies only 10,143 false positives out of 147,724 sequence pairs identified.

The results in Table 1 show no significant difference in representative collection size between our method and CD-HIT, indicating the two approaches are roughly equivalent in terms of accuracy. Figure 4 shows the runtime for our approach and CD-HIT for the releases of GenBank tested. A visual inspection reveals that our approach scales roughly linearly with the size of the collection while CD-HIT is superlinear. When processing the recent August 2005 collection, our approach is more than 6 times faster than CD-HIT.

7 Conclusions

The identification of highly-similar sequence pairs in genomic collections has several important applications in bioinformatics. Previous solutions to this problem involve either an all-against-all comparison with $O(n^2)$ complexity or the use of suffix structures that suffer from large main-memory overheads or long construction times. Therefore, existing approaches are not suitable for processing large collections such as GenBank.

We have applied document fingerprinting techniques to genomic data with the aim of more efficiently identifying pairs of similar sequences in large collections. We have described a new algorithm called slotted SPEX that requires less main-memory and CPU resources when processing genomic collections. We show that slotted SPEX is highly accurate for identifying high-identity sequence pairs, even with long chunk lengths and large quanta. We have also tested the effectiveness of our slotted SPEX approach for removing redundant sequences from large collections. When processing the recent GenBank non-redundant protein database our scheme is more than 6 times faster than the previous fastest approach, CD-HIT, with no significant change in accuracy. Further, our approach scales approximately linearly with collection size.

As future work, we plan to investigate the effectiveness of our approach on nucleotide data. We also plan to apply our slotted SPEX algorithm to English text in applications where the original SPEX algorithm has proved successful.

Acknowledgements

This work was supported by the Australian Research Council.

References

Altschul, S., Gish, W., Miller, W., Myers, E. & 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. & Lipman, D. (1997), "Gapped BLAST and PSI–BLAST: A new generation of protein database search programs", *Nucleic Acids Research* **25**(17), 3389–3402.

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

Bernstein, Y. & Zobel, J. (2005), Redundant documents and search effectiveness, *in* A. Chowdhury, N. Fuhr, M. Ronthaler, H. Schek & W. Teiken, eds, "Proc. CIKM conference", ACM Press, Bremen, Germany, pp. 736–743.

Brin, S., Davis, J. & García-Molina, H. (1995), Copy detection mechanisms for digital documents, *in* "Proceedings of the ACM SIGMOD Annual Conference", pp. 398–409.

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

Buckley, C. & Voorhees, E. M. (2000), Evaluating evaluation measure stability, *in* "Proc. ACM SIGIR conference", ACM Press, pp. 33–40.

Burke, J., Davison, D. & 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. & Cannane, A. (2004), "Improved gapped alignment in BLAST", *IEEE Transactions on Computational Biology and Bioinformatics* **1**(3), 116–129.

Cameron, M., Williams, H. E. & 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. & Brenner, S. (2004), "The ASTRAL compendium in 2004", *Nucleic Acids Research* **32**, D189–D192.

Chao, K., Pearson, W. & Miller, W. (1992), "Aligning two sequences within a specified diagonal band", *Computer Applications in the Biosciences* **8**(5), 481–487.

Fetterly, D., Manasse, M. & Najork, M. (2003), On the evolution of clusters of near-duplicate web pages, *in* R. Baeza-Yates, ed., "Proc. 1st Latin American Web Congress", IEEE, Santiago, Chile, pp. 37–45.

Grossi, R. & Vitter, J. S. (2000), Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract), *in* "STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing", ACM Press, New York, NY, USA, pp. 397–406.

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".

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

Kurtz, S., Phillippy, A., Delcher, A., Smoot, M., Shumway, M., Antonescu, C. & Salzberg, S. (2004), "Versatile and open software for comparing large genomes", *Genome Biology* **5**(2).

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

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

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

Malde, K., Coward, E. & 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", San Fransisco, CA, USA, pp. 1–10.

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

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

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

Shivakumar, N. & García-Molina, H. (1999), Finding near-replicas of documents on the web, *in* "WEBDB: International Workshop on the World Wide Web and Databases, WebDB", Springer-Verlag.

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

Witten, I. H., Moffat, A. & Bell, T. C. (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kauffman.