Nucleic Acids Research, 2003, Vol. 31, No. 11 ***2963–2974***
DOI: 10.1093/nar/gkg379

Efficient clustering of large EST data sets on parallel computers

Anantharaman Kalyanaraman, Srinivas Aluru[1,*], Suresh Kothari[1] and Volker Brendel[2]

Department of Computer Science, Iowa State University, Ames, IA 50011, USA,[1]Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA and [2]Department of Zoology and Genetics and Department of Statistics, Iowa State University, Ames, IA 50011, USA

Received November 20, 2002; Revised March 7, 2003; Accepted March 26, 2003

ABSTRACT

Clustering expressed sequence tags (ESTs) is a powerful strategy for gene identification, gene expression studies and identifying important genetic variations such as single nucleotide polymorphisms. To enable fast clustering of large-scale EST data, we developed PaCE (for Parallel Clustering of ESTs), a software program for EST clustering on parallel computers. In this paper, we report on the design and development of PaCE and its evaluation using *Arabidopsis* ESTs. The novel features of our approach include: (i) design of memory efficient algorithms to reduce the memory required to linear in the size of the input, (ii) a combination of algorithmic techniques to reduce the computational work without sacrificing the quality of clustering, and (iii) use of parallel processing to reduce run-time and facilitate clustering of larger data sets. Using a combination of these techniques, we report the clustering of 168 200 *Arabidopsis* ESTs in 15 min on an IBM xSeries cluster with 30 dual-processor nodes. We also clustered 327 632 rat ESTs in 47 min and 420 694 *Triticum aestivum* ESTs in 3 h and 15 min. We demonstrate the quality of our software using benchmark *Arabidopsis* EST data, and by comparing it with CAP3, a software widely used for EST assembly. Our software allows clustering of much larger EST data sets than is possible with current software. Because of its speed, it also facilitates multiple runs with different parameters, providing biologists a tool to better analyze EST sequence data. Using PaCE, we clustered EST data from 23 plant species and the results are available at the PlantGDB website.

INTRODUCTION

Expressed sequence tags (ESTs) are sequences of typically at most a few hundred base pairs in length that are determined by single-pass sequencing of the 5′ or 3′ ends of cDNA clones. Because the cDNA libraries are typically generated from tissue or developmental stage or otherwise specific mRNA samples and are randomly selected for sequencing, EST representations provide a dynamic view of genome content and expression. Currently, more than 5 million human ESTs, more than 3.7 million mouse ESTs, and large collections of ESTs from various other organisms (many with hundreds of thousands of entries each) are publicly available (http://www.ncbi.nih.gov/dbEST/dbEST_summary.html). The primary interest in these resources is based on the ability to derive sets of unique genes from the data that represent the transcriptome of each species as completely as possible. This task remains a challenging problem not only because of the imposing size of the EST databases but also because of intrinsic difficulties arising from low sequence quality, highly similar (but distinct) gene family members, chimeric cDNA clones, retained introns and alternatively spliced transcripts, incomplete gene coverage, and other limitations. For a recent review of different strategies to address this task and the associated gene indexing databases see Bouck *et al.* (1).

The first step in deriving a gene index from an EST set is to remove redundancy by clustering ESTs representing the same native transcripts. The criteria and tools for clustering are chosen depending on the precise goals of the clustering. (i) ESTs are put into distinct clusters such that each cluster represents a distinct gene, including all alternative transcript isoforms derived from the same gene. This clustering strategy is adopted in NCBI's UniGene database [http://ncbi.nlm.nih.gov/UniGene/; see Schuler (2)]. (ii) Each EST cluster is deemed to represent a distinct mRNA transcript. In particular, alternative transcript isoforms are represented by distinct EST clusters. This strategy is implicit in DNA assembly tools such as Phrap (http://www.phrap.org/), TIGR Assembler (3) or CAP3 (4) that are also widely used for EST clustering. For example, the TIGR Gene Indices are based on CAP3 clustering [http://www.tigr.org/tdb/tgi.shtml; see Quackenbush *et al.* (5)]. (iii) ESTs are first categorized by their RNA source and are subsequently clustered separately for each source sample. The Sequence Tag Alignment and Consensus Knowledgebase (STACK, http://www.sanbi.ac.za/Dbases.html) uses this approach to cluster human ESTs based on tissue-specificity (6). Because in the first strategy, ESTs

*To whom correspondence should be addressed. Tel: +1 515 294 3539; Fax: +1 515 294 8432; Email: aluru@iastate.edu

The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors

derived from alternatively spliced isoforms are grouped in the same cluster, no consensus sequence is generated in UniGene. The other two strategies allow generation of consensus sequences corresponding to the putative transcript represented by each cluster. In cases where the genome of a species is already available, a spliced alignment of such consensus sequences can lead to identification of the locations of the respective genes; this is a more efficient alternative to direct spliced alignment of the ESTs to the genome, because of the latter's redundancy (7).

Other than clone pair information, the primary evidence for ESTs representing the same gene or transcript is derived from significant sequence similarity in a region of sequence overlap. The same principle applies to fragment assembly tools, and from that perspective, such tools can be applied to the EST clustering problem. All these tools generate a consensus sequence corresponding to each putative transcript. Significant sequence similarity can be ascertained by pairwise sequence alignment allowing for mismatches and insertions/deletions using standard dynamic programming algorithms (8,9). However, because the run-time of these algorithms is quadratic in the lengths of the sequences being aligned, it is in practice computationally too intensive to be run for all pairs of sequences. Hence, alternative means of approximate overlap detection are used for fast identification of pairs of sequences with potential for good quality overlap, which we call 'promising pairs' henceforth. The dynamic programming algorithm is then run on the promising pairs only. For example, STACK clustering involves the word-based d2_cluster algorithm (10), and Phrap and CAP3 generate promising pairs using fast algorithms based on exact string matching.

Fragment assembly tools work well when the fragments represent a random sampling of the genomic DNA with uniform coverage. In the extreme case of near single layer tiling for a non-repetitive source, the number of promising pairs is linear in the number of fragments. Because ESTs are derived as a result of gene expression, the number of ESTs that correspond to a gene depends on the level of expression of the gene. Thus, the EST coverage is non-uniform. As a result, the worst-case number of promising pairs due to EST overlaps is quadratic in the number of ESTs. Moreover, assembly programs such as CAP3 and Phrap categorize sequences using all possible strings of length w such that all sequences containing a specific substring of length w are in the same category. This affects the run-time of the software in generating the promising pairs. For instance, if two ESTs share a substring of length $l > w$, then all the $l - w + 1$ substrings of length w that are common to it within the l-length match should be considered before detecting the presence of the l-length common substring. Typically as w is much smaller than l for strongly overlapping ESTs, this affects the run-time in generating these promising pairs. A better alternative will be to devise an algorithm that reports pairs based on maximal common substrings without having to collect all the smaller length common substrings to identify them.

We tested the performance of CAP3, Phrap and TIGR Assembler when applied to EST clustering using an *Arabidopsis* benchmark EST data set. The test runs are done on a single processor of an IBM xSeries node with 2.25 GB of memory so that results can be compared directly with our parallel software. The results are shown in Table 1. While the run-times are large, a more serious problem is the memory requirement. We observed that for large data sets such as the rat collection comprising of 327 632 ESTs, these software programs run out of memory. The motivation behind our work is to overcome this memory bottleneck, in addition to reducing run-time.

The main contribution of our research is a parallel software program, named PaCE (for Parallel Clustering of ESTs), which aims at clustering ESTs that come from the same gene or from paralogous genes. Once clustered, the ESTs corresponding to each cluster can be further processed with an assembly tool, allowing generation of one consensus sequence per putative transcript. This combination enables clustering and assembly of large-scale EST data sets due to the following two reasons: (i) the memory required by our clustering algorithm grows only linearly in the size of the input, and (ii) the input size to the assembly tools is now reduced from the complete set of ESTs to the size of the biggest cluster generated by PaCE (measured in the number of ESTs of a cluster), enabling the assembly tools to generate consensus sequences without running out of memory. In addition to memory benefits, our novel ideas for generation of promising pairs in parallel and management of EST clusters reduce the run-time significantly, while ensuring good quality clustering.

For the purpose of quality assessment, we have used a benchmark clustering created based on spliced alignment of ESTs on the *Arabidopsis thaliana* genome (11). CAP3 was used for assembly on each of our PaCE clusters because it gives the best quality results of the three software programs we tested [see also Liang *et al.* (12)].

MATERIALS AND METHODS

In this section, we first give an overview of the EST clustering problem and our novel approach and then give details of the implementation. Experimentation with current software indicates that pairwise alignment using dynamic programming is the run-time intensive part and generation of promising pairs is the memory intensive part. To satisfactorily address these potential bottlenecks, we designed the following approach. Initially, each EST can be thought of as a cluster by itself. Two

Table 1. Run-times (min) of TIGR Assembler, Phrap and CAP3 on different portions of the complete 168 200 *Arabidopsis* benchmark data set

Input (n)	TIGR Assembler		Phrap		CAP3	
	Run-time	PMU	Run-time	PMU	Run-time	PMU
50 000	321	1.3 GB	38	1.07 GB	72	711 MB
100 000	1114	2.23 GB	85	1.17 GB	150	1.93 GB
168 200	X	X	224	2.2 GB	X	X

PMU, peak memory usage; X, a total of 2.25 GB of memory was not sufficient for the software to complete execution.

EST clusters can be merged provided an EST from each cluster can be identified that show strong overlap using the pairwise alignment algorithm. This process is continued until no further merges are possible. If a pair of identified ESTs does not show strong overlap, the corresponding clusters cannot be merged, and the effort in testing is wasted. However, there may be another pair of ESTs from these clusters that may have strong overlap, causing the clusters to merge when this pair of ESTs is aligned. The order in which promising pairs are processed does not affect the final set of clusters formed. However, the order does have significant influence on the run-time it takes to compute the clusters, as explained below.

Significant savings in run-time can be achieved by fast identification of pairs that would likely yield a positive outcome when the pairwise alignment algorithm is run. A positive outcome helps in merging of two clusters. As a result, it is no longer necessary to test pairs of ESTs where each is drawn from one of the two clusters. Hence, by early identification of promising pairs of ESTs that cause clusters to merge, it becomes unnecessary to align many promising pairs generated at later stages. Thus, instead of merely finding all pairs that meet a certain test criteria (such as sharing a substring of length 20 or more), we generate pairs in decreasing order of overlap quality, as measured by an efficiently computable measure. We use maximal common substring length as the measure. A maximal common substring of a pair of sequences is a substring common to both the sequences that cannot be extended at either end to result in a longer match. The rationale for using the measure is that pairs of ESTs with larger length exact matches are more likely to pass the alignment test. To eliminate the large memory required for storing the promising pairs, we designed an on-demand algorithm that remembers its state and produces the next set of pairs as and when required. We also address the important problem of avoiding generation of the same pair multiple times, even though it is nontrivial to do so because we do not store previously generated pairs. Our algorithm uses the generalized suffix tree (GST) data structure (13).

The organization of PaCE is as follows. We first build a distributed representation of the GST data structure in parallel. This data structure is constructed for the input set of EST sequences and their Watson–Crick complements, and is used for on-demand generation of promising pairs of ESTs in decreasing order of maximal common substring length. The pair generation itself is done in parallel. Maintaining and updating of the EST clusters is handled by a single processor, which acts as a master processor directing the remaining processors to both generate batches of promising pairs and perform pairwise alignment on selected promising pairs. It is not mandatory to perform pairwise alignment of each generated pair because the current set of EST clusters may obviate the need to do so. Hence, the master processor is also responsible for the selection of pairs to be aligned and is a necessary intermediary between pair generation and alignment. In order to reduce communication overhead, the master processor dispatches the selected pairs in batches of size batchsize, a configurable parameter. To provide an added degree of flexibility in balancing the load, we do not require that a pair generated on a processor be allocated to the same processor if a pairwise alignment is needed.

The execution of the software can be labeled as two successive phases: we refer to the GST construction component as the preprocessing phase, and the components for pair generation, pairwise alignment and EST cluster management collectively as the clustering phase. In what follows, we describe each component of PaCE and the core ideas in our memory and time efficient parallel algorithms. For computational purposes, we represent each EST sequence as a string over alphabet $\Sigma = \{A, C, G, T\}$. We use the following notations. Let n be the number of EST sequences and the set $\zeta = \{e_1, e_2, \ldots, e_n\}$ denote the ESTs. The sum of the lengths of all the n ESTs is denoted by N. Let l be the average length of an EST, i.e. $l = N/n$. Let $S = \{s_1, s_2, \ldots, s_{2n}\}$ denote the $2n$ sequences such that $e_i = s_{2i-1}$ and $e_i = s_{2i}$, where each $\bar{e}_i$ denotes the complementary strand of e_i. We use the terms sequence and string in an equivalent manner.

Parallel construction of GST

Let s be a sequence of length m over alphabet Σ. A suffix tree for s is a directed tree with m leaves numbered 1 through m (13). Let the path-label of a node v denote the sequence obtained by concatenating the edge labels on the path from root to v. Let the string-depth of a node v denote the number of characters in its path-label. The tree has the property that the path-label of the leaf labeled i is the suffix starting at position i of s. A GST for a set of n sequences is a suffix tree constructed using all suffixes of the n sequences. If N is the sum of the lengths of all the sequences, the GST has at most N leaves, exactly N leaf labels, $O(N)$ size, and can be constructed in $O(N)$ time (13). The main idea of using a GST data structure for generating promising pairs is that if two ESTs share a maximal common substring, then it will be represented as a path-label of a node in the corresponding GST, with suffixes from the two ESTs starting with the common substring occupying leaves in the subtree of the node.

We construct the GST for S in parallel as follows. Initially, the ESTs are distributed across processors such that each processor has an approximately equal share of the input. Each processor scans its ESTs and partitions the suffixes of these ESTs into at most $|\Sigma|^w$ buckets, based on the first w bases. The total number of suffixes in each bucket over all the processors is computed using a parallel summation algorithm in $O(\log p)$ communication steps, where p denotes the number of processors. The buckets are then distributed to the processors such that (i) all the suffixes in a bucket are allocated to the same processor, and (ii) the total number of suffixes in all the buckets in each processor is close to $O(nl / p)$. In the subsequent step of constructing the local portions of the GST, the former eliminates the need for communication, while the latter ensures load balancing.

For each bucket, the processor responsible for it constructs the portion of the GST using all the suffixes in the bucket. Note that a sequential suffix tree construction algorithm can no longer be used because not all suffixes of an EST fall in the same bucket, unless the EST sequence is a repetition of a single base. To construct the local GST, we use the simple approach of scanning the sequences one base at a time. Assuming the construction of each processor receives $O(nl / p)$ suffixes with an average length of l, the run-time for local GST is $O(nl^2 / p)$. This algorithm works well in practice because l is small and is independent of n.

Note that the local GST on a processor represents a collection of subtrees in the GST for all n ESTs and their reverse complementary strands, except the portion consisting of nodes with string-depth $< w$. Care must be taken in choosing w. While assigning a large value may result in loss of some potential overlapping pairs, assigning a low value will result in a small number of buckets for distribution among processors. A value of 10 will generate as many as $4^{10} > 1\,000\,000$ buckets, enough to distribute them in a load-balanced fashion even on large multiprocessor systems. Because of concern for space-efficiency, each subtree in the local GST is stored in the order of depth-first search traversal of the tree. Each node contains a single pointer to the rightmost leaf in its subtree. All the children of a node can be retrieved using the following procedure. The first (leftmost) child of a node is stored next to it in the array. The next sibling of a node can be obtained by following the pointer to its rightmost leaf and taking the node in the next entry of the array. If the rightmost leaf pointers of a node and a child of it are identical, the child is the rightmost child of the node. Each node also stores its string-depth, where the string-depth is measured with respect to the global GST.

On-demand pair generation

Once the GST for S has been constructed, it can be used to generate promising pairs. Our algorithm for on-demand pair generation is a variant of the suffix tree algorithm for computing all maximal repeats of a sequence (13). A pair of EST sequences should be reported if they share a maximal common substring of length greater than or equal to a threshold value. Because a pair of sequences can have more than one such maximal common substring, our algorithm might generate the pair more than once. The number of such duplicates per pair generated by our algorithm is at most the number of distinct maximal common substrings of the pair. The key here is that we report the presence of a maximal common substring for a pair directly from the suffix tree, without having to look at all the smaller length non-maximal common substrings contained within it. This results in a significant reduction in the run-time as opposed to the methods deployed in generating promising pairs by existing software as explained in the Introduction.

The following provides definitions and notations required for describing our on-demand pair generation algorithm. A substring α of a sequence is said to left-extensible (alternatively, right-extensible) by character c if the character to the left (alternatively, right) of α in the sequence is c. If α is a prefix of the sequence, then it is said to be left-extensible by λ, the null character. For a given pair of sequences, a common substring α is maximal if it is neither left-extensible by the same character nor right-extensible by the same character in both sequences. For a node v in a GST for S, let leaf-set(v) S represent the set of ESTs in S which have at least one of their suffixes as a path-label of some leaf node under v's subtree. If two suffixes from different sequences are in the leaf-set of an internal node in a GST, the sequences share a common substring of length equal to the string-depth of the node. Thus, the pair of ESTs can be generated if the substring can be identified to be maximal.

To generate pairs, we first sort the nodes of the GST in decreasing order of string-depth, and process them in that order to ensure that a pair with a longer maximal common substring is reported before a pair with a shorter length maximal common substring. For each node v in the tree, we store the set of sequences found in the leaf-set of its subtree. This set is partitioned into five sets $l_A(v)$, $l_C(v)$, $l_G(v)$, $l_T(v)$ and $l_\lambda(v)$, referred to as the *lsets* of v. If a sequence s_i is in $l_c(v)$ (for $c \in \Sigma \cup \{\lambda\}$), then there exists a suffix of s_i in the leaf-set of the subtree under v such that the suffix is left-extensible by the character c. If the suffix is the entire sequence, then it is considered left-extensible by λ. The *lsets* at a node are generated after it is processed, and are removed after its parent is processed. This limits the total space required for storing them to $O(N)$, linear in the size of the input.

Figure 1 illustrates the main idea used in the pair generation algorithm. Consider a node v and its children $u_1, u_2, \ldots, u_m$. Before generating the pairs, the *lsets* of each child node are scanned (in no particular order) to ensure that a sequence is present in at most one *lset* of one child node, by arbitrarily removing multiple copies. This scanning can be done in time proportional to the total length of all these *lsets*. After the duplicates are removed, the pairs generated at v are given by $\{(s_i,s_j) \mid s_i \in l_c(u_k), s_j \in l_d(u_l), k \neq l, ((c \neq d) \vee (c = d = \lambda))\}$. After generating the pairs at a node, the *lsets* of the children corresponding to the same character are combined to form the *lsets* at the node. Also, a generated pair is discarded if the sequence corresponding to the smaller EST subscript number is in complemented form. This is to avoid duplicates such as generating both (e_i, e_j) and $(\bar{e}_i, \bar{e}_j)$, or generating both $(e_i, \bar{e}_j)$, and $(\bar{e}_i, e_j)$. It can be shown that the total run-time of this algorithm is proportional to the number of pairs generated plus the cost of sorting the nodes of the GST. Thus, apart from the expense of sorting GST nodes according to string depth, which takes $O(N / p \log (N/p))$ run-time, the algorithm has a generation rate of $O(1)$ run-time per pair.

Parallel clustering

Our parallel EST clustering algorithm makes use of the master-slave paradigm. The master processor is responsible for maintaining and updating the clusters. It receives promising pairs of ESTs from slave processors and determines which of these pairs should be explored using a pairwise alignment algorithm. It dispatches pairs in units of *batchsize* to slave processors to perform pairwise alignments and return the results. Upon receiving the result of a pairwise alignment, it determines if the clusters corresponding to the pair should be merged based on the received results, and additional evidence if necessary. The slave processors are responsible for generating pairs as demanded by the master processor and to perform pairwise alignments of the pairs dispatched by the master processor.

The clusters are maintained by the master processor using the union-find data structure (14). Initially, each EST is in a cluster of its own. We require two operations on the cluster: (i) to find the cluster of an EST (find), and (ii) to merge two clusters (union). The average run-time per operation using the union-find data structure is given by the inverse Ackermann's function (14), a constant for all practical purposes.

The master processor maintains a large buffer of pairs yet to be processed. A message received by the master processor from a slave processor consists of two parts—results of the pairwise alignment for a number of pairs (same as the *batchsize*) and a batch of the next set of promising pairs

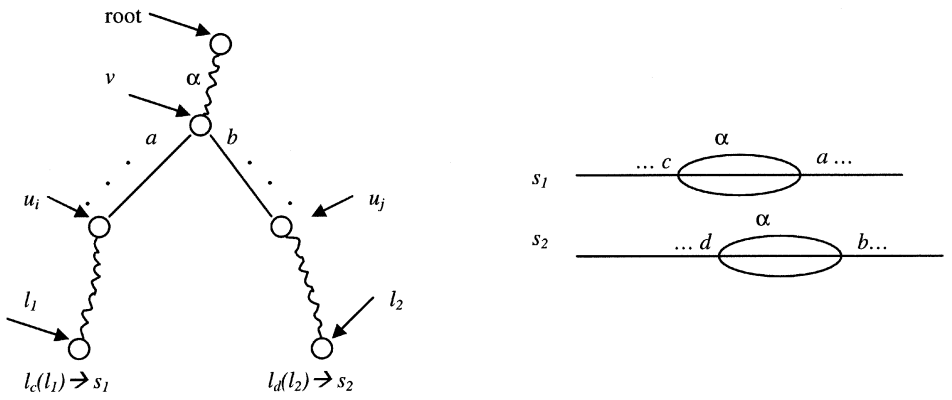

Figure 1. Representation of a maximal common substring alpha of two sequences s_1 and s_2 in the corresponding GST. (**a**) The paths to the suffixes corresponding to the sequences that have the prefix alpha share a common path from the root to an internal node v, whose path-label is alpha. As alpha is maximal, the paths fork from v into two different child nodes whose incoming edges have labels starting with characters a and b (where $a \neq b$). The paths end in leaves where the suffixes are stored in the *lset* corresponding to their left characters, c and d (where $c \neq d$). (**b**) Context of the maximal common substring alpha in the sequences s_1 and s_2.

generated on the slave, the number of which is based on a previous request from the master processor. It updates the EST clusters using the results received from the slave. Once the results are incorporated into the clusters, the master proceeds with the task of queuing the promising pairs received to the buffer of to-be-processed pairs. A pair is added to the buffer if and only if the corresponding sequences are currently in different clusters and neither of the ESTs has been shown to be contained in any other EST. Otherwise, the pair is rejected. The master processor immediately dispatches a message to the slave processor consisting of (i) *batchsize* (fewer, if not available) number of pairs from its buffer, and (ii) the number of promising pairs to be returned along with results of running pairwise alignment algorithm on each pair in (i).

The number of promising pairs the master processor requests from a slave processor is determined as follows. The master keeps track of the ratio μ of the total number of pairs from the most recent set of promising pairs received from the slave to the number of these pairs actually added to the buffer. It also keeps track of δ, which is the ratio of total number of slave processors to the number of slave processors that have not yet run out of promising pairs. Let *nfree* denote the number of free slots in the buffer maintained by the master processor. The number of promising pairs requested from the slave processor is determined by $\delta \times \min(\mu \times batchsize, nfree / p)$. This is to receive approximately *batchsize* number of useful promising pairs from each slave, without running the risk of overflowing the buffer in case all the received pairs are added to the buffer.

To get the process started, each slave processor initially generates $3 \times batchsize$ number of pairs, consisting of three equal portions of *batchsize* number of pairs. At the beginning, all promising pairs must be explored. The processor immediately sends the third portion to the master processor, and starts pairwise alignments on the first portion. Once the results of the first portion are obtained, it sends the results along with a newly generated batch of pairs to the master processor. While waiting to receive another batch of pairs from the master processor, it works on the second portion. Thus, the processor always has the next batch of pairs to work on, between submitting the results of the previous batch and receiving another set of pairs from the master processor. Much of the overhead in communication is masked by this overlapping of computation and communication.

To perform pairwise alignment, recall that a maximal common substring of the pair is already known. Figure 2a shows the dynamic programming table for computing the pairwise alignment. Instead of aligning entire ESTs, we reduce work by merely extending the maximal substring match at both ends using gaps and mismatches. This limits the area of the table to be computed as shown in Figure 2b. To further limit work, we use banded dynamic programming (15), where band size is determined by the number of errors tolerated. Quality can be controlled by the set of parameters described in the Results section.

Data sets

The accuracy of the results is assessed using a benchmark data set consisting of 168 200 *A.thaliana* ESTs (11). To assess quality and run-time performance as functions of data size, two different subsets of the 168 200 ESTs were derived in the following way: from the set of clusters representing the benchmark data set, two subsets of clusters were randomly extracted, such that the total number of ESTs in the clusters was ~50 000 in one and ~100 000 in the other. The data set consisting of 327 632 rat ESTs was downloaded from the NCBI site (http://www.ncbi.nlm.nih.gov:80/entrez/query.fcgi?SUBMIT=y) with the query ('rattus' [Organism] AND EST[PROP]) (date accessed: May 1, 2002).

Software availability

PaCE is freely available for non-profit, academic use. The source code and executables can be obtained by email request

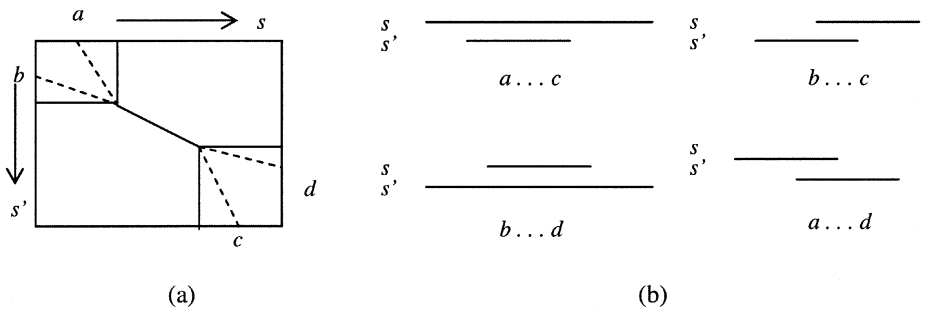

Figure 2. (**a**) Pairwise alignment strategy for extending a maximal common substring at both ends. (**b**) Four types of overlaps accepted as indication to merge clusters, and their corresponding optimal paths in the dynamic programming table. For instance, the labeling *a*…*c* means the alignment of the sequences containing the path from point *a* to point *c*.

to ananthk@cs.iastate.edu. For specific projects, we may also be able to provide access to our parallel cluster.

RESULTS

We implemented PaCE using C and Message Passing Interface (MPI) (16). The software can be run on any platform that supports a MPI compiler. Such compilers are freely available for a variety of architectures (e.g. the MPICH compiler; see http://www-unix.mcs.anl.gov/mpi/mpich). We report the performance and quality results of PaCE using an IBM xSeries cluster consisting of 30 dual-processor nodes with 1.26 GHz Intel Pentium III processors, connected by Myrinet. The memory available at each node is 2.25 GB.

Quality assessment

The accuracy of the results is assessed using a benchmark data set consisting of 168 200 *A.thaliana* ESTs (11). The data set was created by spliced alignment of ESTs to their cognate locations in the *Arabidopsis* genome, with subsequent clustering based on genome location: ESTs with genomic locations overlapping by at least 40 bp were clustered. Out of the 168 200 ESTs, 146 527 ESTs mapped to unique locations on the genome, while the rest (21 673 ESTs) mapped to more than one cognate location. Each EST aligning to more than one genome location was mapped to the cluster corresponding to the location that gave the maximum alignment score with the EST. This procedure of generating the benchmark clusters captures various interesting cases: (i) ESTs originating from the same gene are clustered irrespective of their mRNA transcript source, and (ii) ESTs originating from highly similar genes are separated. Chimeric ESTs were not included in the benchmark data set.

Our procedure was to cluster the benchmark data set with PaCE, and then run CAP3 on each resulting cluster. As a result of this process, each cluster generated by PaCE can potentially give rise to multiple consensus sequences. ESTs are then grouped based on the consensus sequence that each corresponds to; henceforth, we will refer to the set of clusters resulting from this grouping as the PaCE clusters. We compared the PaCE clusters against the benchmark clusters. To study how CAP3 behaves stand-alone for EST clustering, we grouped ESTs based on consensus sequences obtained by directly running CAP3 on the benchmark data set, and compared the resulting set of CAP3 clusters against the benchmark clusters. Running of CAP3 on 168 200 ESTs was enabled by running it on a computer with 3.25 GB of available memory.

To assess quality as a function of the data size, two sets of clusters were extracted from the benchmark clusters, such that the number of ESTs represented by the sets are ~50 000 and ~100 000, respectively. The ESTs were input to the programs in no particular order.

Let a set of clusters generated by a program (PaCE or stand-alone CAP3) be referred to as test clusters. To make a comparison between a set of test clusters and the corresponding benchmark set of clusters, we adopted the following approach. For each set of clusters, generate all pairs of ESTs from each cluster, such that both ESTs of a pair are from the same cluster. Based on the number of such pairs generated the following measurements are defined. A pair generated from the test clustering is called a true positive (TP) if it is also paired in the benchmark clustering; it is called a false positive (FP) otherwise. A pair that is not generated from the test clustering is called a true negative (TN) if it is not paired according to the benchmark clustering; it is called a false negative (FN) otherwise. Based on these measurements, another set of quality measures are defined as follows. Overlap quality indicates the ratio of the number of TPs to the total number of unique pairs extracted from clusters of both results, and is given by OQ = TP / (TP + FP + FN); OQ is also known as Jaccard index (17). Specificity is the fraction of correctly predicted pairs with respect to the total number of pairs from the test clustering, and is given by SP = TP / (TP + FP). Sensitivity is the fraction of the correct pairs generated by the test clustering, and is given by SE = TP / (TP + FN). Overall performance is given by the correlation coefficient:

$$CC = \frac{(TP \times TN - FP \times FN)}{\sqrt{(TP + FP) \times (TN + FN) \times (TP + FN) \times (TN + FP)}}.$$

Ideally, OQ = CC = SP = SE = 100%.

The results of assessing the quality of PaCE clusters and CAP3 clusters against the benchmark clusters for different data sizes are shown in Table 2. Observe that the results generated by both programs are close, with CAP3 showing slightly better results than PaCE for larger data sets. We used the following alignment parameters: match score = 2, mismatch score = –5, gap opening = –6 and gap continuation = –1. In addition, we define a minimum threshold on the

Table 2. Quality assessment of PaCE and CAP3 clusters using different portions of the benchmark data set

n	50 012		100 003		168 200	
	PaCE	CAP3	PaCE	CAP3	PaCE	CAP3
OQ	86.87	89.32	84.84	89.13	88.87	90.35
SP	98.67	98.13	96.2	95.62	96.5	96.15
SE	87.91	90.87	87.78	92.92	91.83	93.74
CC	93.12	94.42	91.89	94.26	94.13	94.94

The comparison in either case is done against the corresponding benchmark clusters.

number of bases that should be part of an alignment for it to be acceptable (value of 40) and a minimum threshold on the quality of the alignment (value of 0.75). This quality is measured as the ratio of the obtained score of the alignment to the score that would have resulted if the aligning region were a complete match. For CAP3 we used 75% identity as the quality threshold parameter value. All parameter values are based on the values that were observed to simultaneously optimize sensitivity and specificity of the results. To enable direct comparison with CAP3, we also compared the PaCE clusters to CAP3 clusters. The assessment results for 168 200 ESTs are as follows: OQ = 95.25%, SP = 98.76%, SE = 96.4% and CC = 97.58%.

Figure 3 shows the number of clusters in the PaCE clustering as a function of the cluster size for 168 200 ESTs. There are 34 clusters with size above 200 (not shown in the graph), with the largest size being 1008. A large number of clusters formed contain single ESTs, which we refer to as singleton clusters. Table 3 shows a comparison of the number of singleton and non-singleton clusters as per all the three results for the 168 200 data set. While singletons constitute 36.6% of the benchmark clusters, they constitute 43.1 and 50.5% of the PaCE and CAP3 clusters, respectively.

Quality assessment using clone pair information

As mentioned before, the benchmark data we use is based on spliced-alignment of the *Arabidopsis* ESTs with its genome. Even though this ensures that the ESTs that are mapped to the same benchmark cluster are from the same gene, it does not necessarily imply the converse. For instance, if two sets of ESTs are derived from the two ends of a cDNA transcript, and if none of the ESTs derived from the 5′ end overlap with any of the ESTs from the 3′ end, then the benchmark clustering will place the ESTs in two different clusters instead of one cluster. This is due to lack of coverage of the cDNA transcript by the set of ESTs that are derived from it, and the spliced alignment program does not have enough information to merge the two clusters.

As our original intention is to create a benchmark clustering based on gene homology, we overcame this difficulty by adding available information about cDNA clones from the EST data set in the following way: ESTs are grouped as pairs based on their source cDNA clones (18). Following this, the benchmark clusters are updated by merging the clusters that are linked by at least one clone pair identifier. Note that it might happen that clone information is not available for some of the ESTs in the input data set, or there are ESTs that were originally derived from non-overlapping cDNA transcripts of the same gene. In such cases, the benchmark clustering is based only on spliced alignment results. For the 168 200 ESTs, 16 992 pairs of ESTs were linked with clone pair identifiers.

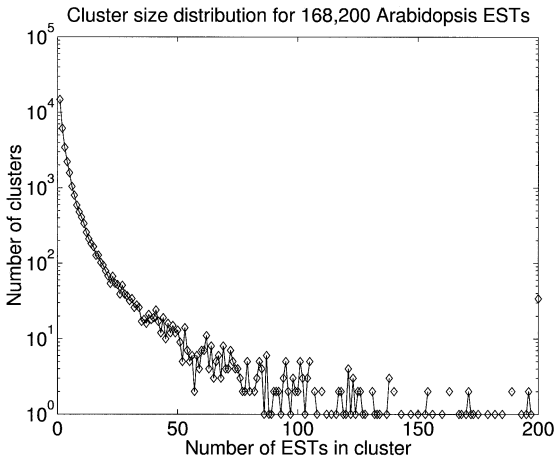

Figure 3. Number of clusters generated by PaCE for 168 200 *Arabidopsis* ESTs as a function of cluster size. There are 34 clusters with size above 200 that are not shown in the graph, with the largest size being 1008.

Table 3. Distribution of the number singleton and non-singleton clusters for benchmark set of 168 200 *Arabidopsis* ESTs

Cluster results	Number of singleton clusters	Number of non-singleton clusters
Benchmark	10 803	18 727
CAP3	17 930	17 556
PaCE	14 802	19 536

PaCE clustering allows for input of clone pair information. Using the aforementioned clone pair information, updated PaCE clusters were obtained in the following manner for the different subsets of the benchmark data sets: cluster the benchmark data set using PaCE with the added input of clone pairs; run CAP3 on each resulting cluster and group ESTs based on consensus sequences. Because CAP3 does not have the clone pair information, the CAP3 step can potentially separate two ESTs that were clustered by PaCE based on clone pair linkage. To overcome this problem, we again merged the clusters resulting from CAP3 based on clone pair information. The results were compared against the corresponding benchmark clusters. To measure the improvement in the quality of clustering arising due to clone pair information, we also compared the PaCE clusters obtained without clone pair information (as explained in the previous section) against the same benchmark clusters.

The results for the data set of 168 200 ESTs and selected portions of it are shown in Table 4. For the 168 200 data set, it was observed that out of a total of 22 029 new benchmark clusters, 5449 are singletons, while in the PaCE clustering obtained with the clone pair information, a total of 29 481 clusters were created, out of which 12 788 are singletons. More importantly, it can be observed from Table 4 that the

Table 4. Quality assessment of the PaCE clusters obtained with and without clone pair information, using different portions of the benchmark data set

n	50 012		100 003		168 200	
	PaCE w/o CP	PaCE w/ CP	PaCE w/o CP	PaCE w/ CP	PaCE w/o CP	PaCE w/ CP
OQ	84.29	88.06	81.94	87.46	85.89	88.74
SP	98.71	97.75	96.28	94.98	96.43	94.94
SE	85.23	89.88	84.62	91.7	88.71	93.14
CC	91.21	93.72	90.26	93.32	92.49	94.04

The comparison in either case is done against the corresponding benchmark clusters updated with the clone pair information. CP, clone pairs; w/, with; w/o, without.

overall quality of the PaCE clusters improved with the addition of clone pair information.

Run-time assessment

We ran our software for various subsets of the *Arabidopsis* benchmark data set using different numbers of processors. The total run-time as a function of the number of processors (denoted by p) is shown in Figure 4a. As can be observed, the run-times show near perfect scaling with the number of processors. The growth of the run-time as a function of the data size for a fixed number of processors is shown in Figure 4b. While the memory required scales linearly with the problem size, the total run-time cannot be analytically determined and depends on the input data set.

A subdivision of run-time into the time spent in each of the two phases for 20 000 ESTs is shown in Table 5. For larger input sizes, the dominant contributor to the total run-time is the clustering phase. The preprocessing phase scales linearly with the number of processors. The clustering phase is expected to take quadratic run-time due to the number of promising pairs generated. However, the run-time spent in doing pairwise alignment is significantly reduced because (i) our pair generation algorithm reduces the number of duplicates generated per promising pair, and (ii) high quality promising pairs are processed first which has a ripple effect of eliminating the need for aligning many other promising pairs. Because of these reasons, for small data sizes the clustering phase runs faster than the preprocessing phase as seen from Table 5.

As an illustration of the capability of PaCE to solve large problems, we clustered 327 632 rat ESTs on 64 processors in under 47 min. The preprocessing phase took ~15 min while the clustering phase took ~32 min.

The total number of promising pairs generated and the number of these pairs assigned for pairwise alignment as a function of the data size are shown in Figure 5. For the 168 200 data set, only 22% of the promising pairs generated are actually assigned for alignment. This clearly explains the reduction in run-time achieved as a consequence of generating the promising pairs in decreasing order of maximal common substring length, as opposed to the traditional way of generating them in an arbitrary order.

The effect of run-time for the clustering phase as the *batchsize* (number of pairs allocated in each batch for pairwise alignment) is varied for clustering 20 000 ESTs on 32 processors is shown in Figure 6. If the *batchsize* is small, the communication overhead between the master processor and slave processors dominates. If the *batchsize* is large the slave processors generate more latency in generating subsequent promising pairs, and also fail to take advantage of the latest clustering information available to determine if an alignment that is assigned is necessary. The optimal *batchsize*, which is expected to increase with increase in the number of processors, can be found experimentally. For the range of processors used in our experiments we found the optimal *batchsize* to be in the range of 40–60. Fixing the *batchsize* and increasing the number of processors used gradually increases the percentage of the total time the master processor is busy. However this is well under 2% even on 60 processors. Thus, using a single master processor is unlikely to be a bottleneck even for a large number of slave processors.

For the run-time results shown here, clone pair information was not included as part of input for PaCE. From our experiments we observed that adding clone pair information to the input improves the run-time. This results from the fact that the program can start with a reduced number of initial clusters (as opposed to starting with each EST occupying an individual cluster of its own).

DISCUSSION

We scrutinized the EST clusters resulting from both PaCE and CAP3 against the benchmark clusters to uncover interesting cases. Many instances where PaCE and/or CAP3 clustering disagree with the benchmark clustering were studied. As a result, various interesting cases such as lack of coverage over a cDNA clone and the effect of clone pair information on the joining of clusters were observed. These observations are illustrated in Figure 7, by showing spliced alignment of the ESTs in question to the *Arabidopsis* genome. In the figures, each blue bounded box denotes a cluster generated by PaCE with the enclosed set of ESTs as its constituents. Similarly, a brown bounded box denotes a CAP3 EST cluster.

Figure 7a illustrates a case of lack of coverage where a set of 5′ and 3′ end ESTs of a transcript do not show full coverage over the transcript. As the primary mechanism of building clusters in PaCE is through detection of significant overlaps, it generates two clusters for the ESTs, one corresponding to each end. However, when the clone pair information of two of the ESTs (one from each end) is provided, PaCE made one cluster containing all the ESTs.

Figure 7b shows a spliced alignment of a set of 12 5′ and 3′ end ESTs, where there exists a pair of 5′ and 3′ end ESTs that overlap. As the overlap is quite significant, PaCE generates one cluster for all the ESTs in the set agreeing with the benchmark clustering. However, CAP3 separated them into two clusters as shown. We observed that this separation occurs only when CAP3 is run directly on the 168 200 data set; if

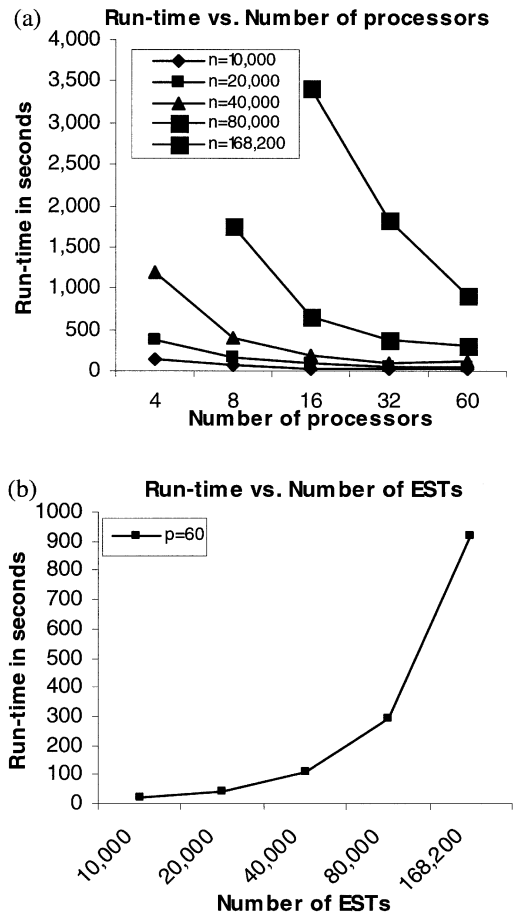

Figure 4. (**a**) Run-times for PaCE as a function of the number of processors. (**b**) Run-times for PaCE as a function of data size for a fixed number of processors.

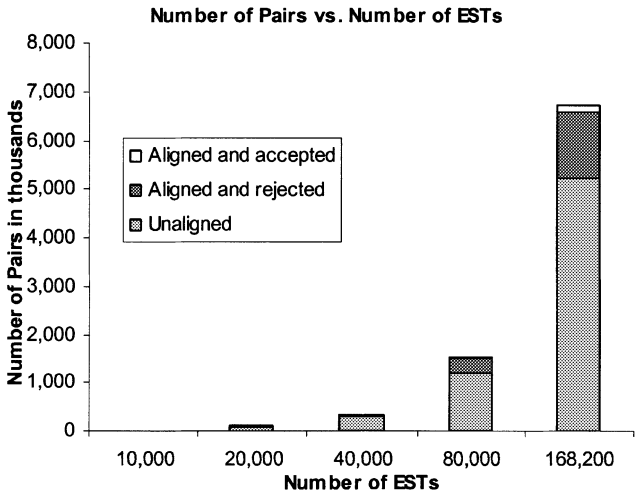

Figure 5. The number of promising pairs generated and the number of pairs aligned as a function of the data size. Also shown is the numbers of pairs actually responsible for the generated clustering ('Aligned and accepted').

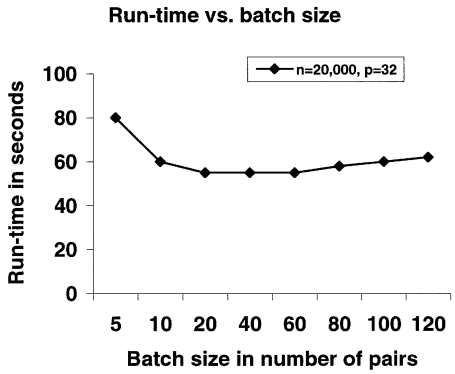

Figure 6. Variation in PaCE run-time as a function of the number of pairs allocated for pairwise alignment per communication.

Table 5. Run-time (s) spent in various components of PaCE for 20 000 ESTs

p	Preprocessing	Clustering	Total
4	273	102	375
8	119	50	169
16	61	26	87
32	38	15	53
60	29	10	39

p, number of processors.

CAP3 is run on only these 12 ESTs as input then it puts them together. As the clusters resulting directly resulting from PaCE put these 12 ESTs in one cluster, the final PaCE clustering after running CAP3 on each PaCE cluster, also had these together.

Figure 7c shows a case where a set of 5′ and 3′ end ESTs show some overlap, but PaCE separates them into two different clusters. This is because the overlap quality is not significant according to the threshold used in PaCE. CAP3, on the other hand, places all the ESTs into one cluster, agreeing with the benchmark clustering.

In the case of an EST that shows strong alignment to multiple gene sources, PaCE may group all the ESTs from these gene sources into a single cluster. The cluster is subsequently broken up during the assembly process. For instance, the 168 200 benchmark data set includes 21 673 ESTs aligning to multiple locations. Running PaCE on this benchmark generated 33 310 clusters. Subsequent to assembly, the number of clusters generated were 34 338, resulting in an increase of 1028 clusters.

Capabilities and flexibility of our software

The performance results of our software illustrate that fast and accurate clustering of large EST data sets with feasible memory requirements is no longer impossible. By experimentation, we found that the current version of PaCE is capable of handling up to 20 000 ESTs per processor equipped with 512 MB of RAM each. Based on this data and the fact that the memory requirement for PaCE scales linearly with the data size, our estimate suggests that it can cluster the human

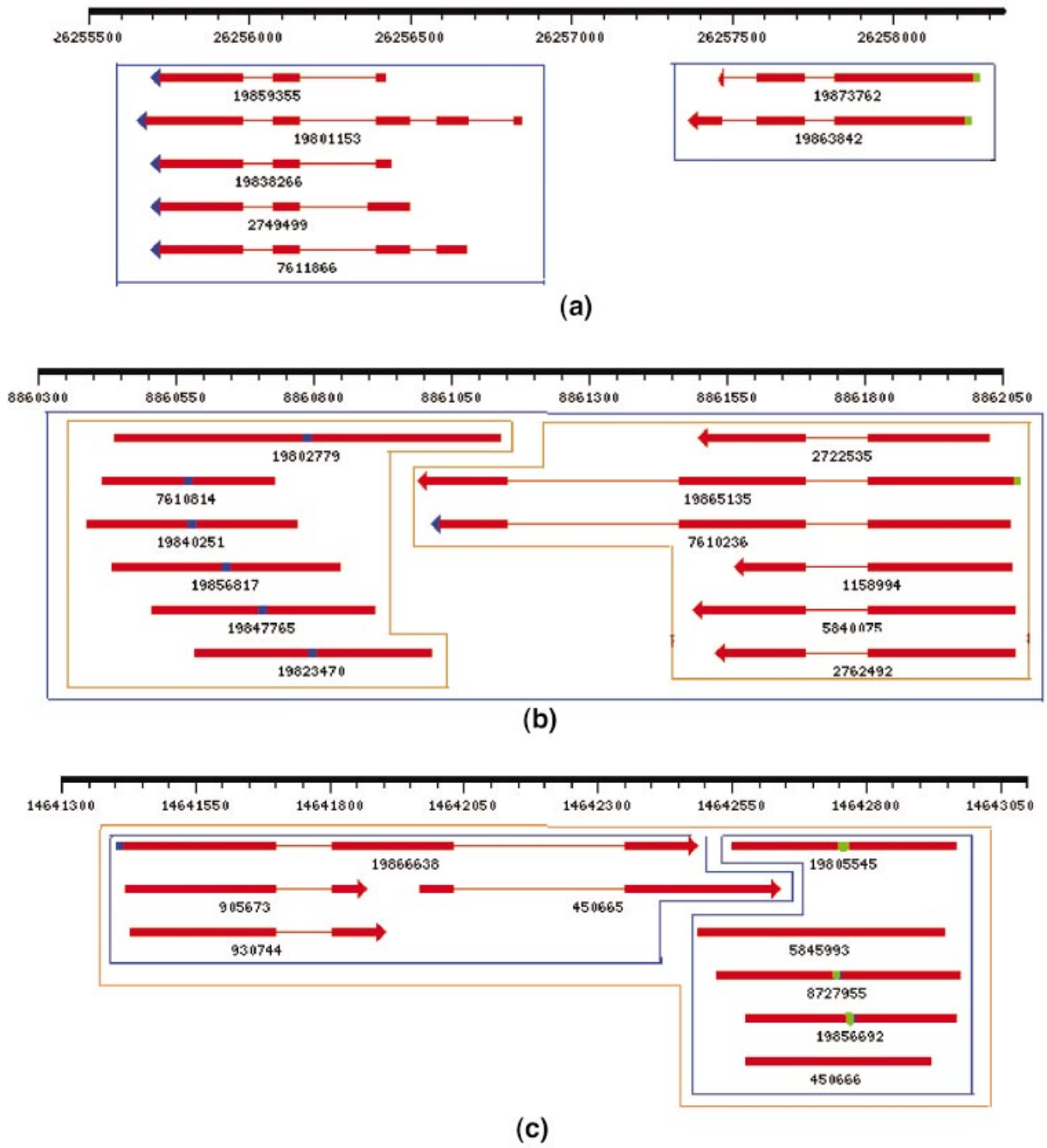

Figure 7. Schematic depiction of spliced alignment of ESTs to the *Arabidopsis* genome, as an illustration of the various cases studied to enable understanding the behavior of our software. ESTs are identified by GenBank gi number. Filled boxes represent exons and thin lines represent introns. Arrows identify the 3′ ends of the ESTs. The multi-exon 5′ ESTs are marked by green color at their 5′-terminus, and the multi-exon 3′ ESTs are marked by blue color at their 3′-terminus. The scale refers to chromosome positions as described at AtGDB (http://www.plantgdb.org/AtGDB/). ESTs clustered by PaCE are enclosed in blue-lined boxes, and ESTs clustered by CAP3 are enclosed in brown-lined boxes. (**a**) Illustration of lack of coverage shown by the 5′ and 3′ end ESTs corresponding to a cDNA transcript (ESTs corresponding to At5g66470, chromosome five). Without clone pair information, PaCE clustered the ESTs into two clusters, but when the clone pair information of 19859355 and 19873762 was made available, all ESTs were put into one cluster. (**b**) Illustration of a case where the 5′ and 3′ end ESTs of a gene have overlap (ESTs corresponding to At2g20670, chromosome two). PaCE correctly put the ESTs into one cluster, while CAP3 separated them into two clusters as shown. (**c**) Illustration of a case where ESTs corresponding to a gene are partitioned into two clusters by PaCE (ESTs corresponding to At4g32480, chromosome four). Even though the figure shows an overlap between the ESTs 450665 and 5845993, the pair was not accepted by PaCE, because its quality was below the defined threshold. CAP3 put all these nine ESTs in one cluster.

EST collection (~5 million ESTs) using 512 such similarly equipped processors. In our experiments, the largest data size clustered by PaCE is 420 694 ESTs for *Triticum aestivum*.

Apart from its performance capabilities, PaCE is adaptable to incorporating various functional refinements to improve the quality of clustering. The software can be tailored to use

Table 6. Distribution of the number of singleton and non-singleton clusters for ESTs from 23 plant species, as generated by PaCE clustering followed by CAP3 assembly

Plant species	Number of ESTs	Number of singleton clusters	Number of non-singleton clusters
Avena sativa	501	257	100
Arabidopsis thaliana	176 911	19 530	19 300
Beta vulgaris	6034	2930	926
Glycine max	284 714	29 290	24 180
Gossypium arboreum	38 894	11 995	5410
Gossypium hirsutum	9461	4395	1272
Hordeum vulgare	262 138	34 365	19 040
Lotus japonicus	32 096	6718	3794
Lycopersicon esculentum	148 358	13 259	14 914
Lycopersicon hirsutum	2504	1393	328
Lycopersicon pennellii	8346	2360	914
Marchantia polymorpha	1415	872	175
Medicago sativa	719	538	68
Mesembryanthemum crystallinum	17 190	5190	2031
Oryza sativa	108 547	21 714	12 151
Pinus taeda	60 226	12 949	7209
Populus tremula × *Populus tremuloides*	20 084	7014	2565
Secale cereale	8930	3798	1267
Solanum tuberosum	94 420	6653	15 986
Sorghum bicolor	84 712	12 242	11 441
Sorghum propinquum	21 387	5218	3530
Triticum aestivum	420 694	58 776	27 928
Zea mays	196 245	13 887	18 721

organism-specific information, such as masking repeat sequences in pair generation. The software can also be extended to predict alternative splicing sites and the frequency of such splicing events in the genome. On the other hand, if organism-specific splice site information is known, then such knowledge can be used to detect alternative spliced isoforms more accurately.

The software is capable of handling ESTs from either end of cDNA clones. This is necessary because, while a majority of the ESTs are derived from the 3′ end, publicly available EST databases also have a significant portion of the ESTs derived from the 5′ end. For instance, of the 327 632 rat ESTs we downloaded from dbEST, ~20% were found to be from the 5′ end. The software can also accept full-length cDNA sequences as part of the input. If made available, the addition of cDNA sequences can detect cases where there is lack of sufficient coverage of cDNA clones by the ESTs, and can improve the sensitivity of the results.

As running the software for large data sets takes less time, the effect of various parameters on EST clustering can be observed by repeated executions and an optimal set of parameters can be determined. The variation in the sensitivity and specificity of the results can be studied by adjusting the set of parameters mentioned in the Results section. Run-time can be tuned by adjusting the *batchsize* parameter based on the number of processors. The software has an in-built load balancing capability during both the preprocessing and clustering phases.

Clustering plant ESTs

As part of an ongoing effort in the Brendel group to compare the gene spaces of different plant species, we clustered EST data sets from 23 species ranging in size from 501 ESTs (*Avena sativa*) to 420 694 ESTs (*T.aestivum*). The results are summarized in Table 6. After clustering each EST data set with PaCE, we ran CAP3 on each resulting cluster to generate a tentative unique contig for the cluster. The multiple alignment of the ESTs in a cluster and the corresponding tentative unique contig may be visually inspected at PlantGDB (http://www.plantgdb.org). Biological analysis of the results will be presented elsewhere. (We automated the process of running CAP3 on each of the clusters resulting from PaCE clustering, and the scripts are also available with the software.)

Conclusion and future research

Our overarching goal has been to facilitate fast and accurate clustering of large EST data sets, which is accomplished through the use of memory-efficient algorithms, algorithmic heuristics and high-performance parallel computing. To summarize, our algorithms achieve the following: (i) reduce the worst-case memory requirement from quadratic to linear, (ii) generate promising pairs in decreasing order of maximal common substring length and cluster the ESTs such that the number of pairwise alignments is reduced by an order of magnitude without affecting the quality of clustering, and (iii) reduce the number of duplicates generated for each promising pair.

We are currently working on various improvements to PaCE. With current distribution of our software, we include scripts to automate the process of applying an assembly program to each cluster generated by PaCE. Our first goal is to extend PaCE to do assembly and build consensus sequences in parallel. This will further simplify the use of PaCE and bring advantages of high performance parallel computing to the assembly phase. We also plan to incorporate quality values available to ESTs as part of input, to ensure quality clustering and assembly.

Tackling lack of coverage is important to improve the sensitivity of the software. Even though the current version of the software is capable of addressing this issue by accepting clone pair information and cDNA sequences as part of input, we intend to improve sensitivity further by consulting protein databases to see if two ESTs that do not overlap encode different parts of the same protein, and if so, combine the clusters containing them. We are evaluating various means to further improve specificity of the software. One approach is to require multiple overlapping pairs as evidence for merging clusters. We also plan to expand the knowledge base of our software by incorporating organism-specific information such as repeat sequences and splice sites and thus enhance the capability of our software.

We are also working on further improving the run-time performance of our software. We are exploring the use of alternative data structures such as suffix arrays, to reduce memory usage without affecting the run-time significantly. Further reduction in run-time can be achieved by not generating promising pairs that correspond to ESTs that have already been clustered, instead of the current method of generating and discarding them.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers, whose reviews were very useful in improving our evaluation methodology. We thank Richa Agarwala, Alejandro Schäffer, Greg Schuler and Jean Terry-Mieg at NCBI for suggestions in improving the system and identifying many useful extensions of this work. We thank Natsuhiko Futamura for contributing some useful implementation techniques in reducing memory requirements. We thank Shannon Schlueter for his help in providing the benchmark data sets and help with Figure 7. Our sincere thanks to Qunfeng Dong for developing the web interface to illustrate our plant EST clustering results at the PlantGDB website. We thank Todd Vision for his comments on an earlier version of this manuscript. This research is supported by NSF under ACI-0203782 and DBI-0110254. S.A. is also supported by NSF CAREER Award under CCR-0096288.

REFERENCES

1. Bouck,J., Yu,W., Gibbs,R. and Worley,K. (1999) Comparison of gene indexing databases. *Trends Genet.*, **15**, 159–162.
2. Schuler,G.D. (1997) Pieces of the puzzle: expressed sequence tags and the catalog of human genes. *J. Mol. Med.*, **75**, 694–698.
3. Sutton,G., White,O., Adams,M. and Kerlavage,A. (1996) TIGR Assembler: a new tool for assembling large shotgun sequencing projects. *Genome Sci. Technol.*, **1**, 9–19.
4. Huang,X. and Madan,A. (1999) CAP3: a DNA sequence assembly program. *Genome Res.*, **9**, 868–877.
5. Quackenbush,J., Liang,F., Holt,I., Pertea,G. and Upton,J. (2000) The TIGR Gene Indices: reconstruction and representation of expressed gene sequences. *Nucleic Acids Res.*, **28**, 141–145.
6. Miller,R.T., Christoffels,A.G., Gopalakrishnan,C., Burke,J., Ptitsyn,A.A., Broveak,T.R. and Hide,W.A. (1999) A comprehensive approach to clustering of expressed human gene sequence: the sequence tag alignment and consensus knowledge base. *Genome Res.*, **9**, 1143–1155.
7. Coward,E., Hass,S.A. and Vingron,M. (2002). SpliceNest: visualizing gene structure and alternative splicing based on EST clusters. *Trends Genet.*, **18**, 53–55.
8. Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search of similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.
9. Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
10. Burke,J., Davison,D.B. and Hide,W. (1999) d2_cluster: a validated method for clustering EST and full-length cDNA sequences. *Genome Res.*, **9**, 1135–1142.
11. Zhu,W., Schlueter,S.D. and Brendel,V. (2003) Refined annotation of the *Arabidopsis thaliana* genome by complete EST mapping. *Plant Physiol.*, in press.
12. Liang,F., Holt,I., Pertea,G., Karamycheva,S., Salzberg,S. and Quackenbush,J. (2000) An optimized protocol for analysis of EST sequences. *Nucleic Acids Res.*, **28**, 3657–3665.
13. Gusfield,D. (1997) *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, London.
14. Tarjan,R.E. (1975) Efficiency of a good but not linear set union algorithm. *J. ACM*, **22**, 215–225.
15. Setubal,J. and Meidanis. J. (1997) *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston, MA.
16. Pacheco,P. (1997) *Parallel Programming with MPI.* Morgan Kaufmann.
17. Jain,A.K. and Dubes,R.C. (1988) *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ.
18. Seki,M., Narusaka,M., Kamiya,A., Ishida,J., Satou,M., Sakurai,T., Nakayima,M., Enju,A., Akiyama,K., Oono,Y. *et al.* (2002) Functional annotation of a full-length *Arabidopsis* cDNA collection. *Science*, **296**, 141–145.