ESTmapper: Efficiently Clustering EST Sequences Using Genome Maps

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

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

Abstract

Expressed sequence tags (ESTs) are short transcribed nucleotide sequences that can be used to discover new genes and measuring gene expression. Because individual ESTs are short and error-prone, ESTs must first be *clustered* to be useful. In this paper, we describe ESTmapper, a new tool for clustering EST sequences based on efficiently mapping ESTs to the genome. Our mapping algorithm is based on first building an eager write-only top-down (WOTD) suffix tree for the genome, then searching for long common substrings between each EST and the genome to build *matching regions*, gapped local alignments between the EST and genome that account for sequencing errors and splicing. Long matching regions are then used to map ESTs to the genome and place ESTs into clusters based on location. Preliminary experimental evaluation shows that though ESTmapper requires a large amount of initial memory to store the genome suffix tree, it is quite precise and more efficient than previous techniques such as TGICL and PaCE when clustering large numbers of ESTs.

1 Introduction

Recent advances in molecular biology techniques such as automated DNA sequencing and DNA microarrays allow scientists to quickly gather huge amounts of gene sequence and expression data. Much of this data is publicly available in collections of large sequence databases at institutions such as NCBI GenBank. The need to use computers to analyze and extract useful information from this wealth of data has led to the development of the area of research known as *bioinformatics*. An important area of bioinformatics relates to the analysis of *Expressed Sequence Tags* (ESTs), single-pass cDNA sequences of transcribed genomic DNA that provide important information about gene discovery and gene expression. ESTs are important because they provide important clues as to the locations of the small percentage (1-2%) of DNA that is actually transcribed into mRNA, especially when the complete transcribed genes are too expensive or difficult to sequence.

Because they are relatively easy to collect and sequence, ESTs are an important source of biological information. For instance, ESTs represent a large part of the information collected in bioinformatic databases, making up 63% of the 29.8 million nucleotide sequences in the October 2003 version of NCBI GenBank (release 138) [1]. In fact, ESTs are so popular NCBI maintains dbEST, a separate sequence database of over 9.6 billion nucleotides from over 19 million EST sequences. dbEST contains over 5.4 million human and 3.9 million mouse ESTs, as well as additional ESTs from another 600 organisms.

One weakness of ESTs is that because they represent large numbers of incomplete, error-prone, and redundant fragments of genes, several EST sequences must be combined in a *cluster* of overlapping EST sequences before they can be processed by assembly tools to discover the full sequence for each gene. Clustering such large collections of ESTs is very memory and computationally intensive, and many algorithms and tools have been developed. The resulting EST clusters have so many uses (gene discovery, gene expression studies, microarray probe set design) that NCBI maintains and continually updates UniGene, a database of automatically generated EST clusters.

[1] But because EST sequences are relatively short, they comprise only 27% of the total number of nucleotides (bases) in GenBank.

Many earlier EST clustering algorithms (TGICL, d2_cluster, GeneNest, CLOBB) rely on performing pairwise comparisons between ESTs, with closely matching pairs put into a single cluster. These algorithms can work well for small collections of ESTs, but obviously do not scale well because of the $O(n^2)$ number of pairwise comparisons needed. More recent approachs (PaCE, Xsact) build a *suffix tree* of all ESTs to identify pairs of ESTs with long common substrings. The number of pairwise comparisons is reduced by first clustering these promising pairs of ESTs, but memory usage is greatly increased by the need to build a suffix tree from the ESTs.

With the advent of high-throughput sequencing of the entire genomes of many species, a new alternative approach becomes possible. Instead of clustering ESTs by comparing ESTs to each other, we can directly compare and map the EST to the genome instead. This approach has the advantage of linear-time memory and computation use with respect to the input EST sequences. Mapping EST sequences to the genome can also yield very precise clusters, particularly for high-quality genome sequences such as the April 2003 release of the human genome, which is 99% complete with an accuracy rate of over 99.99%, and is highly contiguous (with contiguous sections averaging over 27 million bases long). One disadvantage is that the genome for the organism must be available. Memory use is now also dependent on the genome size (computation time can be reduced since genomes can be preprocessed once for all EST searches).

In this paper, we describe ESTmapper, a new tool for clustering EST sequences based on efficiently mapping ESTs to the genome. Our mapping algorithm first builds an eager *write-only, top-down suffix tree* for the genome, then searches for long common substrings between each EST and the genome. We use them to build gapped matching regions that account for sequencing errors and splicing, and use the longest overall matching region to map the EST to the genome. ESTs mapped to overlapping and/or nearby locations in the genome are placed in a cluster, and can be used by other tools. Preliminary experiments show that ESTmapper is precise and very efficient for clustering large numbers of ESTs when compared to other popular EST clustering tools. In summary, the main contributions of this paper are:

- Design and implement both sequential and parallel versions of ESTmapper, a new EST clustering technique based on mapping ESTs to the genome.
- Experimental comparison of precision, memory use, and computation requirements of ESTmapper with other EST clustering tools (TGICL, PaCE).
- Evaluating the precision of clusters produced by ESTmapper relative to other techniques.

In the remainder of the paper, we begin by describing two approaches used by previous EST clustering tools. We present the ESTmapper algorithm, and perform an experimental evaluation based on results from a prototype implementation. We compare with related work and conclude.

2 Background

Pairwise Comparisons Most EST clustering algorithms are based on pairwise comparisons, where pairs of ESTs in different clusters are compared against one another. If the two ESTs are found to be highly similar in some fashion, the two clusters are joined. The process continues until no more clusters can be joined. Different metrics may be used for performing pairwise comparisons, but most tools are based on BLAST, a popular pairwise alignment algorithm. Pairwise comparison approaches are efficient for small inputs, but are not scalable to large numbers of ESTs because the number of comparisons tend to grow quadratically relative to the number of ESTs (i.e., pairwise comparisons is an $O(n^2)$ algorithm, where n is the number of ESTs). The number of pairwise comparisons can be reduced by using transitive closure to join clusters, but may still require a full all-to-all comparison in the worst case.

TGICL The TIGR Gene Indices CLustering tools (TGICL) [PHL+03]. is an example of a popular software system for clustering large EST data sets using pairwise comparisons. 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 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 [LHP+00]. TGICL has been parallelized for PC clusters using PVM.

Suffix Trees The prohibitive expense of all-to-all pairwise comparisons has been a well-recognized problem for EST clustering algorithms. Researchers have recently tried improve efficiency by using *suffix trees* to reduce the comparison cost. A suffix-tree is a data-structure that allows many problems on strings (sequences of characters) to be solved quickly and efficiently [Gus77]. 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 linear time, and finding longest common substrings between two sequences and longest repeated substrings only require linear time. Unfortunately, there is a large multiplicative factor in the size of the suffix tree relative to the original sequence. Even efficient suffix tree implementations can suffer a 30-fold increase in memory relative to the original sequence.

More recently, researchers have developed efficient implementations for building *write-only, top-down* (WOTD) suffix trees [GJS32]. 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).

PaCE PaCE (Parallel Clustering of ESTs) is one of the first software systems designed to exploit the power of suffix trees to avoid the need for all-to-all pairwise comparisons [KAKB03]. It first constructs (in parallel) a generalized suffix tree consisting of all the EST sequences. To construct the suffix tree in parallel, an suffixes are first sorted and sent 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 can then start 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 these promising pairs of ESTs, 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 the building a suffix tree for the input EST data set requires a large amount of memory. The authors were able ameliorate this problem by parallelizing their algorithm to run on multiple computer, splitting the suffix tree so that each computer only needs to hold a portion of the suffix tree in memory. Using this approach the authors were able to cluster up to 420,000 ESTs on a 30-node dual-processor IBM Pentium PC cluster, with 2GB memory per node. The authors estimate that 512 nodes with 0.5GB memory each would be required to cluster the 5 million human ESTs in GenBank.

3 ESTmapper

3.1 Advantages

As we have seen, suffix-tree based approaches to EST clustering like PaCE are promising, but have a major weakness in that they require large amounts of memory to process large numbers of ESTs. The key insight motivating our approach for ESTmapper is that instead of building a suffix tree for the input EST data set, we can *build a suffix tree for the genome, and efficiently map the ESTs to the genome instead*. This approach has a number of major advantages:

Linear Computation Time The algorithm is very efficient since ESTs may be compared to a suffix tree in linear time for very large numbers of ESTs, even in the worst case.

Preprocess Suffix Tree Another advantage is that while the number of ESTs continues to grow, the size and content of the genome is fixed for each organism. As a result the suffix tree for each genome can be generated just once and reused for each clustering.

Incremental Updates ESTmapper is also well-suited to incrementally updating clusters as new ESTs are added, since the mapping of existing ESTs remains unchanged. For an incremental update only the new ESTs need to be mapped to the genome, and the mappings may be compared with older mappings to calculate clusters.

Improved Precision In addition, clusters formed by mapping ESTs to the genome are also likely to be more accurate than clustering based on comparing ESTs alone. Researchers for other tools have already claimed better clustering results by including genomic or full-length cDNA sequences. Developers for PaCE even evaluated the precision of their clusters by comparing against a set of ESTs clustered by their locations in the genome. In fact, the default parameters for ESTmapper were chosen to mirror those applied by hand by biologists for the EST benchmark used by the PaCE developers.

Another advantage is that ESTs mapped to nearby locations in the genome may be put into the same cluster, even if they do not have significant (or any) overlap. Such clusters are obviously undetectable by approaches which compare ESTs to each other.

3.2 Disadvantages

In choosing to use mapping to the genome as the key approach in ESTmapper, we need to face several issues.

Genome Availability A major disadvantage of ESTmapper is that it depends on having the genome sequence available to use as a map for clustering ESTs. As more organisms are sequenced, this limitation should decrease. Currently over half of all the ESTs in GenBank are from organisms with sequenced genomes. In fact, ESTs from just the top three species with sequenced genomes (human, mouse, rat) account for 52% of all ESTs in GenBank. Even if the complete genome is not available, even having partial genomic survey sequences for an organism will probably be sufficient for mapping many of its ESTs.

Memory Usage One weakness of ESTmapper is that a large amount of memory is needed to hold the suffix tree for the entire genome. To overcome this problem, ESTmapper can reduce its memory usage by partitioning the genome and building separate suffix trees for each portion of the genome. Fortunately selecting partitions is easy since nature has already partitioned large genomes into discrete sequences known as chromosomes. ESTmapper then iteratively compares ESTs against each of the suffix trees, recording the mappings found and selecting the best overall mapping at the end. We expect memory usage to be less of an issue in the future since memory sizes continue to grow, while genome sizes remain constant.

Gapped Alignments Doing pairwise comparisons with algorithms such as BLAST automatically accounts for the fact that mappings between ESTs and the genome are not exact due to mutations, splicing, and sequencing errors. In fact, because ESTs are single-read sequences, they are known to contain a relatively high percentage (2-3%) of errors, particularly at sequence endpoints. As a result it is rare to find large portions of ESTs to be exact substrings of the genome. ESTmapper cannot thus simply find the longest common substring between the EST and genome, but instead uses common substrings as a starting point for building gapped alignments to account for sequencing errors, substitutions, indels (insertions/deletions), and splicing.

3.3 Algorithm

The algorithm used by ESTmapper consists of the following steps.

1. Preprocess the genome

   The genome sequence is read from file and converted into a eager WOTD suffix tree. 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 for later use.

2. Map ESTs to the genome

   Each ESTs is mapped using the suffix tree for the genome. If the genome has been partitioned into multiple suffix trees to reduce memory use, the list of mappings need to be stored and combined at the end, and the first two steps (finding common substrings, building matching regions) must be repeated for each suffix tree to find the best overall mapping. Mappings are computed as follows:

   (a) Find long common substrings
   All suffixes of each EST are compared to the suffix tree for the genome (or its current subset, if genome is partitioned) to find the long common substrings above a user-specified minimal length. Common substrings and their locations are stored in order of their location. The process can be accelerated by skipping substrings known to be shorter than minimal length. ESTmapper finds all common substrings above a certain length (rather than the longest common substring) to allow for a small number of gaps in local alignments between ESTs and the genome. Common substrings are stored in order of their location in the genome.

   (b) Build (gapped) matching regions
   To handle splicing, and because ESTs are single-read sequences that are error-prone, the ESTmapper algorithm performs some extra computation to combine nearby common substrings. ESTmapper examines the list of long common substrings and locations, and combines substrings into a single gapped *mapping region* if two common substrings are sufficiently close together when mapped onto the genome. Building matching regions can be done quickly since common substrings are stored in order of their locations. The longest matching region is used to determine the mapping of the entire EST to a location in the genome.

3. Build clusters

   Once all ESTs have been mapped to a location in the genome, their locations can be used to very precisely map ESTs to clusters based on overlap and nearness inthe genome. Two ESTs are clustered if their sequences overlap or are co-located by at least a user-specified minimal cluster distance.

The full algorithm used by ESTmapper is presented in Figure 1.

3.4 Parallelization

Being able to exploit the power of parallel computing is a major advantage for computationally intensive applications such as clustering ESTs. It turns out that the ESTmapper algorithm, like many problems in computational biology, is embarrassingly parallel and easily parallelized at a coarse-grain level for efficient parallel execution. The parallel versions of ESTmapper works as follows.

Shared-memory version (Pthreads) ESTmapper may be parallelized according to the master-worker parallel paradigm using *pthreads* on shared-memory multiprocessors (SMPs). The master thread performs all the setup, including reading the suffix tree of the genome into memory. It then assigns ESTs to workers (pthreads) to calculate the mapping for each EST. When all ESTs have been mapped to the genome, the master thread merges mappings and calculates clusters.

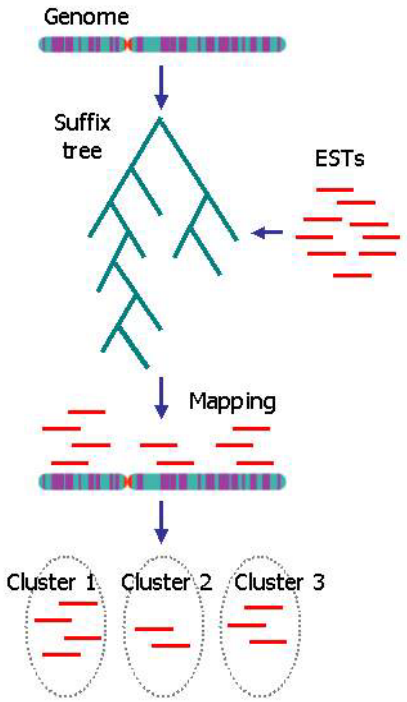
Load balancing is simple since the bulk of the computation is mapping ESTs to the genome, and the mapping time is proportional to the number of ESTs being mapped (since ESTs are all roughly the same

```
  // preprocess genome
  for each (subset) genome sequence
    build & store suffix tree T

  // cluster ESTs
  for each (subset) genome (if genome is partition
  {
    for each EST data set
    {
      load suffix tree T

      // find long common substrings and
      //   build gapped matching regions
      for each EST E
      {
        for each EST suffix
          find long common substrings S for suffix
          if length of S >= MIN_MATCH_SUBSTRING
             store S & its location in T
        for all S
          examine locations of S1, S2
          if S1 near S2 with gap G <= MAX_GAP
            form/extend matching region M for S1,S
        find longest M
        if length of M >= MIN_MATCH_REGION
          map E to location of M in genome
        }
    }
    // build clusters from mapped ESTs
    for each mapped EST in order of its location
      if ESTs to previous EST <= MIN_CLUSTER_DISTANCE
        place ESTs in same cluster

    output ESTs and their locations in each cluster
  }
}
```



Algorithm

Build a suffix tree for the genome

Search for long common substrings between each EST and the genome

Build gapped matching regions to account for sequencing errors and splicing

Use the longest overall matching region to map EST to locations in the genome

ESTs mapped to overlapping / nearby locations form a cluster

Figure 1: ESTmapper Algorithm

length). So the master thread just needs assign roughly the same number of ESTs to each thread to ensure an even division of work.

The two remaining components of the computation are reading in the precomputed suffix tree (and premapped ESTs, if doing incremental updates) from memory, and calculating clusters based on the locations from the mapping. File I/O to read suffix trees into memory is not amenable to parallelization, and clustering is a small enough percentage of the overall computation (less than 1% of running time for large data sets) that it has not been worthwhile parallelizing, though it could easily also be computed in parallel since EST mappings have already been sorted based on location.

Message-passing version (MPI) ESTmapper may also be parallelized using the MPI communications library for a PC cluster. To reduce communication, each processor reads the suffix tree into memory independently. The master node assigns ESTs to individual nodes, then each node maps its ESTs to the genome locally, without any need for interprocessor communication. Once mappings are computed each node sends its mappings to the master node for clustering. ESTmapper is very efficient on clusters, since little communication is needed, just at the beginning and end of the overall computation to send out ESTs and retrieve their mappings. Performance can be improved if each node stores a copy of the suffix tree in its local disk.

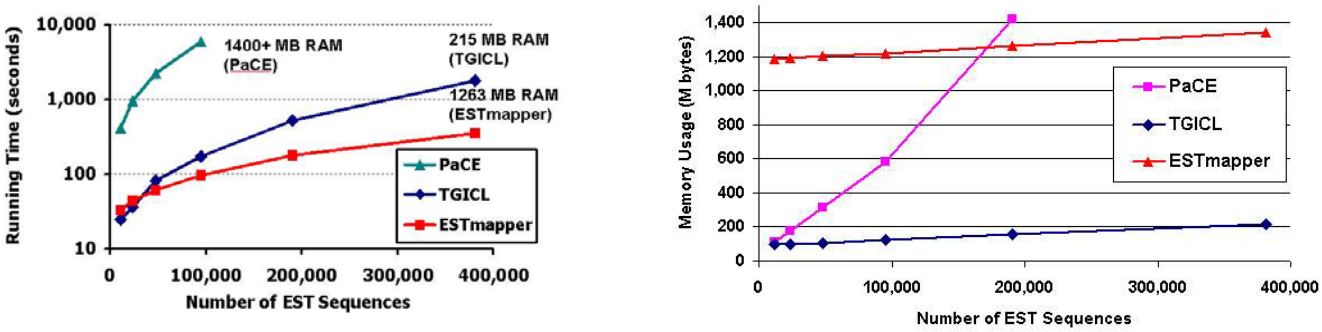


Figure 2: Running time and memory usage of EST clustering algorithms on SunFire 6800 with 8 processors

4 Experimental Evaluation

Environment To evaluate the performance and precision of ESTmapper, we downloaded the latest versions of the TGICL and PaCE software packages. All three programs were compiled and installed on a Sun SunFire 6800 shared-memory multiprocessor (SMP) with 24 UltraSparc processors and 24G memory, as well as a Linux PC cluster (each with dual AMD Athlon 1.6 GHz processors and 1G bytes of memory). ESTmapper itself was built on top of the WOTD suffix tree implementation [GJS32].

Data Sets For our experiments, we downloaded genome and EST sequences from NCBI GenBank for Arabidopsis thaliana (mustard plant) The Arabidopsis genome is 119 million bases long, divided into 5 chromosomes between 18 and 30 million bases and two shorter mitochondrial and chlorophyll sequences of 154K and 367K bases. There were 190,740 Arabidopsis ESTs with average length of 440 bases. In experiments we also look for mappings based on the reverse complement of each EST, resulting in an effective input data set of 381,480 EST sequences. For Homo sapiens (human), we download the pre-formatted human genome and EST sequences from UC Santa Cruz. The Human genome is significantly larger, being comprised of 3.4 billion bases divided into 22 chromosomes plus the X and Y chromosomes. There were 5,429,448 human EST sequences.

Genome Preprocessing On the SunFire 6800, creating a single suffix tree for the Arabidopsis genome required 1316 seconds and 1.237 GB of memory. The precomputed tree can be loaded into memory in 27 seconds. Partitioning the genome into two roughly even parts produced two suffix trees of size 636MB and 676MB, each of which can be read into memory in about 13 seconds. Our current ESTmapper implementation uses 32-bit pointers, which limits the amount of memory it can use to about 3GB. The human genome was too large to be combined into a single suffix tree, so we were forced to partition the human genome into 24 suffix trees (1 per chromosome, plus the X & Y chromosomes). Building the suffix trees required from 260MB to 2.6GB of memory and required a total of 8.46 hours. Precomputed suffix trees together take up 24.8 GB of storage on disk and require about 1800 seconds to load into memory. Overall, smaller genomes such as Arabidopsis and Drosophila are easily handled on a single PC, and large genomes such as Human and Mouse are doable on a medium sized server.

Performance Comparison Next we compare the computation time and memory usage of ESTmapper, TGICL, and PaCE by clustering a selection of ESTs (average length around 440 bases) from Arabidopsis against its full genome. Results on the SunFire 6800 are shown in Figure 2. Results on the Athalon Linux PC cluster are shown in Figure 3. Both set of results are for 8-processor parallel executions. In all figures the X-axis represents problem size, while the Y-axis represents either running time (in logarithmic scale) or memory usage. On the SunFire ESTmapper used a single suffix tree, while on the Linux PC cluster ESTmapper uses either 2 or 7 suffix trees due to memory limitations.

We see that pairwise comparison EST clustering techniques like TGICL can actually be very fast and efficient for clustering smaller numbers of ESTs. ESTmapper starts achieving better performance only when the number of ESTs clustered increases above 20,000. For larger data sets the quadratic number of

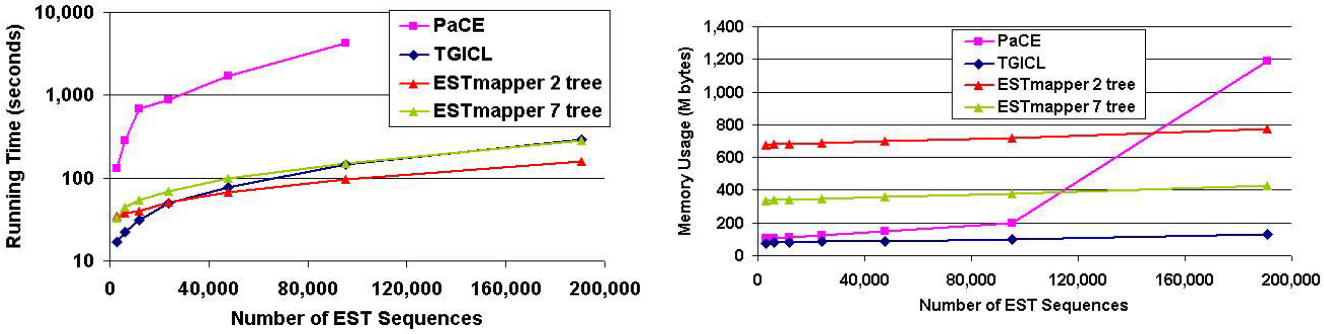


Figure 3: Running time and memory usage of EST clustering algorithms on Athalon Linux PC cluster with 8 nodes

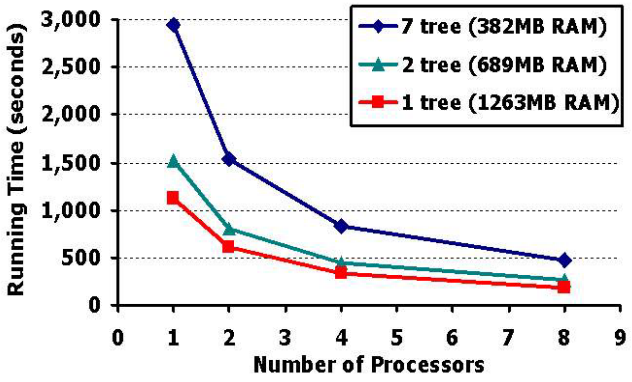


Figure 4: Running time after partitioning genome into multiple suffix trees, SunFire 6800 with 8 processors

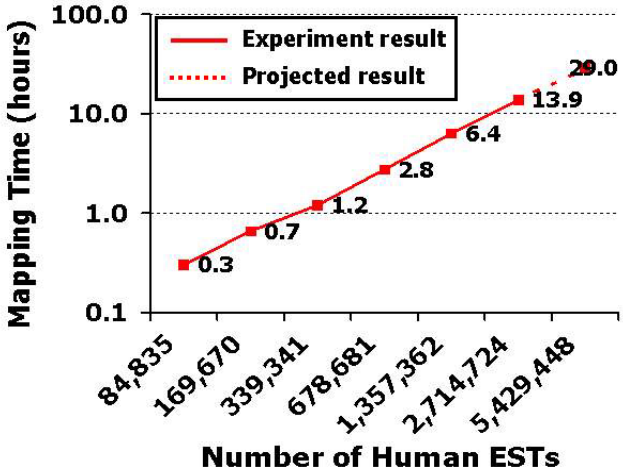


Figure 5: Running time of ESTmapper on large collections of human ESTs on SunFire 6800 with 8 processors

comparisons begins to slow down TGICL and increase its memory use. For 380K ESTs on the SunFire, ESTmapper is about five times faster than TGICL. PaCE is efficient in memory use for small numbers of ESTs, but quickly increases in memory use because it builds a suffix tree of all input ESTs, with the amount of memory increasing past ESTmapper after 200,000 ESTs. We were unable to obtain results for PaCE with large numbers of ESTs due to excessive running times. In comparison, memory use for ESTmapper increases very slowly with the number of ESTs. Instead, most memory is used to store the suffix tree for the genome.

ESTmapper Scalability We also evaluated the parallel scalability of ESTmapper for multiple processors, as well as its performance when the genome is partitioned into smaller pieces to reduce memory usage. Figure 4 presents ESTmapper running times and memory usage when 7, 2, and 1 suffix trees are used for the Arabidopsis genome, from 1 to 8 processors on the SunFire 6800. We see that due to its highly parallel algorithm (excluding reading in the suffix tree), ESTmapper achieves near linear speedups with increasing number of processors. Partitioning the genome into multiple suffix trees can greatly reduce memory usage, at the expense of increasing overall computation time. Going from a single suffix tree to two trees halved memory usage but increased execution time by about 50% with 8 processors. Using 7 suffix trees reduced memory usage further with greater running times.

Finally, we demonstrate the scalability of ESTmapper by using it to cluster the set of human ESTs in dbEST against the entire human genome. Results are shown in Figure 5. Recall that the current ESTmapper implementation has around a 3GB memory limit because it uses 32-bit pointers. As a result performance is reduced because we are forced to use separate suffix trees for each chromosome. Nonetheless ESTmapper was able to cluster 2.7 million ESTs in 13.9 hours using 8 processors. A running time of 29.0

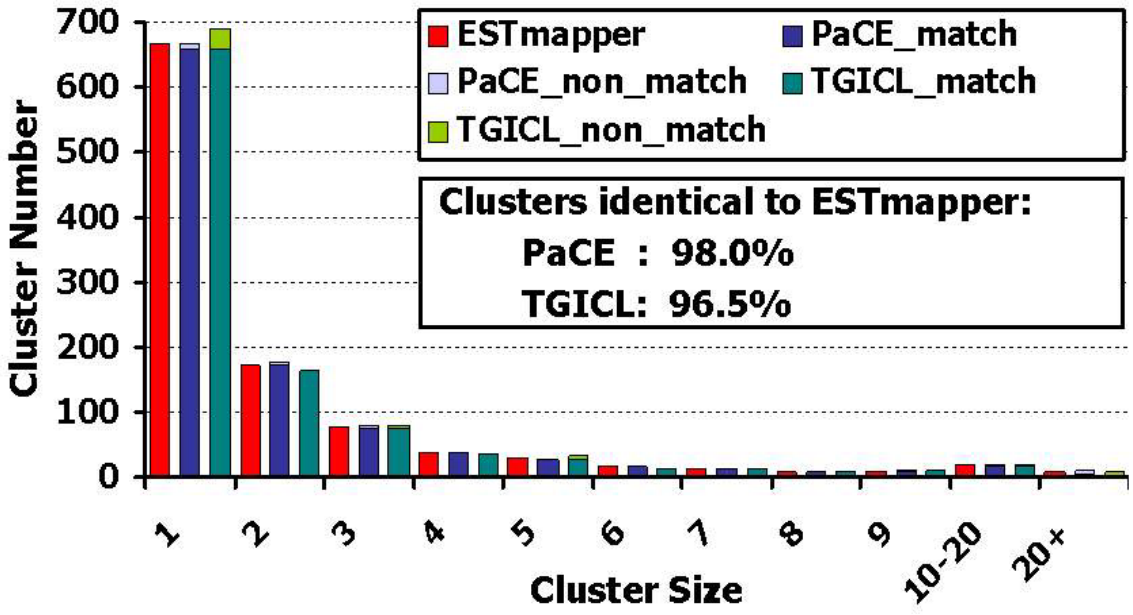


Figure 6: Precision of ESTmapper compared to TGICL and PaCE

hours is projected for the full 5.4 million Human ESTs, the full set of ESTs could not be clustered at once due to memory use above the 32-bit pointer limit. We see that clustering times are quite linear respect to the input size, even up to millions of ESTs. Memory usage is also fairly linear. In comparison, regression analysis predicts TGICL (127 hours) and PaCE (over 10,000 hours) would require significantly more time to cluster the entire set of human ESTs.

ESTmapper Precision We also performed some preliminary analysis of the precision of ESTmapper. First, we performed comparisons between ESTmapper TGICL, and PaCE, using the set of ESTs in dbEST mapped to the second chromosome of Arabidopsis. We counted for each algorithm the number of clusters formed and their sizes. We also counted the number of clusters in TGICL and PaCE that exactly matched clusters formed by ESTmapper. Results are shown in Figure 6. We see that the methods produced identical results for over 95% of the clusters. PaCE and TGICL produce slightly more singleton clusters than ESTmapper, with the number of clusters increasing when ESTs match each other less closely.

Second, we compared the clusters produced by ESTmapper against the first 100 clusters for Arabidopsis in the NCBI UniGene database. We found that precision depends on the maximum allowed distance used to cluster ESTs mapped to the genome. When using a distance of 5000 bases, 90 of the first 100 EST clusters in UniGene were identical to clusters found by ESTmapper. In comparison, 70 TGICL and 58 PaCE clusters were exact matches to the first 100 EST clusters in UniGene. We expect the precision of ESTmapper to improve further when we begin using more sophisticated gapped alignment algorithms similar to those used by UniGene.

Observations We believe these preliminary results are very encouraging. ESTmapper is at least as precise as existing EST clustering tools, and has the potential to scale to much larger sets of ESTs than previous algorithms. It also has very good parallel performance and can efficiently handle incremental updates, unlike pairwise comparison methods. The main weaknesses for ESTmapper are its memory usage and the requirement for mapping against the entire genome. However, these will become less problematic as memory sizes increase (because genome size remains constant) and the genomes of more organisms are sequenced.

5 Related Work

BLAST Many of clustering algorithms use BLAST (Basic Local Alignment Search Tool) to find closely matching local alignments between pairs of EST sequences [AGM+90]. BLAST is the most widely used search tool for screening large bioinformatic sequence databases, and accounts for a large portion (if not the majority) of the computation performed in bioinformatics. Its search strategy is based on using scoring matrices to compare short subsequences (words) in the query sequence against the entire target DNA or protein sequence database to find statistically significant matches, then attempting to extend these matches to find the most similar sequences or subsequences. BLAST is such a popular tool that several sequential and parallel versions (WU-BLAST, megaBLAST, mpiBLAST, BLAST++) have been developed to improve its performance. Nonetheless, BLAST is too computationally intensive to use directly map ESTs to the genome.

BLAT An alternative pairwise alignment tool is known as BLAT [Ken02]. BLAT maintains a precomputed hash table index of the locations of all non-overlapping subsequences (words) of length k. Performance is dramatically improved because queries do not need to scan the 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. The authors of BLAT were able to align 3.7 million human ESTs against the entire human genome in 220 CPU hours by keeping a 1 GB index in memory. BLAT represents an interesting alternative to using suffix trees. Since its precision and expense depends on the value of k used to generate the index, more work is needed to compare the performance and precision of BLAT to ESTmapper.

CAP3 Many clustering tools also use CAP3 (Contig Assembly Program) to *assemble* the multiple ESTs in a cluster into a single consensus sequence [HM99]. CAP3 uses a large number of forward-reverse constraints to locate and correct errors and link contigs separated by gaps. The algorithms in CAP3 are designed to tolerate common errors made by automatic DNA sequencers. CAP3 can also be applied directly on sets of unclustered EST sequences, but is usually only used for individual EST clusters because of its high computation time and memory usage.

UniGene/MegaBlast NCBI UniGene is probably the mostly widely known and used collection of EST clusters [W+03]. Historically, UniGene clustering proceeds in stages, beginning with a all-to-all pairwise sequence comparison using megaBLAST to find initial clusters [ZSWM00], followed by additional passes based on biological knowledge to improve precision and make the resulting clusters more gene-oriented. Because of the large number of processing steps using different criteria, UniGene does not actually provide its software in a single package. Instead users must access and download the EST clusters produced by researchers at NCBI. Very recently in 2004 UniGene has also begun using alignments to the genome as the basis for building clusters for organisms with sequenced genomes. NCBI describes the criteria used for deciding when ESTs should be placed in a single cluster, but has not made clear how such mappings are calculated.

SpliceNest SpliceNest is a graphical tool that displays the mapping to the genome of consensus EST sequences from the GeneNest clustering tool [CHV02]. The display is designed to help users visualize possible alternative splicing models for the gene. SpliceNest is designed to display a single consensus sequence at a time, while ESTmapper attempts to map large sets of EST to the genome efficiently.

GeneNest GeneNest is a software and database system for automated generation and visualization of gene indices [HBR+00]. GeneNest starts from EST sequences from the EMBL and UniGene databases, applies extensive sequence clipping to improve the quality of the resulting EST sequences, the applies all-to-all pairwise BLAST to find closing matching pairs. ESTs with at least 50% near-perfect matches are merged into a single cluster. Clusters are then assembled into one more more consensus sequences sharing global similarity. The authors claim adding some genomic sequences containing putative genes to the clustering can lead to dramatic improvements in clustering compared to UniGene and TGICL.

CLOBB CLOBB (CLuster On the Basis of BLAST similarity) is another EST clustering algorithm that is based on all-to-all pairwise comparisons [PGB02]. CLOBB is designed to support incremental construction of EST clusters, and supports functions such as tracking cluster merging, identifying groups of related clusters, and tries to avoid chimeric clusters caused by improperly joined ESTs. The primary feature of CLOBB is its portability, achieved by writing the entire software as a Perl script relying on local BLAST installations for pairwise comparisons based on BLAST similarity scores.

STACK/d2_cluster The STACK (Sequence Tag Alignment and Consensus Knowledgebase) system uses the d2_cluster algorithm, which also creates clusters based on pairwise comparisons between EST sequences [BDH99]. However, d2_cluster differs from previous pairwise algorithms (e.g., UniGene, TGICL) in that instead of using a local alignment algorithm like BLAST, it considers two ESTs to be related if a sufficiently large percentage of bases are identical within a fixed-length sliding window. Such comparisons are more quickly computed than local alignments, but require longer matching sequences to be accurate. Sequences between clusters are compared and clusters are merged if any pair of sequences between the two clusters are found to match sufficiently. The authors compare d2_cluster to UniGene, and found it to be somewhat more aggressive in merging clusters. d2_cluster has been parallelized on the SGI Origin2000.

Xsact Xsact is another algorithm that attempts to avoid performing all-to-all pairwise comparisons [MCJ03]. Unlike PaCE, the Xsact algorithm uses a *suffix array*, a lexicographically ordered array of all suffixes of the EST sequence. Xsact generates the suffix array using a radix sort, and reduces memory requirements by generating suffix arrays for subsets of the EST data. Once generated, the suffix array is used to find pairs of ESTs with long common substrings. Suffix arrays have asymptotically worse $O(nlog(n))$ performance, but still improve on the worst-case quadratic-time behavior of all-to-all pairwise comparisons.

Instead of using a local sequence alignment algorithm like BLAST, Xsact compares two sequences by calculating a score by finding the longest set of *consistent* (i.e., non-overlapping and in the same order) matching substrings between each pair of ESTs. ESTs pairs above a certain similarity score are merged into a single cluster hierarchically, starting with the highest scoring pairs. Clusters are then split according to the clustering threshold. The authors implemented Xsact in Haskell, a functional programming language, and were able to cluster 10,000 ESTs on a single Intel Xeon PC with 1GB memory.

OASIS Like ESTmapper, OASIS is a tool that builds a WOTD suffix tree of large sequence databases [MPK03]. The goals are quite different, however. OASIS attempts to reduce the cost of sensitive pairwise alignment algorithms such as Smith-Waterman, by modifying their dynamic programming algorithm to operate on a lazy WOTD suffix tree to eliminate redundancy from common substrings. Experiments performing batched searches of short (60 bases) sequences of the SWISS-PROT database of protein sequences and the Drosophila (fruit fly) genome demonstrated performance comparable to BLAST and an order of magnitude improvement over Smith-Waterman. In comparison ESTmapper uses eager WOTD suffix trees to find long common substrings to construct long gapped alignments for clustering.

6 Conclusions

In this paper, we examined existing algorithms for clustering ESTs and identified their weaknesses. We developed ESTmapper, a new approach for clustering ESTs by efficiently mapping them to the genome using suffix trees. We developed a prototype implementation and compared its precision and performance to two existing clustering tools. Though a number of weaknesses remain in our approach, preliminary results have been very encouraging. We plan on continuing to improve and extend ESTmapper to make it a more useful and widely applicable tool. Our work on ESTmapper to enhance EST clustering is a part of our overall goal—taking advantage of increasing computation power to provide more useful information to bioinformatics researchers.

7 Acknowledgements

The authors are very grateful to Anantharaman Kalyanaraman for assistance with using and interpreting results from PaCE.

References

[AGM+90] Altschul, Gish, Miller, Myers, and Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[BDH99] J. Burke, D. Davison, and W. Hide. d2_cluster: a validated method for clustering EST and full-length cDNA sequences. *Genome Res*, 9(11):1135–1142, Nov 1999.

[CHV02] 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.

[GJS32] R. Giegerih, S. Jurtz, and J. Stoye. Efficient implementation of lazy suffix trees. *Software—Practice ad Experience*, 33:1035–1049, 2032.

[Gus77] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1977.

[HBR+00] S. Haas, T. Beissbarth, E. Rivals, A. Krause, and M M. Vingron. GeneNest: automated generation and visualization of gene indices. *Trends Genet*, 16(11):521–523, 2000.

[HM99] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Res*, 9:868–877, 1999.

[KAKB03] 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.

[Ken02] W. Kent. BLAT – The BLAST-like alignment tool. *Genome Research*, 12:656–664, 2002.

[LHP+00] 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.

[MCJ03] K. Malde, E. Coward, and I. Jonassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19:1221–1226, 2003.

[MPK03] 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, September 2003.

[PGB02] John Parkinson, David Guiliano, and Mark Blaxter. Making sense of EST sequences by CLOB-Bing them. *BMC Bioinformatics*, 3(31), October 2002.

[PHL+03] 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.

[W+03] D. Wheeler et al. Database resources of the national center for biotechnology. *Nucleic Acids Research*, 31:28–33, 2003.

[ZSWM00] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology*, 7:203–214, 2000.