SST: An Algorithm for Searching Sequence Databases in Time Proportional to the Logarithm of the Database Size

Eldar Giladi[1] **Michael G. Walker**[1]
`egiladi@incyte.com` `mwalker@incyte.com`
James Ze Wang[2] **Wayne Volkmuth**[1]
`wangz@Cs.Stanford.Edu` `volkmuth@incyte.com`

[1] Research Informatics, Incyte, 3174 Porter Drive, Palo Alto CA 94303, USA
[2] Department of Computer Science, Stanford University, Stanford CA 94089, USA

Keywords: sequence similarity search, clustering, k-tuple encoding

1 Introduction

In the current efforts to generate and interpret the complete genome sequences of humans and model organisms, large scale searches for near exact sequence matches are frequently performed. Examples include programs that assemble DNA from shotgun sequencing projects which initially search for overlapping fragments, large scale searches of EST databases against genomic databases to determine the location of genes, and cross species genomic comparisons between very closely related genomes. We have developed an algorithm, called SST (Sequence Search Tree), that searches a database of DNA sequences for near exact matches, in time proportional to the logarithm of the database size n.

2 The SST algorithm

In SST, we partition each sequence into fragments of fixed length W called "windows" using multiple offsets. Windows begin at position $j * \Delta$, and end at position $j * \Delta + W$, with $j = 0, 1, 2 \dots$. Typical values of the parameters are $0 \leq W \leq 1000$ and $5 \leq \Delta \leq W/2$. Then each window is mapped into a vector of dimension 4^k that represents the frequency of occurrence of its component k-tuples. Specifically, the I'th entry in the vector is the number of occurrences of tuple I in that window. Here we associate the integer

$$I(a_1 a_2 \cdots a_k) = \sum_{l=1}^{k} 4^l M(a_l),$$

with the k-tuple of nucleotides $a_1 a_2 \cdots a_k$, $a_i \in \{\text{A,C,G,T}\}$ where $M(a_l) = 0, 1, 2, 3$ for $a_l =$ A, $a_l =$ C, $a_l =$ G, $a_l =$ T, respectively. Typically $4 \leq k \leq 6$.

The mapping has transformed the problem of finding near exact matches to query windows in the database to that of finding nearest neighbors in vector space. This search can be done very efficiently by constructing a tree-structured index of vectors. We construct the index using tree structured vector quantization (TSVQ), by recursively searching the data for clusters that provide binary (or higher-order) partitions using k-means clustering [2].

We identify the nearest-neighbors of a query sequence by partitioning the query into *non-overlapping* windows and searching the tree-structured index for nearest neighbor windows in the database. Only windows which are at L_1 distance less than a threshold T are returned. However, SST is not guaranteed to return all nearest neighbors at distance $\leq T$, as we shall see in our computations. Our work is most closely related to [1, 3, 4].

3 Computational Results

We illustrate the performance of SST by applying it to detecting overlapping fragments in shotgun sequence assembly. We fragment a 1.5 megabase sequence of genomic DNA several times using a Poisson process with $\lambda = 300$ nucleotides. From the pool of fragments we generate a set of 61350 sequences thus simulating a 12-fold coverage of the genomic stretch. We then randomly introduce errors into each fragment at a rate of 5%. Insertions, deletions and substitutions where equally likely.

We apply SST to search the database of fragments against itself for overlapping sequences, with overlap size ≥ 50. In our computations the window size W and the tolerance T where varied in the range $30 \leq W \leq 50$ and $0 \leq T \leq 30$.

Figure 1 indicates the true positive rate (TPR) and the $\log_{10}$ of the false positive rate (FPR) as a function of W and T. The TPR increases as W decreases and as T increases, and so does the FPR. With optimal parameters $W = 30$, $T = 15$, the TPR and FPR are .95 and 0.0007, respectively.

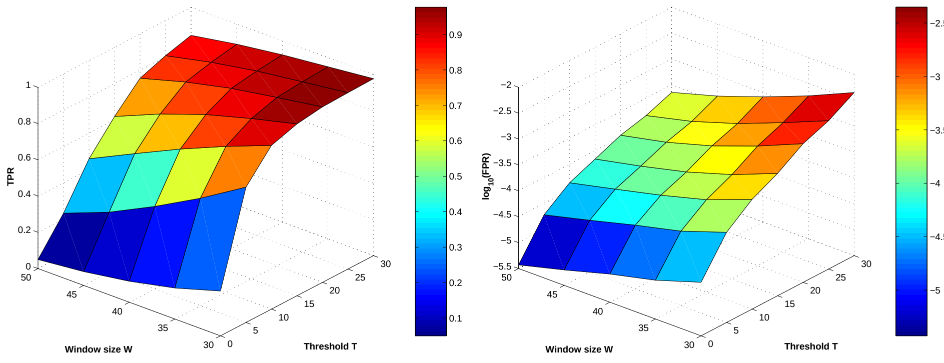

Figure 1: The TPR as a function of the window size W and the the distance threshold T on the left. The **log** base 10 of the FPR as a function of the window size W and the distance threshold T on the right. Data has an error rate of 5 %.

We also compare the speed of SST to BLAST in order to highlight the computational complexity of SST. In this computation we vary the coverage of the genomic stretch. The following Table shows the linear scaling of the time per query with BLAST versus the nearly constant time with SST.

Coverage	BLAST search time	BLAST time per query	SST time per query
$6\times$	02 : 03 : 09	0.2409	0.0164
$12\times$	07 : 27 : 47	0.4379	0.0186
$24\times$	28 : 10 : 25	0.8266	0.0173

References

[1] Burkhardt, S., *et al.*, Q-gram based database searching using a suffix array (QUASAR), *Proc. RECOMB 99*, ACM press, 1999.

[2] Gersho, A. and Gray, R., *Vector Quantization and Signal Compression*, Kluwer, 1992.

[3] Ukkonen, E., Approximate string-matching with q-grams and maximal matches, *Theoretical Computer Science*, 92(1):191–211, 1992.

[4] Yona, G., *Methods for global organization of all known protein sequences*, Computer Science, Hebrew University, 1999.