

Indexing Genomic Databases for Fast Homology Searching

Twee-Hee Ong¹, Kian-Lee Tan^{1,2}, and Hao Wang¹

¹ Department of Computer Science
National University of Singapore

3 Science Drive 2, Singapore 117543
{ongtweeh, tankl, wanghao}@comp.nus.edu.sg

² Genome Institute of Singapore
1 Science Park Road, The Capricorn #05-01
Singapore Science Park II, Singapore 117528
gistankl@nus.edu.sg

Abstract. Genomic sequence databases has been widely used by molecular biologists for homology searching. However, as amino acid and nucleotide databases are growing in size at an alarming rate, traditional brute force approach of comparing a query sequence against each of the database sequences is becoming prohibitively expensive. In this paper, we re-examine the problem of searching for homology in large protein databases. We proposed a novel filter-and-refine approach to speed up the search process. The scheme operates in two phases. In the filtering phase, a small set of candidate database sequences (as compared to all sequences in the database) is quickly identified. This is realized using a signature-based scheme. In the refinement phase, the query sequence is matched against the sequences in the candidate set using any local alignment strategies. Our preliminary experimental results show that the proposed method results in significant savings in computation without sacrificing on the accuracy of the answers as compared to FASTA.

1 Introduction

Molecular biologists frequently query genomic databases in their search for sequence homology. Traditionally, this process involves matching a query sequence against each of the database sequences where the similarity computations between two sequences typically have a complexity that is either linear or quadratic to the length of the sequences involved. However, several recent development have led to such brute force approach to be no longer practical. First, genomic databases are increasing in size in terms of both number of sequences and length of the sequence (doubling in size every 15 or 16 months [1]). Second, the number of queries directed at these databases are very large (over 40,000 queries per day [1]) and the user numbers and query rates are growing. Third, there is an increasing need to mine a sequence database for useful information. This typically requires all pairwise similarity between all the sequences to be computed.

Thus, the need to move away from exhaustive search techniques will keep getting stronger, and novel and efficient methods for searching genomic databases are necessary and called for.

One promising direction in the literature is to maintain an abstraction or index of the database sequences that can be used to identify a (small) subset of the database sequences that are broadly similar to the query. In this way, those database sequences that are not likely to be in the answer set can be pruned away without being accessed, thus saving both disk and computation cost. The candidate answer set is subsequently refined using a more effective alignment algorithm to pick out the final answer set.

Most of the existing works represent a sequence by its subsequences (also called *motifs*). Thus, sequences with the same set of motifs are likely to be homologues. However, these schemes typically restrict to fixed-length motifs. Moreover, they are storage inefficient, and their effectiveness (sensitivities) are limited because of the lack of inexact match when comparing motifs.

In this paper, we present a new indexing scheme to speedup homology search. The scheme is novel in the following ways. First, our approach can handle motifs of different length. Second, we account for inexact match when comparing motifs. Finally, a sequence's motifs are represented by a m -bit vector, called a *signature*. We note that m can be fine-tuned in order to trade off performance and storage. At one extreme, if m corresponds to the number of motifs, then, each motif can be mapped to a unique position in the vector. At the other extreme, if $m = 1$, then the vector is effectively useless as all motifs will be mapped to the same bit. The set of all signatures corresponding to the database sequences form the signature file.

The search process is a filter-and-refine strategy. In the filtering phase, motifs are extracted from the query sequence to generate a query signature. The query signature is matched against the signature file to return a candidate set of sequences. This step can be performed quickly as bitwise operations are very fast. In the refinement phase, the query sequence is matched against the candidate sequences using any exhaustive approach to pick out the answer set.

We implemented the proposed scheme, and conducted a preliminary experimental study on a real dataset. Our results show that a reasonable size m is sufficient for good performance. Moreover, the proposed scheme is significantly faster than FASTA [2] without sacrificing on the quality of the answers returned.

The rest of this paper is organized as follows. In the next section, we review some related work. Section 3 presents the proposed signature-based scheme. In Section 4, we report on the results of our experiments, and finally, we conclude in Section 5 with directions for further study.

2 Related Work

SSearch [3], FASTA [2] and BLAST [4] are all examples of exhaustive search tools. Each entry in the database is consulted before formulating an answer set

to a query. Index-based systems have been studied. Here, we review three such schemes that we extract from [5].

RAMdb (Rapid Access Motif database) [6] is a system for finding short patterns (called *motifs*) in genomic databases. Each genomic sequence is indexed by its constituent overlapping intervals in a hash table structure. For each interval, an associated list of sequence numbers and offsets is stored. This allows a quick lookup of any sequences matching a query sequence. A long query sequence is split into shorter non-overlapping motifs that are used to query the database. RAMdb is best suited for the lookup of query motifs whose length is equal or slightly longer than the indexed interval length. RAMdb requires a large index twice the size of the original flat-file database (including the textual descriptions) and suffers from lack of special-purpose ranking schemes designed for identifying initial match regions. In addition, the non-overlapping interval of query motifs means false dismissals unless the frame happens to coincidentally align with the start of the interval frame.

The FLASH search tool redundantly indexes genomic data based on a probabilistic scheme [7]. For each interval of length n , the FLASH search structure stores, in a hash-table, all possible similarly-ordered contiguous and non-contiguous subsequences of length m that begin with the first base in the interval, where $m < n$. As an example, for a nucleotide sequence ACCTGATT the index terms for the first $n = 5$ bases, where $m = 3$, would be ACC, ACT, ACG, ACT, ACG and ATG; each of the permuted strings begins with the base A, the first base in the interval of length $n = 5$. The hash-table then stores each permuted m -length subsequence, the sequences that contain the permuted subsequences, and the offsets within each sequence of the permuted subsequence. The key idea of the flash scheme is that the permuted scheme gives an accurate model that approximates a reasonable number of insertions, deletions, and substitutions in genomic sequences. The authors found that FLASH was of the order of ten times faster for a small test collection than BLAST and was superior in accurately and sensitively determining homologies in database searching. FLASH utilizes a redundant index, which is stored in a hash-table and is uncompressed, and impractically large. For a nucleotide collection of around 100 Mb, the index requires 18 Gb on disk, around 180 times the collection size.

CAFE [1] is based on a partitioned search approach, where a coarse search using an inverted index is used to rank sequences by similarity to a query sequence and a subsequent fine search used to locally align only a subset of database sequences with the query. The CAFE index consists of three components: a search structure, which contains the index terms or distinct intervals, that is, fixed-length overlapping subsequences from the collection being indexed; inverted lists, which are a carefully compressed list of ordinal sequence numbers, where each list is an index of sequences containing a particular interval; and, a mapping table, which maps ordinal sequence numbers to the physical location of sequence data on disk. Queries are evaluated by representing the query as a set of intervals, retrieving the list for each interval, and using a ranking structure to store a similarity score of each database sequence to the query. Like FLASH, CAFE

uses an overlapping interval. It uses a compression scheme to try to make the index size more manageable. The author notes that although it is more computationally efficient than the exhaustive methods, “exhaustive systems generally have better retrieval effectiveness”.

3 Signature-based Retrieval

In this section, we present the proposed signature-based retrieval scheme. The mechanism involves several components, and we shall discuss each of them. First, we need to extract a set of features that can be used to represent the sequences. Second, the extracted information is encoded as a signature, and a signature file is used to facilitate speedy retrievals. Third, we need to know how to compute the similarity between two sequences.

3.1 Representing the semantics of a sequence

In this paper, we represent each sequence by a set of *independent* subsequences extracted from the sequence. We refer to these subsequences as *motifs*. This concept resembles that of representing a document by its keywords in information retrieval research.

For the approach to be practical and useful, several issues have to be addressed. First, we need to determine a suitable set of motifs. Traditionally, one approach of finding motifs is to extract all subsequences whose length is between l_{min} and l_{max} that satisfy a given minimum support constraint [8]. Our algorithm, shown in Figure 1, is more robust and allows motifs of different length to be obtained. Essentially, given a subsequence, we count the number of database sequences that contain the subsequence. For those that are too small, i.e., below a minimum support constraint β , we do not treat them as motifs. For those that appear in too many sequences, i.e., above a maximum support constraint α , we also do not treat them as motifs. This is because such subsequences are not discriminating enough. Instead, we extend the subsequence length and use the extended subsequence as the motifs. We note that steps 1 to 10 of the algorithm allows the algorithm to pick a suitable starting length for motifs, if most of the subsequences of a certain length appear in many sequences, then it makes sense to start with a longer subsequence.

Second, given that we have identified a set of motifs, associate each database sequence with a subset of the motifs. To make the search effective, we want motifs of a sequence to be as unrelated or independent as possible. Two motifs are said to be dependent if and only if (a) either one is the prefix of the other or one is subsequence of the other, and (b) their intersection of their respective supporting sets is non-trivial [8]. Thus, for each sequence, we first find the set of motifs it supports and from this set we select a maximal set of independent ones.

To illustrate, Table 1 shows an example. On the left, we have 8 protein sequences. Suppose, we assume that we use a minimum support of 50%, i.e.,

Algorithm GenerateAllMotifs**Input:** maximum support α , minimum support β , threshold γ

1. $l_{min} \leftarrow 3$
2. repeat
3. $tooShort \leftarrow \text{False}$
4. generate all subsequences of length l_{min}
5. let k_i denote the number of sequences in the database that corresponds to motif m_i
6. let M be the number of motifs whose k_i value is greater than α
7. if $M > \gamma$
8. $tooShort \leftarrow \text{True}$
9. $l_{min} \leftarrow l_{min} + 1$
10. until not($tooShort$)
11. for each subsequence s generated with length l_{min}
12. let c_s denote the number of sequences that contain s
13. if $\alpha > c_s > \beta$
14. s is added into the list of motifs
15. else if $c_s > \alpha$
16. extend s and include its extensions as motifs
17. step 16 may be repeated for the extended motifs if their count exceeds α
18. return the complete list of motifs

Fig. 1. Generation of motifs.

subsequences that appear in fewer than 4 sequences will be pruned away. Further, assume that we use a maximum support of 75%, i.e., subsequences that appear in 6 sequences or more will be extended. The middle table shows the motifs (those without ‘*’) that will be generated. Consider a subsequence that has been marked with ‘*’ in the table, say A Q V. We note that A Q V appears in 6 database sequences. As such, it is extended to A Q V H (since it is the only one that appear in the database). Interestingly A Q V H also appears in 6 database sequences, which resulted in it being further extended to A Q V H K, A Q V H M and A Q V H P. Thus, the latter 3 subsequences are captured as motifs instead of A Q V and A Q V H. The table on the right shows the selected motifs for each database sequence. Here, we see that for sequence S_1 , we have A Q V H and A Q V H K which are dependent, and A Q V H K is selected as it is “maximal”. Similarly, for sequence S_6 , we have D A L and A L G which overlapped. In this case, we arbitrarily picked one of them.

The third issue concerns an efficient and compact strategy to represent the motifs of the database sequences. Our approach is to map each database sequence to a V -bit vector (called a *signature*) as follows. Consider a sequence with k motifs, m_1, \dots, m_k . Initially, all the V bits of the signature is set to 0. We define a hash function $h(m)$ that maps a motif m to a value in the range 1 to V . Then, for motif m_i ($1 \leq i \leq k$), bit $h(m_i)$ is set to 1. Note that depending on the hash function used, *collision* can occur, i.e., it is possible for $m_i \neq m_j$ such that

Table 1. Motif generation example

Seq ID	Sequence	Pattern	Support	Seq ID	Selected Motifs
S_1	AQVHKHKKSV DAM	ALG	4	S_1	AQVHK, HKKS
S_2	AQVHKKSGSDGLP	*AQV	6	S_2	KKS, AQVHK
S_3	AQVHKHVAQIKDP	DAL	4	S_3	AQVHK, IKD
S_4	AQVHKALGPHKKS	HKK	4	S_4	AQVHK, ALG, HKKS
S_5	DALGPAQVHMHKKS	IKD	4	S_5	ALG, AQVHM, HKKS
S_6	AQIKDDALGPAQP	KKS	5	S_6	DAL, IKD
S_7	KKSPQIKDQVG	QIK	4	S_7	KKS, IKD
S_8	QIKDALGMAQVHP	*QVH	6	S_8	ALG, AQVHP, IKD
		QVHK	4		
		QVHM	1		
		QVHP	1		
		*AQVH	6		
		AQVHK	4		
		AQVHM	1		
		AQVHP	1		

$h(m_i) = h(m_j)$. In such a case, we expect *false drops* to occur, i.e., two dissimilar sequences may have the a significant number of bits set at the same locations and are treated as similar. The collection of all signatures produced by all the database sequences is the signature file.

3.2 The Similarity Measure

Two sequences are similar if they share some common motifs. Under the signature-based representation of sequences, the signatures representing both sequences are similar if they may only differ in some of the bits. This only requires a simple operation (logical AND) to compute the intersection between two signatures.

Let Q and D denote the signatures of a query sequence and a database sequence respectively. Then, the two sequences have the same motif if and only if the corresponding bits in both signatures are set (note that it is possible to be a false drop). Thus, the similarity measure, SIM , between Q and D can be determined as:

$$SIM(Q, D) = \frac{BitSet(Q \wedge D)}{BitSet(D)} \quad (1)$$

where $BitSet(BS)$ denotes the number of bits in the vector BS that are set, and ' \wedge ' represents the bitwise logical-AND operation. Now, if both sequences share many common motifs, the similarity computed will be closed to 1.

3.3 The Retrieval Process

The retrieval process comprises two phases. In the first phase, we identify a small set of candidate sequences (from the database sequences) that are homologous to

the query sequence quickly. This is done using the signature files as follows. Given a query sequence, we first determine its signature. This requires finding out the set of motifs that can be extracted from the sequence. This is done efficiently as follows. We use a window of length l_{min} and generate the subsequences of l_{min} as the window slides through the sequence. The subsequences are then sorted and compared against the motifs of the database sequences (this can be efficiently done using integer comparison, rather than string comparison). This process may be repeated for extended subsequences. Once the set of motifs is identified, the resultant signature is compared against those stored in the signature file. The database sequences that are similar can then be ranked and retrieved accordingly, and the top ranked sequences form the candidate set.

In the second phase, we compare the query sequence against each of the database sequences in the candidate set using a more comprehensive matching algorithm such as FASTA or BLAST. The best set of sequences can then be obtained.

3.4 Optimized Scheme for Improved Retrieval Effectiveness

In the above discussion, we have presented a very basic signature-based approach whose effectiveness may be limited. Here, we present extensions that can improve its effectiveness:

- In the similarity metric, when comparing two bit vectors, we have assumed that every motif is equally important. To account for frequently occurring low complexity motifs, we scale each of the motifs following the *inverse-document-frequency* methodology. This is also commonly used in information retrieval research. In this approach, if a particular motif appears in m out of n sequences, its weight is multiplied by $\log(n/m)$. The effect of this scaling is that infrequently occurring motifs are given higher weight than motifs that occur in almost every sequence.
- In our earlier discussion, we have pruned away those motifs whose counts fall below the minimum support. To avoid missing such motifs completely, we mapped such motifs to those that are similar to them. Referring to our example in Table 1, DAM in sequence S_1 may be treated as DAL.
- Another limitation in the above discussion is that the ordering of the motifs is not considered. To resolve this, we create an intermediate layer between the signature file and the database sequences. Each signature of a sequence also serves as an inverted list by pointing to the list of motifs associated with the sequence (note that the list of motifs is encoded, and not stored as strings). Instead of retrieving the database sequences, once the candidate sequences are identified, an alignment algorithm is applied on the list of motifs associated with them to further rank them. The refined ranking is then used to further prune the list of candidate set to a smaller set before the final matching of query and database sequences are performed.

4 Preliminary Results

To study the effectiveness (accuracy) and efficiency of the proposed signature retrieval method, we conducted some performance study. We implemented the basic signature scheme as described using the global approach [8] and present our results in this section. The dataset used for the experiment is essentially the entire protein sequence database, PIR1-PIR4, maintained by the National Biomedical Research Foundation (NBRF-PIR) at the Georgetown University Medical Center. This set has about 190,874 protein sequences.

4.1 On Retrieval Effectiveness

To study the retrieval effectiveness of the proposed method, we use FASTA as the basis for comparison. In other words, we consider our method accurate as long as it can return the same set of answers as that returned by FASTA.

Table 2 shows the results of the experiments. For the experiments, we set l_{min} to 3 and 4, and minimum support to 4.0% and 0.9% of the dataset respectively. For $l_{min}=3$, there are a total of 3322 motifs generated, while 3326 motifs were generated for $l_{min}=4$. In the figure, the first column represents the percentage of the database sequences that are retrieved as the candidate set after the filtering phase; the second and third column denotes the percentage of answers that are in FASTA but not contained among the database sequences retrieved in the signature scheme with l_{min} set to 3 and 4 respectively; the fourth column shows the percentage of answers that are in FASTA but not contained in the database sequences retrieved using an inverted file index (similar to that described in CAFE [1]), where length of q-gram was set to 4³.

From the results, we note that if we retrieve the top 10% of the sequences from phase 1 of the proposed scheme, the scheme missed about 12.5% of the answers produced by FASTA, with $l_{min}=3$. The results are slightly worse for $l_{min}=4$. This is expected as the probability of two sequences having longer exact motifs is lower. Also, by comparing the signature scheme to the inverted file implementation, we note that the accuracy of the inverted file is better than the scheme. This is expected, as the inverted file indexes all the q-grams that are present in the database, while for the signature scheme, the motifs that are below the minimum support are not present in the signature index. However, the space requirements of the inverted file is much more than that required by the signature scheme, and by a suitable length used for the motifs, we are able to obtain nearly the same accuracy as the inverted file implementation.

Also, it is encouraging to note that by scanning 20% of the database sequences, we can get reasonably good accuracy as the results obtained by FASTA. Our investigation also shows that the answers that are missed ranked fairly low under FASTA's answers. In other words, all the high ranking sequences are also ranked highly under the proposed scheme.

³ At the time of the experiments, we were only able to obtain the implementation for q-gram of 4

Table 2. On retrieval effectiveness

% of sequences retrieved	% of answers missed (signature)	% of answers missed (signature)	% of answers missed (inverted file)
	$l_{min}=3$	$l_{min}=4$	
1%	40.0%	47.5%	13.3%
2%	35.0%	40.83%	9.17%
5%	16.0%	33.3%	5.83%
10%	12.5%	22.5%	5.83%
15%	11.67%	14.17%	5.0%
20%	7.5%	10.8%	4.16%

Table 3. On retrieval efficiency

length of query	signature-based method			FASTA	Inverted File
	Phase 1	Phase 2	Total		Phase 1
100 bp	58.8	0.18	58.98	78.41	13
200 bp	58.2	2.05	60.25	132.59	45
300 bp	58.4	15.11	73.51	177.55	71
400 bp	58.8	17.16	75.96	209.51	104
500 bp	58.9	20.76	79.66	240.74	127
600 bp	59.0	20.35	79.35	249.38	126

4.2 On Retrieval Efficiency

Table 3 shows the running time results of the experiments with l_{min} and length of q-gram for inverted file both set to 4. The results obtained are over multiple runs with multiple different queries. The timings for the proposed signature scheme are presented for all the stages; including phase 2 which is using FASTA on the top 10% of the database sequences retrieved in phase 1. The timings shown for the inverted file shows only the time to retrieve all records of all the q-grams present in the query sequence. First we observe that the query string length seems to have an effect on the cost of the algorithm: as the query length increases, the processing cost also increases. Second we note that all queries incur approximately the same cost in phase 1. This is expected since each query sequence’s bit vector has to be compared against all vectors in the signature file. Third, we find that the proposed scheme is generally more efficient for long query sequences. Its gain can be more than 50% that of FASTA. For short sequences, the gain is less significant.

5 Conclusion

In this paper, we have proposed a novel filter-and-refine approach to speed up homology search in large databases. In the filtering phase, a signature file is used to determine a small subset of candidate database sequences that are potentially homologous to the query sequence. In the refinement phase, the candidate

set and the query sequences are aligned using any known alignment algorithm. Our preliminary results on real data set showed that the proposed approach is promising. In particular, we can significantly reduce the processing cost without sacrificing much of the accuracy. We believe that the proposed scheme's efficiency can be further improved by partitioning the signature file according to the protein (super-)families. We would also like to exploit the sequence length to facilitate more efficient search. We are currently working in these directions.

References

1. H. Williams and J. Zobel. Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering (to appear)*, 2001.
2. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. In *Proceedings Natl. Acad. Sci. USA Vol. 85*, pages 2444–2448, 1988.
3. D. J. States, W. Gish, and S. F. Altschul. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods: A Companion to Methods in Enzymology*, 3(1):66–70, 1991.
4. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
5. M. Dipperstein. Dna sequence databases. In <http://www.cs.ucsb.edu/mdipper/dna/DNApaper.html>.
6. C. Fondrat and P. Dessen. A rapid access motif database (ramdb) with a search algorithm for the retrieval patterns in nucleic acids or protein databanks. *Computer Applications in the Biosciences*, 11(3):273–279, 1995.
7. A. Califano and I. Rigoutsos. Flash: A fast look-up algorithm for string homology. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, pages 56–64, Bethesda, MD, 1993.
8. V. Guralnik and G. Karypis. A scalable algorithm for clustering protein sequences. In *Proceedings of the BIOKDD 2001 Workshop (see <http://www.cs.rpi.edu/zaki/BIOKDD01>)*, 2001.