# ESTmapper: Efficiently Aligning DNA Sequences to Genomes

Xue Wu, Woei-Jyh (Adam) Lee, Chau-Wen Tseng

Department of Computer Science
University of Maryland at College Park
{wu,adamlee,tseng}@cs.umd.edu

## Abstract

*With improvements in technology, scientists are able to sequence the full DNA (genome) of an increasing number of organisms. One way biologists can take advantage of this genomic sequence data is to use it in conjunction with expressed sequence tag (EST) information to find genes and their splice sites. We describe how ESTmapper uses an eager write-only top-down (WOTD) suffix tree to efficiently align DNA sequences against genomes, and compare its precision and performance against popular techniques for DNA alignment (BLAT, sim4, Spidey, BLAST, megaBLAST) and EST clustering (TGICL and PaCE). Experimental results show that ESTmapper is 3 to 1000 times faster than current techniques for aligning and clustering DNA sequences, and produces alignments of comparable or better quality.*

## 1. Introduction

Recent advances in molecular biology techniques such as automated DNA sequencing have allowed scientists to quickly gather huge amounts of DNA sequence data. Two types of DNA sequence data are particularly interesting: genomic DNA and *expressed sequence tags* (ESTs). Genomes are long (thousands to millions of bases) DNA sequences representing the complete DNA of an organism, carefully constructed with high accuracy from many experiments. Deciphering the function of each portion of the genome remains one of the major challenges facing molecular biologists.

In comparison, ESTs are short (400-800 bases) single-pass DNA sequences. Despite being fragmentary and error-prone, ESTs are of interest because they represent portions of the genomic DNA that are actually transcribed into mRNA (expressed), and can be used to find genes and their splice sites. ESTs are also relatively easy to collect and sequence, making up 62% of the 38.9 million nucleotide sequences in NCBI GenBank (Oct'04, release 144) [1].

1 But because EST sequences are relatively short, they comprise only 29% of the total number of bases in GenBank.

With the increasing number of species whose genomes have been sequenced, the ability to efficiently and precisely map ESTs and other DNA sequences to genomes becomes an important tool for biologists. Such mappings can be used to answer a number of important questions, such as gene finding [26], EST clustering [3, 18], finding alternative splicing [4, 10], and identifying gene function [24].

Since the ability to align DNA sequences to the genome is so useful, researchers have investigated many techniques for performing such alignments [7, 15, 22, 26, 27]. However, due to the large size of many genomes (around 3 billion bases for Human and Mouse) and the large number of DNA sequences collected by biologists, aligning DNA to genomes can pose a computational challenge. Using standard sequence alignment techniques such as BLAST may be too expensive when a single high-throughput automated DNA sequencer (e.g., ABI Prism 3730xl) can output two million bases of sequence per day. Producing accurate alignments is also problematic, since genomes contain many very similar if not duplicate DNA sequences that can result in multiple plausible alignments to different portions of the genome. Careful analysis is needed to distinguish between the possible alignments and calculate the most plausible mapping.

In this paper, we describe ESTmapper, a new tool for aligning DNA sequences to genomes based on an eager *write-only, top-down* (WOTD) suffix tree for the genome. While other researchers have used suffix trees to speed up pairwise sequence alignment [20] and perform EST-to-EST [13, 19] or genome-to-genome [5, 16] alignments, we believe we are the first to use suffix trees for EST-to-genome alignments. Using the common substrings found by suffix trees as a starting point, we design algorithms to efficiently compute precise alignments of DNA sequences to genomes. Preliminary experiments suggest that ESTmapper is precise and very efficient when compared to other popular alignment and EST clustering tools. The algorithm can also be easily parallelized for improved performance.

## 2. Related Work

### 2.1. Sequence Alignment

Pairwise sequence alignment compares two sequences to find and align the most similar substrings based on some metric. It is probably the most commonly performed computation in bioinformatics, and many algorithms have been developed. Basic Local Alignment Search Tool (BLAST) [2] is the most popular and widely used tool for sequence alignment and similarity search. Here we focus on algorithms used by researchers to align DNA to genomes.

*megaBLAST* is a program from the NCBI BLAST software suite that uses a greedy algorithm to align nucleotide sequences [29]. The program provides good performance for highly similar sequences with minor differences, and is thus frequently used for genome alignments.

*BLAT* is an alternative pairwise sequence alignment algorithm [15]. BLAT maintains a precomputed hash table index of the locations of all non-overlapping substrings (words) of length $k$. Performance is dramatically improved because queries do not need to scan the entire sequence database. Only the sections of the sequence database with hits in the index need to be examined to compute more detailed local alignments and scores. Substrings that yield too many (hundreds or more) hits to the genome can be filtered out and ignored, or alignment time may increase dramatically.

*Sim4* is one of the oldest and most frequently used programs for aligning spliced DNA sequence with genomic sequence, allowing introns and a small number of sequencing errors [7]. Unlike BLAST, it attempts to recognize biological valid splice sites for non-contiguous alignments to the genome. Sim4 was created because of the inefficiency of BLAST when mapping large numbers of cDNA sequences to genomes. Researchers use Sim4 for studying gene-to-genome annotation and alternative splicing. However, Sim4 can be slow since it uses dynamic programming. Older EST alignment tools such as *est_genome* [21] and *est2gen* [8] also use dynamic programming, and too slow to search entire genomes.

*Spidey* is a computer program to align spliced sequences to genomic sequences [27]. It is incorporated in the NCBI Toolkit for biologists to study gene annotation and alternative splicing. To find good alignments, Spidey uses NCBI BLAST to produce a list of candidate alignments, then refines the alignments while considering splice sites.

*Squall* is a tool similar to BLAT, but especially designed to map ESTs to genomes [22]. Squall uses lookup tables to quickly find candidate substrings (21 bases) within 300 bases of the beginning and end of each EST sequence. It improves efficiency by discarding all candidates that map to more than ten locations in the genome. If the distance between the start and stop candidates is less than 3 million bases, a more precise algorithm is used to calculate and score possible EST to genome alignments.

The authors claim squall is 100 times faster than BLAT. However, we find BLAT to be much faster than reported in their paper, indicating the authors may have not filter out excessively common substrings from the BLAT hash index as recommended. The authors report 0.03, 1.69, and 12 seconds to align each RefSeq sequence to human chromosome 22 using Squall, BLAT, and sim4 on a PrimePower 1000. In comparison, for the same data set on a SunFire 6800 we found it takes 0.02, 0.054, and 31 seconds using ESTmapper, BLAT, and sim4.

### 2.2. EST Clustering

Clustering is usually the first step in using ESTs for gene finding. Historically clustering algorithms compare ESTs against each other to form clusters. NCBI also maintains UniGene, a reference list of EST clusters automatically generated from ESTs in dbEST [28].

*TGICL* (TIGR Gene Indices CLustering tools) is an example of a popular software system for clustering large EST data sets using pairwise comparisons [23]. TGICL uses mgBLAST, a modified version of megaBLAST that provides additional output filtering and uses a dynamic offset within a database for incremental searches. MgBLAST is used to quickly perform an all-to-all pairwise comparisons between EST sequences. Processing can be performed in parallel by partitioning the database, then merging compressed sorted files.

Clustering uses a greedy algorithm based on the best alignments, and known full-length cDNAs can be used as seeds to improve efficiency and produce larger clusters for incremental updates. Clusters output are passed to the CAP3 assembly tool [12] as multi-FASTA files and then assembled into high-quality consensus sequences. TGICL is used to generate the TIGR Gene Indices for 60 different species with between 10 thousand and 4 million EST sequences [17]. TGICL has been parallelized for PC clusters using PVM.

*PaCE* (Parallel Clustering of ESTs) is one of the first clustering tools designed to exploit the power of suffix trees to avoid the need for all-to-all pairwise comparisons [13]. It first constructs (in parallel) a generalized suffix tree consisting of all EST sequences by first sorting ESTs and sending each to the appropriate processor. Once the suffix tree is complete, a variation of the algorithm for finding longest repeated substrings can be used to find all pairs of ESTs with common substrings above a certain threshold. The clustering algorithm then starts with pairwise comparisons between

ESTs with long common substrings, greatly increasing the likelihood of forming a cluster.

By using an on-demand algorithm for generating promising EST pairs, the PaCE approach generally requires only $O(n)$ number of pairwise comparisons, though $O(n^2)$ are still needed in the worst case. Additional refinements are needed to create and maintain clusters in parallel. Clusters produced by PaCE are of high quality when compared to a benchmark EST set from Arabidopsis created by aligning ESTs directly to the genome. A weakness of the PaCE system is that building a suffix tree for the input EST data set requires a large amount of memory, proportional to the number and size of ESTs. The authors were able ameliorate this problem by parallelizing their algorithm to run on a PC cluster, splitting the suffix tree so that each node only needs to hold a portion of the suffix tree in memory.

### 2.3. WOTD Suffix Trees

A suffix-tree is a data-structure that allows many problems on strings (sequences of characters) to be solved quickly and efficiently [11]. It is formed by calculating and storing all the suffixes of a string in a *trie* structure. Suffix trees are very efficient data structures. They can be constructed in $O(n)$ time, and finding longest common substrings between two sequences and longest repeated substrings only require $O(n)$ time. Unfortunately, there is a large (30+) multiplicative factor in the size of the suffix tree relative to the original sequence.

More recently, researchers have developed efficient implementations for building *write-only, top-down* (WOTD) suffix trees [9]. WOTD suffix trees are more expensive to build in that they require $O(nlog(n))$ expected and $(O(n^2))$ worst time to construct, but they require only a 10 to 12-fold increase in memory relative to the original sequence, and display good cache locality in performing searches. WOTD suffix tree implementations may be *eager* (build the full WOTD tree immediately) or *lazy* (build portions of the WOTD tree as needed as queries are processed).

## 3. ESTmapper

The design of ESTmapper is based on two observations. First, the genome of an organism is relatively fixed and rarely updated. As a result we can preprocess the genome once and reuse the result to perform many mappings to the genome. Second, quality alignments between DNA sequences and the genome are likely to have long common substrings of exact matches, and longer common substrings are likely to yield better mappings. Together these two observations inspired us to use suffix tree as the underlying technique for ESTmapper, since the suffix tree for the genome can be built once and used to efficiently find long common substrings.

### 3.1. Algorithm

The ESTmapper algorithm consists of the following steps:

**1) Preprocess the genome.** The genome sequence is read from file and converted into a eager WOTD suffix tree. If the suffix tree is too large to fit into memory, the genome can be partitioned and one suffix tree built for each partition to reduce memory usage. Preprocessing the genome only needs to take place once per genome, since the eager WOTD suffix tree is a flat array and can be easily stored and reloaded for future mappings.

**2) Map DNA sequences to the genome.** Each DNA sequence is mapped using the suffix tree (or trees) for the genome, considering both its minus and plus strand. If the genome has been partitioned into multiple suffix trees, the following mapping steps must be repeated for each tree, storing the list of mapping results and combining them at the end, to find the best overall mapping. Mappings are computed as follows:

**2a) Find long common strings.** All suffixes of each DNA sequence are compared to the suffix tree for the genome (or its current subset, if genome is partitioned) to find the long common strings above a user-specified minimal length. Long common substring lengths and locations on both DNA and genome sequence are stored for next step processing. The process can be accelerated by only considering every one out of $k$ suffixes with negligible loss of precision.

**2b) Extend long common substrings.** Each pair of long common substrings found is extended in both directions, with a default match reward of 1, mismatch penalty of $-3$ and dropoff value of $-11$. This substring extension step is similar to NCBI BLASTN's match extension step, except ESTmapper's extension is based on long common substrings instead of matched k-mers. The extended substrings are then sorted in order of location.

**2c) Build (spliced/gapped) mappings.** To handle splicing and gaps, ESTmapper examines the list of long substrings and locations, and combines them into a single spliced/gapped *mapping region* if two substrings are mapped sufficiently close to each other on the genome. Building mappings can be done quickly since the extended substrings are stored in order of locations. The lengths and locations of each mapping region's component substrings are stored for the refinement step.

**2d) Refine mappings.** ESTmapper examines neighboring substrings in each mapping to determine whether they are close enough on both the DNA and genomic sequences to be merged into a single alignment (corresponding to a single exon), with only small gaps and/or mismatches. Boundaries of substrings are then adjusted so they match splice donor and acceptor sites, minimizing changes to the alignment as much as possible.

**3) Choose best mappings.** Finally, each mapping is scored based on the selected match reward, mismatch and gap penalties. An expected E-value for the entire mapping is calculated using alignment statics [1, 14], and used to select the best mapping for each DNA sequence to the genome.

### 3.2. Parallelization

Being able to exploit the power of parallel computing is a major advantage for computationally intensive applications such as mapping large number of DNA sequences onto genomes. It turns out that the ESTmapper algorithm, like many problems in computational biology, is embarrassingly parallel and can be easily parallelized at a coarse-grain level for efficient parallel execution. The parallel versions of ESTmapper work as follows.

*Multithreaded ESTmapper.* On shared-memory multiprocessors (SMPs) ESTmapper may be parallelized according to the master-worker parallel paradigm using pthreads. The master thread first reads the suffix tree of the genome into memory sequentially. Next, the master thread spawns workers (pthreads) and assigns DNA sequences to each worker to find mappings. When all sequences have been mapped to the genome, the master thread collects and outputs the result.

Load balancing is simple since the bulk of the computation is mapping DNA sequences to the genome, and the mapping time is usually proportional to the number of bases being mapped. So the master thread just needs to assign roughly the same number of bases to each thread to ensure an even division of work.

*Message-passing ESTmapper.* On message-passing machines such as PC clusters, ESTmapper may be parallelized using the MPI communications library to communicate between processors. To reduce communication costs, each processor reads the suffix tree into memory independently. The master node assigns and sends DNA sequences each node. Every node then finds genome mappings for its sequences locally, without any need for interprocessor communication. Once mappings are computed for assigned sequences, each node sends its mappings to the master node to be collected and output.

ESTmapper is very efficient on clusters, since little communication is needed, just at the beginning and end of the overall computation to send out DNA sequences and retrieve their mappings. Performance can be improved further if each node stores a copy of the suffix tree in its local disk.

## 4. Experimental Evaluation

### 4.1. Evaluation Environment

To evaluate ESTmapper, we compared its performance and precision with other mapping and clustering algorithms. We downloaded and installed the latest versions of BLAT, Sim4, Spidey, TGICL, PaCE, and the NCBI BLAST software suite and toolkit on a Sun SunFire 6800 SMP with 24 processors (UltraSparc III 750Mhz) and 72 GB memory. For message-passing speedups we also ran ESTmapper on a Linux PC cluster with 1.6 GHz AMD Athlon processors and 1 GB memory per node.

For test data, we downloaded EST, gene, and genome sequences for Arabidopsis Thaliana (mustard plant) and Homo Sapiens (human) from NCBI, UCSC genome browser and Trust Sanger Institute. The Arabidopsis genome is about 19.7 million bases, and the Human genome is about 3 billion bases. The EST and gene DNA sequences we use were chosen because they were previously mapped to the genome by biologists (or put in widely accepted clusters). We can thus used them to evaluate the precision of mapping and clustering algorithms used by ESTmapper and other tools. Three data sets used in many of our experiments are:

- DataSet1: 263,255 EST and cDNA sequences (average length 734 bases) from Arabidopsis UniGene build #44 from NCBI
- DataSet2: 28,952 Arabidopsis genes (average length 1322 bases) and 5 Arabidopsis chromosomes from NCBI.
- DataSet3: 936 Human genes (average length 1936 bases) mapped to chromosome 22 and Build #30 of Human chromosome 22 from Trust Sanger Institute.

### 4.2. Basic Performance

We begin by examining the time and memory required by ESTmapper with respect to suffix trees and multiple processors. WOTD suffix trees can be more expensive to build than other suffix trees in that they require $O(nlog(n))$ expected and $O(n^2)$ worst time to construct. However, they can yield good performance for substring searches, since the WOTD data structures provide good cache locality. Since the WOTD suffix tree is stored in a flat array, users can also build the suffix tree once and store it on disk for later use.

When constructing a WOTD suffix tree for the genome, we need to select the number of trees we plan to use. ESTmapper can build a single suffix tree for the entire genome (by concatenating the DNA sequence of each chromosome), one tree per chromosome, or any number of trees (by first concatenating, then splitting the DNA sequences of all chromosomes). Some additional bookkeeping is required to keep track of the original chromosomal locations of each sequence. When the genome is partitioned into multiple suffix trees, ESTmapper can iteratively compare nucleotide sequences against each suffix tree, record the mappings found, and select the best overall mapping at the end.

Figure 1a shows the result of using multiple suffix trees to map Arabidopsis ESTs from DataSet1 to the entire genome. We see ESTmapper runs faster with fewer suffix trees, but trees are larger and require more memory. We thus have a classic memory/speed trade-off in choosing how to use ESTmapper. Choosing the proper number of trees to use depends on the available memory. If insufficient memory is available to hold the suffix tree for the entire genome, ESTmapper can reduce its memory usage at the expense of longer running times by partitioning the genome and building separate suffix trees for each portion of the genome.

Figure 1b shows the performance of multithreaded ESTmapper on the SunFire 6800 SMP and message-passing ESTmapper on a Linux PC cluster when mapping DataSet1 to the entire Arabidopsis genome. We see that ESTmapper obtains fairly good 8-processor speedup for the SMP (6.9) and PC cluster (5.7).

We believe speedups are lower on the PC cluster because (sequential) file I/O to load suffix trees is both slower (44 seconds on PC cluster, versus 14 seconds on SMP), and takes up a larger portion of total execution time on the faster PCs (186 ESTs mapped/second on PC nodes, versus 83 ESTs mapped/second on the SMP). If the file I/O time is eliminated, ESTmapper achieves a 8-processor speedup of 7.5 on the SMP and 8.0 on the PC cluster. We thus expect ESTmapper speedups to improve for larger input data sets, where file I/O is less important. Memory use remains constant for ESTmapper relative to the number of processors, since each processor maintains a local copy of the suffix tree.

## 4.3. Genome Mapping

*Performance Comparison.* The performance of six DNA-to-genome alignment tools (ESTmapper, BLAT, Sim4, Spidey, BLAST and megaBLAST) were evaluated with Arabidopsis and Human data. We mapped 936 Arabidopsis genes from DataSet2 onto chromosome I, and 936 Human genes in DataSet3 onto chromosome 22. All tools were run sequentially on a single processor of the SunFire 6800. Running times are shown in Figure 1c. “ESTmap” stands for ESTmapper and “megaB” stands for megaBLAST. Note that running times are presented using log scale along the Y-axis.

We see that ESTmapper is the fastest among six evaluated software tools, and has reasonable memory usage. It is about 3–6 times as fast as BLAT, 21–45 times as fast as megaBLAST, 1000 times as fast as Spidey and 4000 times faster than Sim4. These results seem reasonable when we remember that only ESTmapper and BLAT preprocess the genome to improve performance. ESTmapper is likely more efficient than BLAT because it is able to find the longest common substrings directly, rather than extending hits.

Figure 1d presents the memory used by each of the alignment tools. We see megaBLAST and BLAST use the most memory, while BLAT requires the least memory. ESTmapper requires about 4 times more memory than BLAT.

*Precision Comparison.* Next, we compared the precision of the DNA-to-genome mappings found by four tools (ESTmapper, BLAT, Sim4, Spidey) for all 28,952 Arabidopsis genes and 936 Human genes from DataSet2 and DataSet3. We used the Arabidopsis gene annotation information provided by biologists at TIGR and Human gene annotations from biologists at Trust Sanger Institute as the correct mapping. Results are shown in Table 1.

|  | Spidey | BLAT | Sim4 | ESTmapper |
|---|---|---|---|---|
| Arabidopsis | 96.0% | 99.6% | 94.3% | 96.2% |
| Human | 90.8% | 89.7% | 99.6% | 99.9% |

**Table 1. Precision Comparison with 28,952 Arabidopsis and 936 Human Genes**

We see that BLAT is the most precise for Arabidopsis (99% correct), but ESTmapper and Spidey are close behind (96% correct). For Human ESTmapper and sim4 are the most precise (almost 100% correct), while BLAT and Spidey are lagging (90% correct). The only Human gene ESTmapper mapped incorrectly it missed because there is one exon which is too small to be found with the default ESTmapper settings. Most such small exons can be found by ESTmapper with a finer grain search, at the cost of reduced processing speed. Fortunately, very small exons usually do not occur often in large genomes [6, 25].

## 4.4. EST Clustering

Next, we compared the performance and precision of using genome mappings to cluster ESTs against other clustering algorithms such as PaCE and TGICL. We form EST clusters out of all ESTs mapped to nearby or overlapping locations in the genome. For the comparison, we used genome mappings from ESTmapper, Spidey and BLAT.

*Performance Comparison.* In Figure 1e, we compared ESTmapper performance with TGICL and PaCE for 190,740 Arabidopsis ESTs mapped to the Arabidopsis genome preprocessed as a single suffix tree. All three

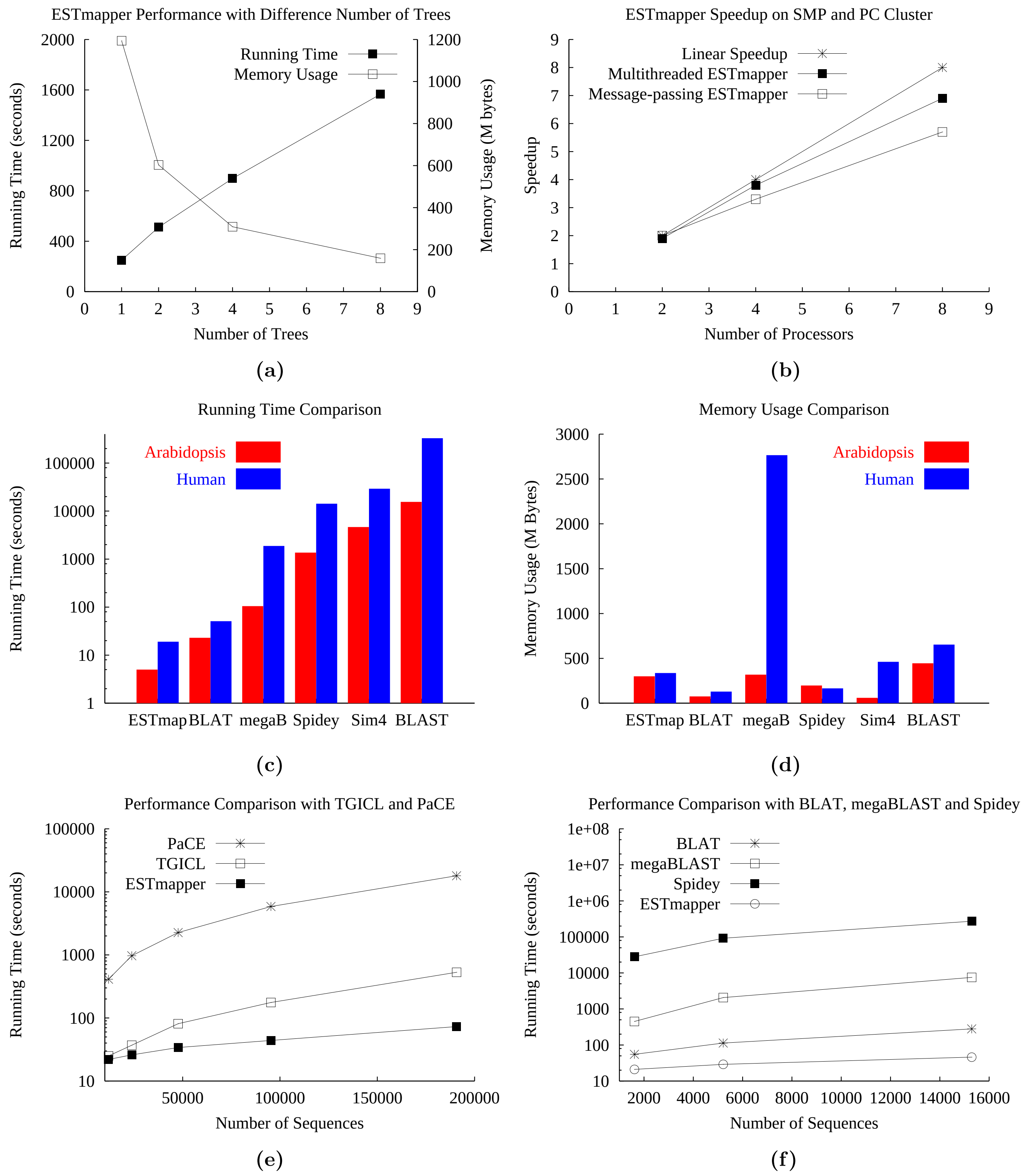

**Figure 1. Performance Comparison of EST Mapping and Clustering Algorithms**

tools were timed using 8 processors on the SunFire 6800. Note that running time is displayed in log scale along the Y-axis. We see that ESTmapper is significantly faster for large numbers of ESTs. Figure 1f shows the performance of BLAT, Spidey, and megaBLAST when used to cluster ESTs using one processor on the SunFire 6800. Due to the slow processing speed of Spidey and high memory usage of megaBLAST, for the second set of experiment we were only able to use 15,293 EST sequences. We see once again that ESTmapper is the most efficient clustering algorithm.

We also found ESTmapper has almost constant memory usage of about 1 GB. In comparison, memory usage for PaCE and megaBLAST increases very quickly as the number of ESTs increases. PaCE used 1.4 GB memory on each processor when processing 190,740 sequences. megaBLAST used about 4 GB memory when processing 15,293 sequences and ran out of memory when the number of ESTs was increased.

*Precision Comparison.* To compare the precision of the clusters produced by the different algorithms, we used BLAT, Spidey, PaCE, TGICL and ESTmapper to cluster ESTs from Arabidopsis UniGene build #44 in DataSet1. Because of the memory limitation with PaCE and speed limitation with Spidey, we only used the 15,293 EST sequences from the first 1000 Arabidopsis clusters in UniGene. We measured the percentage of clusters exactly matching UniGene, the number of clusters produced by each algorithm, and the number of singleton clusters (with a single EST). The results are shown in Table 2.

| | Spidey | BLAT | TGICL | PaCE | ESTm. |
|---|---|---|---|---|---|
| % identical | 80.5% | 81.4% | 72.6% | 60.5% | 97.1% |
| # clusters | 930 | 1152 | 1006 | 1575 | 1006 |
| # singletons | – | 101 | 296 | 573 | 9 |

**Table 2. Precision Comparison with 1000 Arabidopsis UniGene Clusters**

Again, ESTmapper produced clusters that are most similar (97%) to UniGene clusters. The other clustering techniques based on mapping ESTs to the genome were next (80%), while EST-only techniques produced significantly different clusters. ESTmapper also produced the smallest number of singletons clusters when compared to other algorithms. In addition to the results in Table 2, we also have clustering comparisons results for TGICL, BLAT and ESTmapper using the total 20640 Arabidopsis UniGene clusters. Among the three algorithms, TGICL found 53.8% exactly matched clusters, BLAT found 66.5% exactly matched clusters, and ESTmapper found 83.6% exactly matched clusters. Though all three algorithms are less precise for the full set of Arabidopsis UniGene clusters, ESTmapper still provides the results closest to UniGene.

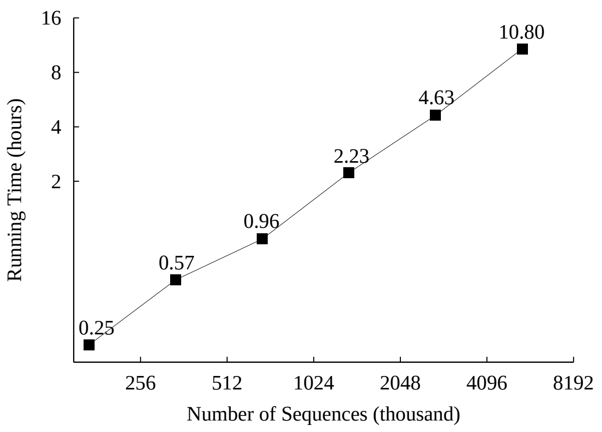

**Figure 2. Clustering $5.5 \times 10^6$ Human ESTs**

*Scalability.* Finally, to evaluate the scalability of ESTmapper, we also measured the computation time and memory usage of ESTmapper when used (on a SunFire 6800 with 8 processors) to cluster all 5.5 million human EST sequences by mapping them against the human genome (Build 35). To reduce memory use, the genome was split into 30 equal-sized pieces requiring about 1 GB suffix trees for each piece. Results are shown in Figure 2. ESTmapper was able to cluster all ESTs in 10.8 hours (0.056 seconds per EST for each processor), with the processing time increasing fairly linearly with the number of ESTs. Performance can be improved by using more memory and fewer trees.

## 4.5. Discussion

Our experimental evaluation of ESTmapper shows it is quite efficient and precise when compared with other sequence alignment techniques such as BLAT. The only disadvantage is that ESTmapper requires more memory and disk storage to hold and store the suffix tree needed for the genome. Fortunately the amount of memory and disk space available in computers is quickly increasing while genome size stays constant, so memory use should become less of an issue as time goes on. For the purpose of EST clustering, ESTmapper also performed very well compared to other clustering algorithms. The main disadvantage here is that ESTmapper can only be used to cluster ESTs for organisms with sequenced genomes. As a result ESTmapper will probably serve simply to complement tools such as TGICL and PaCE.

## 5. Conclusions

In this paper, we presented ESTmapper, a new approach for efficiently aligning DNA sequences to genomes, and using such alignments to cluster ESTs. We developed a prototype implementation and compared its precision and performance to existing alignment and clustering tools. Though a number of weaknesses remain in our approach, preliminary results have been very encouraging. Our work on ESTmapper is a part of our overall goal—taking advantage of increasing computation power to provide more useful information to bioinformatics researchers.

## 6. Acknowledgments

The authors are very grateful to Jim Kent for assistance with selecting parameters for improving BLAT performance, and Anantharaman Kalyanaraman for assistance with using and interpreting results from PaCE. We wish to thank Damayanti Gupta for obtaining some preliminary results comparing the precision of clustering algorithms, and students in CMSC838T in Spring 2004 for help in evaluating alternative DNA mapping methods.

## References

[1] S. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–80, 1996.

[2] S. Altschul, W. Gish, E. Miller, E. Myers, and D. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] R. Chen, A. Russell, G. Li, N. Tsinoremas, and G. Cavet. Human transcript clustering. *Poster at RECOMB'04*, 2004.

[4] E. Coward, S. Haas, and M. Vingron. SpliceNest: visualization of gene structure and alternative splicing based on EST clusters. *Trends Genet*, 18(1):53–55, 2002.

[5] A. Delcher, S. Kasif, R. Fleischmann, J. Peterson, O. White, and S. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[6] M. Deutsch and M. Long. Intron-exon structures of eukaryotic model organisms. *Nucleic Acids Research*, 27(15):3219–28, Aug 1999.

[7] L. Florea, G. Hartzell, Z. Zhang, G. Rubin, and W. Miller. A computer program for aligning a cdna sequence with a genomic dna sequence. *Genome Research*, 8(9):967–74, Sep 1998.

[8] M. Gelfand, A. Mironov, and P. Pevzner. Spliced alignment: A new approach to gene recognition. *Proc. Natl. Acad. Sci.*, 93:9061–9066, 1966.

[9] R. Giegerih, S. Jurtz, and J. Stoye. Efficient implementation of lazy suffix trees. *Software—Practice ad Experience*, 33:1035–1049, 2032.

[10] C. Grasso, B. Modrek, Y. Xing, and C. Lee. Genome-wide detection of alternative splicing in expressed sequences using partial order multiple sequence alignment graphs. *Pacific Symposium of Biocomputing*, 9:29–41, 2004.

[11] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1977.

[12] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Res*, 9:868–877, 1999.

[13] A. Kalyanaraman, S. Aluru, S. Kthari, and V. Brendel. Efficient clustering of large EST data sets on parallel computers. *Nucleic Acids Research*, 31(11):2963–2974, 2003.

[14] S. Karlin and S. Altschul. Applications and statistics for multiple high-scoring segments in molecular sequences. *Proc Natl Acad Sci U S A*, 90(12):5873–7, Jun 1993.

[15] W. Kent. Blat–the blast-like alignment tool. *Genome Research*, 12(4):656–64, Apr 2002.

[16] S. Kurtz et al. Versatile and open software for comparing large genomes. *Genome Biology*, 5, 2004.

[17] F. Liang, I. Holt, G. Pertea, S. Karamycheva, S. Salzberg, and J. Quackenbush. An optimized protocol for analysis of est sequences. *Nucleic Acids Research*, 28:3657–3665, 2000.

[18] B. Lin and T. Burcham. Using the human genome as a framework for sequence clustering and microarray design. *Poster at RECOMB'04*, 2004.

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

[20] C. Meek, J. Patel, and S. Kasetty. OASIS: An online and accurate technique for local-alignment searches on biological databases. In *Proceedings of the Conference on Very Large Databases (VLDB'03)*, Berlin, Germany, Sept. 2003.

[21] R. Mott. EST_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA. *Computer Applications in the Biosciences*, 13(4):477–478, 1997.

[22] J. Ogasawara and S. Morishita. Fast and sensitive algorithm for aligning ESTs to human genome. In *IEEE Computer Society Bioinformatics Conference (CSB'02)*, Stanford, CA, Aug. 2002.

[23] G. Pertea, X. Huang, F. Liang, V. Antonescu, R. Sultana, S. Karamycheva, Y. Lee, J. White, F. Cheung, B. Parvizi, J. Tsai, and J. Quackenbush. TIGR Gene Indices clustering tools (TGICL): a software system for fast clustering of large EST datasets. *Bioinformatics*, 19(5):651–652, 2003.

[24] T. Pohar, H. Sun, S. Liyanarachchi, S. James, S. Stapleton, and R. Davuluri. A bioinformatics approach toward identification of genes involved in hematopoiesis and leukemia. *Poster at RECOMB'04*, 2004.

[25] M. Sakharkar, V. Chow, and P. Kangueane. Distributions of exons and introns in the human genome. *In Silico Biology*, 4:32, 2004.

[26] A. Sczyrba, J. Krüger, and R. Giegerich. e2g - a web-based tool for effciently aligning genomic sequence to EST and cDNA data. *Poster at RECOMB'04*, 2004.

[27] S. Wheelan, D. Church, and J. Ostell. Spidey: a tool for mrna-to-genomic alignments. *Genome Research*, 11(11):1952–7, Nov 2001.

[28] D. Wheeler et al. Database resources of the national center for biotechnology. *Nucleic Acids Research*, 31:28–33, 2003.

[29] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology*, 7:203–214, 2000.