Algorithms for the Analysis of Expressed Sequence Tags

Thesis submission for the degree of *doctor scientiarum*

Ketil Malde

Department of Informatics, University of Bergen, 2005

PROMOTER

> I don't know any field where they are making more rapid progress than they are in biology today. — *Richard Feynman*

Bioinformatics is an emerging science in the borderland between biology and computer science (informatics). Biology, and in particular molecular biology, has experienced its own industrial revolution since the 1950s, and current laboratories are filled with advanced, automated equipment producing a continuous stream of data.

While computer scientists will often be skeptical of claims of rapid progress from other sciences, the industrialization of biology is resulting in exponential growth in the data bases. (This is not unlike Moore's law, which stipulates that the number of transistors that can be put on a chip doubles at regular intervals.) The sheer quantities of data being produced require an equally industrialized approach to analysis, and the science of bioinformatics is aimed at developing the necessary tools; databases, software tools, statistical methods, and algorithms.

Like the industrial revolution itself, the use of technology was perhaps primarily thought of as a tool to provide better efficiency in the established methods, but is now opening up new possibilities not previously attainable – or even imagined.

As the databases and the computations we want to perform on them grow in size and complexity, the demand for computational power increases proportionally. The supercomputers now being built to attack bioinformatics problems rival those that intended for more traditional pursuits – like the analysis of nuclear weapons. Perhaps some day we will reach the important milestone where *Homo sapiens* spends more resources on improving people's lives than on destroying them.

ACKNOWLEDGMENTS

First and foremost, I wish to thank the University of Bergen and the Department of Informatics for giving me the opportunity to pursue a Ph.D., as well as the Salmon Genome Project of the Norwegian Research Council, who has provided my scholarship for this work.

In particular, I have appreciated my advisors, Inge Jonassen and Eivind Coward, who have always found time in (and in a pinch, outside of) their otherwise very busy schedules to support me when it has been necessary. At the same time, I have enjoyed the freedom to pursue my own whims and interests to a large degree.

I am also grateful to my colleagues at the Department of Informatics and at CBU, who are always prepared to answer my naive questions as I try to learn molecular biology by osmosis. In particular, I must mention Trond Hellem Bø for his insightful comments and temperate responses to my frequent interruptions of his work, and Pål Puntervoll, for proof–reading and advice on the biology–related sections. I don't know who puts on the coffee in the morning, but I do appreciate that, too. Thanks go to Craig Jonathan for weeding out the worst abuses of the English language, any remaining offenses (as well as the deleted ones, of course) are solely my own fault.

I've had the pleasure of collaborating on two of the papers; first with Korbinian Schneeberger during his visit to Bergen, and then with Alexander Sczyrba, during my stay in Bielefeld. I would also like to thank Robert Giegerich and the "Praktische Informatik" group at the University of Bielefeld, and Martin Vingron and his department at the Max Planck Institute for Molecular Genetics in Berlin for their hospitality and support during my visits. I am looking forward to visiting them again, should the opportunity arise.

It is a well–known dogma in science that the important thing isn't getting the right answer, but asking the right questions. While the right questions may be a good idea, I find it even more stimulating to be asked the wrong questions. Children have a propensity for mixing the literal with the figurative, applying logic where it doesn't apply, and arriving at the most marvelous conclusions. So to Hedvig, Eyvind, and Oda: thanks for the inspiration!

Finally, my gratitude goes to my wife, Marian, for her encouragement and support along the way, especially at the times it seemed more like a blind alley.

Ketil Malde, Bergen 2004

If I haven't seen further, it is by standing in the footprints of giants

Contents

SUMMARY

In this thesis, I focus on aspects of the EST analysis pipeline, including the development and implementations of novel algorithms for repeat detection and masking, clustering, and assembly.

I introduce central concepts in molecular biology at a level that should be comprehensible and useful to a computer scientist, including detailed descriptions of the mechanisms involved as information is passed from genes via mRNA to proteins, the processes involved in sequencing nucleic acids (in particular as it applies to mRNA), and the pipeline for computational analysis of EST sequences. Particular emphasis is placed on the various errors and artifacts that can arise in the process and how they affect the analysis of sequence data.

I also include a brief introduction to data clustering and suffix–based data structures to the level necessary to understand the papers; for more complete treatment, the reader should consult the literature.

Repeat masking is an important step in the EST analysis pipeline. I discuss some of the problems with the traditional approaches, and the development of a novel algorithm that identifies repeats without the use of an external database. The algorithm is tested on ESTs from a variety of species, with good results.

For EST clustering, both an automatic and a genome–based EST clustering algorithm are described. The automatic algorithm is implemented in the publicly available tool `xsact`, and benchmarks show that it produces high–quality clusters in short time compared to contemporary methods. While a good benchmark data set is crucial for comparing clustering quality, no such sets are available. With the intent of obtaining such a benchmark data set, we construct a genome–based clustering and validate it against the annotated genes. We compare the result against UniGene, and against some of the available clustering tools.

Finally, I develop a sequence assembly algorithm, implemented in the `xtract` tool. The algorithm is tested by comparing output to full–length mRNA sequences. The results show that this algorithm can produce higher quality contigs than other tools in common use (CAP3, Phrap, and TIGR assembler).

AVAILABILITY

`http://www.ii.uib.no/~ketil/bioinformatics`

BACKGROUND

1.1 GENES AND GENOMES

Gregor Mendel is usually considered the father of genetics for his systematic and extensive experiments on hereditary traits in pea plants in the middle of the 19th century [55]. In 1909 the Danish botanist Wilhelm Johannesen coined the term "gene" to describe a factor that confers a heritable trait, or the basic unit of heritable information.

The word "gene" is used differently depending on context. In molecular biology, it is often used to refer to the transcribed region (often including the promoter) of a protein coding sequence. For our purposes, we define a gene to be any region of the genome that is transcribed to RNA.

1.1.1 The central dogma

The basic model states that the genetic information, describing the makeup of the cell, and by extension, the complete organism, is stored in the cell as DNA. In *eukaryotes* (organisms whose cells have a cell nucleus), this DNA is located in the cell's nucleus (organelles like mitochondria and chloroplasts also have DNA containing genes). DNA is duplicated during cell division, leaving both descendant cells with replicate copies of the entire genome. The protein–coding genes in the DNA are transcribed to *messenger RNA* (or *mRNA*), and the mRNA in turn is translated into protein. This transfer of information between different kinds of molecules is referred to as the *central dogma* [13, 14].

Proteins consist of sequences of amino acids, and the mRNA contains the information for constructing them. Nucleotide triplets, or *codons* in the mRNA sequence encode the amino acids of the corresponding protein. The mapping from nucleotide triplets to amino acids is called the *genetic code*.

Most of the machinery that makes a cell work consists of proteins, and the responsibilities of proteins encompass a number of diverse roles, including structural functions, transport, signaling, and enzymes.

1.1.2 The basics

DNA (deoxyribonucleic acid) and RNA (ribonucleic acid) consist of long chains of *nucleotides*. For DNA, the nucleotides are adenine, cytosine, guanine and thymine, giving the alphabet `A`, `C`, `G`, and `T`. Each nucleotide (or *base*) has the ability to form a bond with one of the others (`A` to `T`, `C` to `G`), which gives DNA its common double–stranded structure. The forming of double strands is often referred to as *annealing* or *hybridization*, while splitting up a double stranded

DNA is termed *denaturation* or *melting*. In RNA, thymine is replaced by uracil (U), which has the ability to bond to both adenine and guanine. RNA is commonly single stranded, but often forms a secondary structure with different parts of the RNA bonding. Strands of RNA and DNA have defined *directions*, with a 5' end and a 3' end, and hybridization joins two strands going in opposite directions, and containing the complementary nucleotides (*reverse complement*).

As there are 20 amino acids, protein sequences are described with a correspondingly larger alphabet.

1.1.3 Transcription

The transcription of genes from DNA to RNA requires a protein complex called *RNA polymerase*. Prokaryotes have a single RNA polymerase responsible for transcribing all genes, but in eukaryotes, there are three different RNA polymerases, named pol I, pol II, and pol III. Only pol II is responsible for transcribing protein–coding genes. Transcription is illustrated schematically in Figure 1.1.

Protein coding genes contain a *poly–A signal* near the end, and this will initiate the attachment of a long (on the order of 200 nucleotides) sequence of adenines. This process is called *polyadenylation*, and the attached adenines are referred to as a *poly–A tail*.

The expression of a gene is controlled by signals in a *promoter* region that precedes the gene. RNA polymerase requires some additional enzymes, called *transcription factors*, that bind to *transcription factor binding sites* that typically reside in this region. A large collection of known transcription factor binding sites can be found in the TRANSFAC database [83].

DNA is normally densely packed in the chromosomes in a form called *chromatin*, where the DNA is tightly wound around histone proteins in structures called *nucleosomes*. There is therefore an additional regulatory mechanism that serves to expose the genes for transcription. This consists of a large protein complex called the *mediator* which serves to "unwind" the DNA, exposing the gene. The mediator is aided by smaller proteins called *activators* that bind to specific sites called *enhancers*.

1.1.4 Splicing

Many eukaryote genes have a complex structure, consisting of alternating blocks called *exons* and *introns*. The exons contain the information for the final transcript, while the introns are removed from the transcript in a process called *splicing*. Figure 1.1 shows the unspliced sequence, called a *pre–mRNA*[1], and the corresponding mRNA that result from the splicing process. Exons can be very short [81], but introns appear to have a certain minimum length [87].

Splicing is most commonly performed by a complex consisting of proteins and *snRNPs* (small nuclear ribonucleoprotein particles), called the *spliceosome*.

The exon–intron and intron–exon boundaries are often called *donor* and *acceptor* sites, respectively. They are characterized by specific patterns of nu-

[1] Strictly speaking, it is not necessarily pre–*mRNA*, as some genes for tRNA and ribosomal RNA also have introns, and thus go through splicing process.

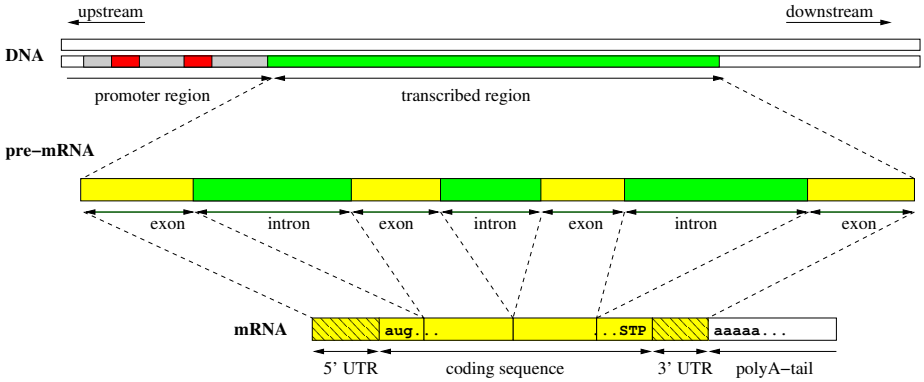

Figure 1.1: The structure of a gene. A region of the genome is transcribed to a pre–mRNA. A poly–A tail is added, and the splicing process removes introns to produce the final mRNA.

cleotides, most common are introns that start with GT or GC and end with AG, but variations exist.

The splicing process consists of "bending" the intron, attaching the donor site (the 5' end or the intron) to an adenine close to the acceptor site (3' end). The intron thus forms a loop with a short tail (this is called a *lariat*, since the shape resembles a lasso), and this brings the 3' end of the upstream exon close to the 5' end of the downstream exon, so that they can be joined.

Some introns have the ability to perform splicing without the spliceosome. There are two variants, one forms a lariat, and the other leaves a linear intron. This form of splicing is rarer, and depends on the introns being conserved to a much larger degree than the spliceosome–mediated splicing process.

In many cases, the splicing process is regulated to produce a variety of transcripts from a single gene [57, 58]. The spliceosome can recognize an alternative donor or acceptor site, resulting in alternative ends for an intron, in effect producing an exon of variable length. The splicing process can also take the donor site of one intron and attach it to the acceptor site of another, removing one or more exons as well as the introns. Another possibility that can occur is that an intron is retained in the mRNA, instead of being removed.

A single gene can in this manner give rise to several different mRNA sequences. These are referred to as *splice variants* or *alternative isoforms*. The exons that are sometimes excluded in the splicing process are called *alternative exons*.

Alternative isoforms can also arise by differences in transcription, where transcription is either started or ended at an alternative position, resulting in an alternative 5' and 3' end, respectively.

The splicing process described above is more accurately referred to as *cis–splicing*. There is also another form of splicing called *trans–splicing*, where one pre–mRNA is spliced to a pre–mRNA from a different gene [34]. Trans–splicing is thought to be rare (but not unknown, see e.g. Caudevilla *et al.* [10]) in higher organisms, but common in lower eukaryotes (e.g. in the nematode *C.elegans*, all genes are trans–spliced [71]).

There is also a mechanism called RNA editing that modifies single nucleotides

of the mRNA or inserts short sequences. This takes place after splicing, and is usually strongly regulated.

1.1.5 Translation

The finished mRNA is exported from the nucleus, and is translated to protein in the cell's *ribosomes*. The region of the mRNA that contains the blueprint for the protein sequence, is called the *coding sequence* (abbreviated *CDS*) of the transcript, and it starts with an AUG codon (representing the amino acid methionine), and ends with one of three *stop*, or nonsense, codons.

A grouping of nucleotides in a sequence into triplet codons is called a *reading frame*. There are three possible reading frames — six if you consider reverse reads. An *open reading frame*, or *ORF*, is a reading frame that does not contain a stop codon.

1.1.6 Pseudogenes

Pseudogenes are genes that the process of evolution has rendered without function. Pseudogenes usually arise either from gene duplication events or from retrotransposition, the latter being the more common cause [86]. When gene duplication creates redundant copies of a gene, subsequent mutations can eliminate one of the copies without loss of evolutionary fitness, and a pseudogene results.

In most cases, mutations in the region of the gene hinders transcription, but sometimes pseudogenes are transcribed, and the transcript is either not translated, or translated into a dysfunctional protein sequence [56].

The presence of pseudogenes presents a challenge to similarity searches in the genome as well as other kinds of sequence analysis, as pseudogenes will often be very similar to existing genes.

1.2 EXPRESSED SEQUENCE TAGS — ESTS

An *expressed sequence tag*, or *EST!definition*, is a sequence that (ideally) represents a fragment of an mRNA transcript. Early on, expressed sequence tags were intended as a means to identify clones of mRNA transcripts, and to map genes to positions in the genome [3, 63, 68].

1.2.1 The sequencing process

While variations exist, ESTs are generally produced using the following process.

i. Reverse transcriptase produces cDNA from the mRNA

Reverse transcriptase reads RNA and, using it as a template, constructs a complementary DNA molecule.

ii. Amplification with PCR

Polymerase chain reaction is used to amplify the cDNAs. Unlike RNA polymerase, DNA polymerase needs a short starting sequence called a *primer* to start the process. An enzyme called a *terminal transferase* is used to

attach a poly–C tail to the cDNAs, so that an oligo–dG primer can be used in replication [62]. Alternatively, arbitrary primers can be used [49].

In the PCR process, sequences that resemble each other's reverse complements can anneal, and function as primers for the polymerase. The result is a clone containing parts of both sequences. This is referred to as a *chimera*.

iii. The cDNA is inserted into a plasmid

Small, circular DNA molecules, called *plasmids* are cut using *restriction enzymes*, and the amplified cDNA is inserted, using *DNA ligase*.

iv. The plasmid is inserted into a vector organism

Plasmids can be inserted into bacteria (often *E.coli*), and the bacteria grown on a substrate. When the bacterium divides, the genetic material in it, including the plasmid, will be replicated. The bacteria are distributed thinly enough that the descendants of a single ancestor bacterium can be separated. From each of these cultures, multiple copies of one original plasmid can be harvested.

v. Random length transcription

A primer is used as the starting point for polymerase to generate nucleotide sequences of random lengths. The primer is usually selected from the plasmid sequence, close to the inserted cDNA. Ideally, this will result in a sequence starting at either the 5' or 3' end of the insert. Random termination is achieved by using a small amount of dideoxyNTP, which are similar to nucleotides, but lack the ability to continue transcription. When by chance a ddNTP is used by the polymerase instead of a normal NTP, transcription will be terminated.

vi. Polyacrylamide gel electrophoresis

A polyacrylamide gel allows diffusion of molecules. The speed of the diffusion varies with molecule size, and this effect is used to separate the transcribed molecules by length.

vii. Read the sequence from labels

Finally, the gel is scanned, and the labeled molecules are identified. If the ddNTP terminating the sequences are tagged with different labels, a single column is sufficient [69][2]. The final output is usually a *chromatogram*, showing a set of oscillating curves derived from the intensity of the labels. Each curve thus represents the presence of one particular nucleotide in the sequence.

1.2.2 Primer walking for full–length sequencing

The method for producing ESTs can be modified to produce the sequence for the full length of the clone. First, the start of the sequence is determined using the

[2] A alternative method no longer in common use, is to use four different reactions, each producing transcript terminating in one particular nucleotide, and use four different columns, interleaving the results [26, 7].

above method. Then, a new primer can be designed to hybridize near the end of the sequenced region, and the process can be reiterated from Step *v*. Similarly, a new primer can then be designed to extend this sequence, and so on, until the complete sequence is determined. This process is more cumbersome than simple EST sequencing, but has been used to good effect e.g. to identify human genes [36].

1.2.3 Some notes on ESTs

For organisms where the genome sequence is still unknown, ESTs provide an inexpensive and readily available source of genetic information. As the EST databases grow in size, ESTs can be used to reconstruct the gene sequences, as well as identify sequence variation. It is therefore still an invaluable tool, even after the genome has been sequenced.

Genes are very unevenly expressed, and many genes are only expressed in certain tissues or states. While there exist methods for *library normalization*, it is difficult to completely identify all genes of an organism solely from ESTs [6].

By their nature, ESTs do not contain information about introns (except for possibly retained introns) or regulatory elements for the genes residing outside the transcribed region.

1.3 THE EST ANALYSIS PIPELINE

As we have seen in Section 1.1, the transcription and splicing processes can often cause sequence variations. In addition, the sequencing process can introduce errors and artifacts. Finally, there can be difficulties in the sequences themselves. These factors must be taken into account when the sequences are analyzed.

The various steps in EST analysis are illustrated in Figure 1.2.

1.3.1 Sequencing and base calling

In addition to the natural sequence variation described above, artifacts can be introduced by the sequencing process itself. Base calling is the process of interpreting chromatograms, yielding the sequence of letters representing the nucleotides. In addition to `A`, `C`, `G`, and `T`, there are also additional characters defined, representing one of several possible nucleotides. Only `N` (matching aNy nucleotide) is commonly used. An associated quality measure is often provided for each position in the sequence [17]. The most commonly used base calling tool is `Phred` [18].

After the sequences have been generated, metadata is usually added, in the form of *sequence annotations*. Commonly, clone–ID, read end (5' or 3'), sequence type (full–length or EST) are included.

The chromatogram typically starts close to the primer with low initial quality. Near the end (after roughly 500–800 bases) the quality gradually degrades as the difference in mass between molecules, as well as the number of them, diminishes. Additionally, the sequencing process will often introduce random errors as single nucleotides are incorrectly read due to noise.

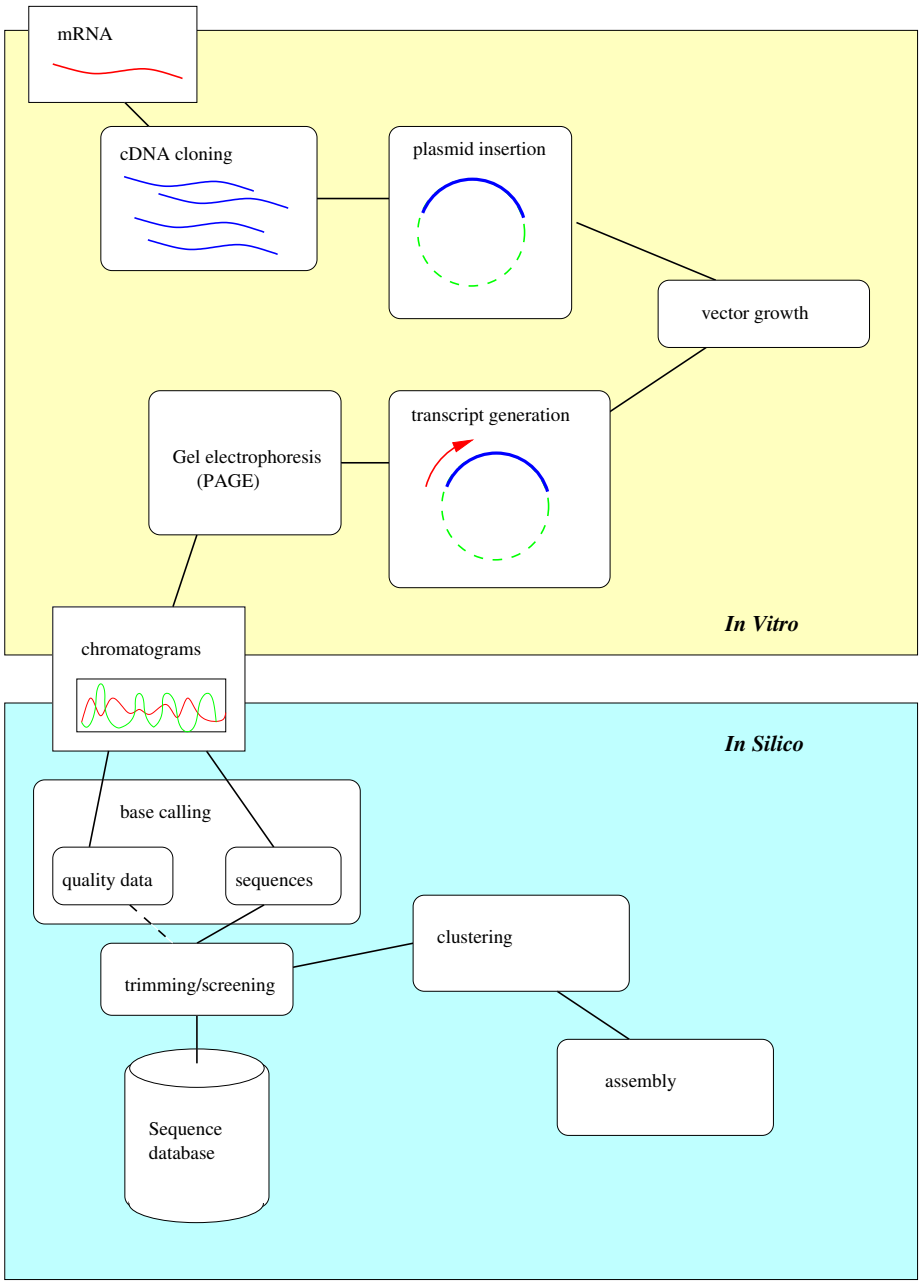

Figure 1.2: The EST production process, and the computational EST analysis pipeline.

Natural Sequence Variations

Feature	**Type**
Single–nucleotide polymorphisms (SNPs)	Genomic variation
Alternative starting and ending exons	Transcriptional variation
Alternative starting or ending positions in exons Alternatively skipped exons Retained introns	Splicing variations
Modified bases Short inserted sequences	RNA Editing

Sequencing Artifacts

Feature	**Type**
Read errors Stuttering Quality degradation	Sequencing artifacts
Chimeric sequences Vector sequence Genomic sequence	Contamination

Problematic Sequence Features

Feature
Low–complexity regions
Simple repeats
Tandem repeats
Repeats
Homologous genes/gene families
Low coverage
Sense–antisense transcription

Table 1.1: Sequence variations, errors and artifacts, and inherent sequence features that cause problems for sequence analysis.

The sequencing is also prone to *stuttering*, as stretches of repeated nucleotides (in particular Gs or Ts) can cause the polymerase to slip back and reprocess a small part of the sequence. When this happens in Step *v* of the sequencing process (see page 7), the result can be small insertion in the sequence [88, 62] and/or quality degradation of the following sequence [18].

If the cDNA forms a secondary structure, this can affect its speed through the gel, and distort the resulting sequence.

1.3.2 Cleaning up the sequences

Before further analysis can be performed, it is necessary to remove any artifacts and contamination. This usually includes *quality clipping* or *trimming* of the sequences, that is, removing or masking the parts at the start and end where the sequencing process has produced unreliable results.

The sequencing often includes, in addition to the RNA sequence, part of the vector (plasmid) sequence used in the cloning process. As with the other artifacts, these vector sequences must be masked out. This is normally done by either comparing to the exact vectors used, or, if the vector is unknown, comparing to the genomes of common vector organisms.

In addition to the errors introduced in the laboratory processes, there are naturally occurring sequence features that can cause problems for subsequent analysis, listed in Table 1.1. The sequences are therefore normally scanned for simple repeats (short, repeated sequences) and low–complexity regions, and repeats (sequences that occur in multiple genes) are masked. If left in the sequences, these features will cause difficulties in the clustering stage (see the next section), since they will tend to match transcripts from other genes, and thus tend to merge clusters that represent different genes.

`RepeatMasker` [77] is a commonly used tool for cleaning up sequences before clustering. It will mask regions of sequences containing a high proportion of some nucleotides (low–complexity regions), and regions consisting of short, repeated words (simple repeats). Finally, it uses `crossmatch` to search a database of vector sequences and known repeats, and mask any occurrence of these. `crossmatch` uses Smith–Waterman sequence alignment with affine gap penalties [28]. This algorithm is slow, in particular for large databases. An alternative is to use a faster tool like `vmatch` [2] for this purpose.

1.3.3 Clustering

The masked sequences can now be grouped together according to the originating gene[3]. This can be determined by matching against the gene's sequence or by matching against the genome and using the genomic position for clustering. Since the genome is unavailable for many organisms, and perhaps also because it is computationally less intensive for smaller data sets, it is more common to compare the ESTs against each other. Automatic EST clustering is commonly done by defining a similarity function, a threshold, and performing agglomerative single–linkage clustering (see Section 1.4).

[3] Another possible goal is to group sequences according to originating transcript – i.e. separating splice variants into different clusters. This is used by the TIGR clusters [72].

The similarity function is commonly based on a local alignment algorithm, like Smith–Waterman or BLAST [66, 64]. Other possible approaches are using exact substrings [43], the number of matching q–grams, the Euclidean distance between q–gram frequencies (symmetric or asymmetric D^2) [33, 8] or the co–linear sets of exact substrings discussed in Paper II. Zimmermann et al. [88] gives a brief overview of the different methods, and evaluates the resulting clusterings.

The choice of algorithm as well as the threshold affects the correctness of the clustering. Genes from conserved gene families and homologous genes will often have similarities in their sequences. If the similarity function penalizes approximate matches strongly (e.g. only uses exact matches), it may split clusters due to sequencing errors. On the other hand, a similarity function too lenient of approximate matches will tend to merge related genes in a single cluster [42].

The number of sequences produced from a particular region of the genome is called its *coverage*. Low coverage of a gene can leave unsequenced regions in the gene, and the clustering process will be unable to merge clusters representing separate segments of the gene, unless other information (e.g. annotated clone–ID) is used.

Multiple genes can also share part of the genome, typically being situated on opposite strand. *Sense–antisense transcription* often has a regulatory function [76, 85], but makes it very difficult to separate ESTs from the two genes by sequence comparison alone.

1.3.4 Assembly

The final step that we will discuss is sequence assembly. The sequences comprising a cluster are analyzed, and fitted together to reconstruct the originating mRNA sequence as closely as possible. The output sequences are often referred to as *contigs*.

Algorithms for assembly often work by constructing an *overlap graph*, where each sequence is represented as a node and there exists a (directed) edge between nodes if the end of one sequence overlaps (i.e. matches) the beginning of another. An assembly is a Hamiltonian path through the nodes in the graph, traversing each edge at most once (see Section 2.6). The presence of repeats, errors, and sequencing artifacts in the sequences complicate this further.

Several tools for EST sequence assembly exist. `Phrap` [29] is one commonly used tool, it is very fast and uses relatively little memory. `CAP3` [35] generally produces better quality contigs [48] (also corroborated by Paper V), but uses more resources on large clusters. Other options are `miraEST` [11], the `TIGR assembler` [79], and `gap4` [78]. There are also sequence assembly tools that are more specialized for genome assembly [60, 4, 39].

1.3.5 Discussion

In addition to the separate tools mentioned above, there are also comprehensive packages that integrate the necessary tools to perform each step in the analysis pipeline. The most ubiquitous examples are probably the Staden package [78] (which uses `pregap4` for sequence screening and `gap4` for assembly) and `STACKpack` [22] (using `crossmatch/RepeatMasker`, `d2_cluster`, `Phrap`, and the analysis tool `CRAW`).

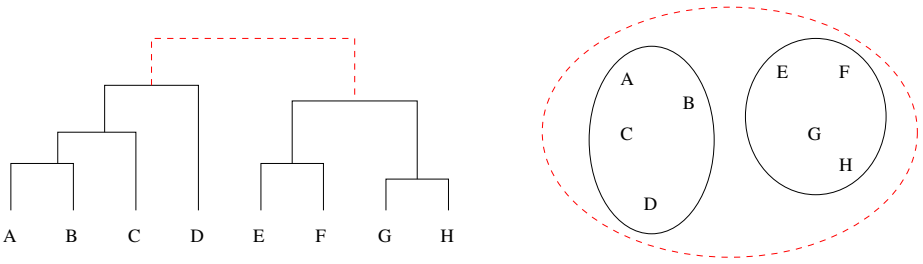

Figure 1.3: A sample dendrogram (left), representing a hierarchical clustering (right; only the top-level clusters are shown). The distance between two clusters can be inferred from the height of the edge joining them.

It is tempting to make the observation that for any type of error or artifact that can arise in the lab, there is a natural sequence variation that is nearly indistinguishable. The sequencing process can stutter, inserting a few extraneous letters in the sequence, but so can an RNA editing event. Single base variation can be natural variation in the gene (*SNP*), RNA editing, or a sequencing error. Chimeric ligation can result in a sequence similar to trans–splicing. And so on.

It is sometimes possible to disambiguate when the natural variations occur often enough that multiple sequences that support it can be found. If this is not the case, one should be very careful in drawing conclusions about gene variation based solely on EST data.

1.4 DATA CLUSTERING

Data clustering is a collection of techniques used in many different scenarios and on many different types of data. In bioinformatics, some examples of the use of data clustering are managing gene expression data, protein family classification, and EST analysis.

1.4.1 Approaches to clustering

An *agglomerative* clustering algorithm builds clusters from the bottom up. Initially, each object is its own cluster. The algorithm finds the two closest clusters, and merges them. This step is repeated, until the desired clustering (measured either by the number of clusters, the distances between them, or some other measure) is achieved.

Conversely, a *divisive* clustering algorithm starts with all data in a single cluster, and proceeds by partitioning them into smaller clusters. As in the agglomerative process, this step is repeated, until a stop criterion is reached.

Both methods give us *hierarchical* clusterings naturally, which can be described by *dendrograms*. An example is shown in Figure 1.3. There also exist non–hierarchical algorithms, see e.g. Jain *et al.* [40] for a review, or Jain and Dubes' book [41] for a more in–depth treatment.

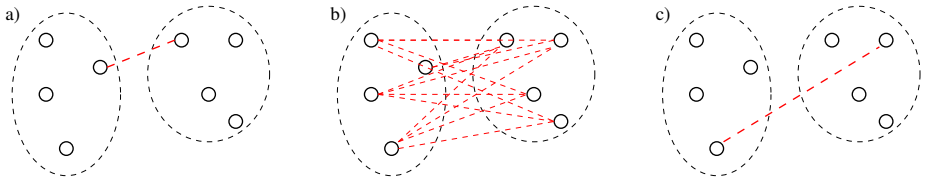

Figure 1.4: Different distance measurements exemplified by a) single, b) average, and c) complete linkage.

1.4.2 Distances

In order to cluster, we need some way of calculating distances between clusters. Often, this distance is derived from the distance between objects. If the objects to be clustered are points in an Euclidean space, Euclidean distance can be used; for sequences, it is common to use a distance based on local or global alignment score.

The simplest distance measures derive the distance between clusters from the set of distances between pairs of objects, so that each pair contains an object from each cluster.

Complete and single linkage define the cluster distance as the maximum and minimum of the set of pair distances, respectively. Slightly more complex is average linkage, using the average of the distances.

Single linkage clustering has the property that it can generate "thin" clusters, by clustering chains of data points (see Figure 1.4). This is often an undesirable trait. On the other hand, complete linkage will tend to produce denser clusters. Average linkage can be seen as a compromise, and it is commonly used in bioinformatics for e.g. clustering gene expression data.

Note that there exist a variety of other distance measures that can be used for clustering data, and there is a large literature on the subject (e.g. [41, 40]).

1.4.3 Sequence clustering

The choice of clustering algorithm is often limited by the data space. In particular for EST clustering, where each sequence can be an arbitrary fragment of the originating sequence, it is desirable to be able to cluster chains of partially overlapping sequences. Thus, the perceived disadvantage of single–linkage clustering is in this case exactly what we need.

1.5 DATA STRUCTURES FOR STRING SEARCHING

Searching for a *pattern* in a data set is one of the most fundamental problems in computer science. Early algorithms for searching used preprocessing of the pattern to speed up the search. The idea behind algorithms like Knuth–Morris–Pratt and Boyer–Moore [30] is to maximize the distance the pattern can be moved when a mismatch is discovered.

It is often the case that a large and relatively static database is repeatedly searched for different short patterns. This limits the benefit of pattern preprocessing, instead it makes sense to preprocess the *data*.

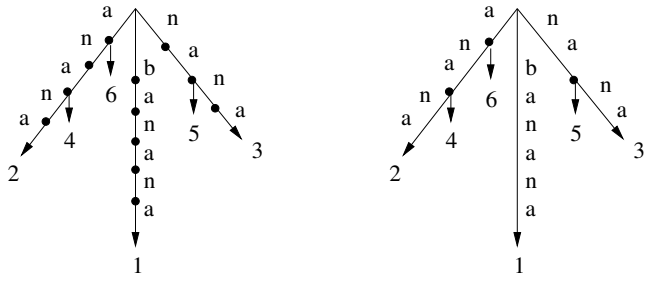

Figure 1.5: The suffix trie (left) and the suffix tree (right) representing the string "banana". The leaves are numbered by the position of the corresponding suffix.

A simple example is the *dictionary*, or *inverted list*. If the data consist of words, we can store each word with its occurrences in the data. Determining the presence of a word can, with a good choice of data structure, be performed in time proportional to the length of the word, regardless of the size of the data set.

For data without defined word boundaries, we can do something similar by constructing a *suffix trie*[4]. Given an alphabet Σ and a string $S \in \Sigma^*$, a suffix trie for S is a tree where each suffix of S is represented by a unique leaf, and each branch from a node is labelled with a character from Σ. The branch labels on a path from the root to a leaf are the characters constituting the suffix represented by that leaf.

Searching for a substring in the data can be performed by traversing the tree from the root and checking for the presence of branches corresponding to the characters of the substring.

The main disadvantage of the suffix trie is that its construction requires $O(n^2)$ time and space.

1.5.1 The suffix tree

Paths in the suffix trie where the nodes have only a single child can be contracted to a single edge, labelled with a string instead of a single character. The resulting data structure is called a *compact suffix trie*, or *suffix tree*.

Surprisingly, the suffix tree can be constructed in $O(n)$ time. Weiner's algorithm [82] was the first to achieve this, but an alternative $O(n)$ algorithm was later published by McCreight [54], with better performance in practice. Almost 20 years later, Ukkonen published another algorithm [80].

The algorithms are described in [30], and [24] gives a detailed comparative analysis.

1.5.2 The suffix array

While suffix trees have important theoretical advantages, the $O(n)$ algorithms suffer from practical disadvantages. Giegerich and Kurtz [23] showed that it

[4]The suffix trie is sometimes called the *atomic suffix tree* [24]

was often faster in practice to perform a simple sorting of the strings making up the suffixes, even if the asymptotic complexity is inferior. They ascribed this to the poor memory locality of the suffix tree construction. Additionally, the construction of suffix trees uses excessive memory, requiring $15n$ bytes or more [45].

In an attempt to address these problems, it was suggested storing the suffixes in sorted order in a linear array [27, 52, 53]. This *suffix array* can be constructed simply by sorting the strings, and it can be searched in $O(\log n)$ time using binary search.

As the suffixes are inserted in the array, the sorting process needs to traverse deeply enough that suffixes are disambiguated. The required depth is called the *longest common prefix*, or *LCP*, between two suffixes. The LCP between pairs of consequtive suffixes is often stored in a separate column in the array.

Several efficient algorithms for suffix array construction have been proposed. Construction can be performed in $8n$ bytes of space and $\Theta(n \log n)$ time [46]. Other algorithms reduce the space requirement to $5n$ bytes (i.e. only storing the input text and the suffix array itself), but may use $O(n^2 \log n)$ time [38, 75]. In practice, these algorithms perform well as long as the LCP is low. Ferragina and Manzini includes a brief review, and also proposes a new algorithm [20].

The suffix array data has been further augmented, leading to the *enhanced suffix array* [1]. This supports all the operations of the suffix tree, but can be stored in $12n$ bytes.

CONTRIBUTIONS

2.1 LIST OF PAPERS AND OVERVIEW

The included papers are:

- Paper I: Korbinian Schneeberger, Ketil Malde, Eivind Coward, and Inge Jonassen. Masking repeats while clustering ESTs. *Submitted for publication.*
- Paper II: Ketil Malde, Eivind Coward, and Inge Jonassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19 no. 10:1221–1226, 2003.
- Paper III: Ketil Malde. Space–efficient incremental single–linkage clustering. *Unpublished*.
- Paper IV: Ketil Malde, Alexander Sczyrba, and Inge Jonassen. A method for constructing EST clustering benchmarks. *Submitted for publication*.
- Paper V: Ketil Malde, Eivind Coward, and Inge Jonassen. A graph based algorithm for generating EST consensus sequences. *Bioinformatcis, in press*.

This chapter gives an overview of the contributions in this thesis, in the order they are involved in the EST analysis pipeline. In addition to short introductions to the papers, I also describe work done after publication, including further developments and enhancements to the published algorithms and tools. I will also discuss related work by others that is not treated in the papers themselves.

2.2 MASKING REPEATS

`RepeatMasker` appears to be the most commonly used tool for masking sequences. As mentioned in Section 1.3.2, it is computationally intensive, mainly due to the optimal alignment algorithm employed by `crossmatch`.

Another potential question is whether the repeat databases are appropriate for the task. There are several potential problems:

i. The repeat database may be incomplete, as not all repeats for an organism are known and classified. If an unknown repeat exists, this can lead to sequences from otherwise unrelated genes being clustered together (Paper IV contains an example of a sequence where an unknown repeat causes clusters to be incorrectly merged.)

ii. EST analysis is particularly useful for new organisms. This often means that no repeat database specific to the organism exists, and repeats are often masked against databases for other species presumed to be closely related.

iii. As mentioned, a sequence is masked by alignment to the known repeats. However, clustering can be performed using any of a variety of distance measures, and e.g. a distance measure using exact matches will be less susceptible to approximate repeats. Masking them can instead cause the clustering to fail to group sequences that originate from the same gene, in particular if coverage is low.

iv. The repeat databases are often constructed with genome assembly as a goal. Thus it contains many repeats found in the intra–genic regions of the genome. Transcribed regions, on the other hand, and in particular the coding sequence of genes, have strict limitations on their content, and many of the repeats that are common in the genome as a whole, do not occur, or occur so rarely as not to make a difference, in EST sequences.

2.2.1 Library–less repeat detection

For an EST containing a subsequence that is present in several genes (i.e. a repeat), the region containing this subsequence will tend to have more matches against other sequences than the rest of the EST. The higher the coverage of the genes involved, the more marked the difference.

The repeat detection algorithm developed in Paper I collects all matches between sequences, and uses this to generate a match count profile for each sequence. Regions of the sequence that are found to match more sequences than the average, are masked as repeats.

Surprisingly, this simple method appears to work very well, and some of the results presented in the paper even indicate that better clustering quality can be achieved with this method than by using traditional comparison to repeat libraries.

The current work is mainly a proof of concept at this stage. Currently, it parses the output from `xsact` (discussed in Section 2.3) to detect matches. For efficiency and ease of use, repeat detection and masking should be integrated in the clustering tool.

As the sizes of clusters increase, so does the probablity that similarity arising by chance (e.g. sequencing errors) will merge the clusters. It is possible that clustering quality could be improved by detecting and masking sequences with low quality, but this possibility has not been explored yet.

2.3 SEQUENCE CLUSTERING USING A SUFFIX ARRAY

As discussed in Section 1.3.3, EST clustering performance is generally performed building a transitive closure of sequences with similarity above a certain threshold.

The transitive closure can be constructed by calculating the distances between all possible pairs of sequences in a *distance matrix* . With n sequences,

this obviously carries an $O(n^2)$ worst case time complexity. It is possible to improve this in an opportunistic fashion by avoiding comparisons between sequences already found to be in the same cluster. The worst case complexity is still quadratic, as each sequence must be compared to all other sequences in the case that no sequences match any other.

2.3.1 The suffix–based algorithm

The suffix array (or suffix tree) makes it possible to identify all shared substrings of a given length in data set in $O(n)$ time. This can be used to accelerate the clustering. If the clustering criterion requires an exact match of a certain length[1], a suffix array can be used to efficiently produce candidate pairs for the clustering. If a pair of sequences do not share a required exact match, they will not be considered.

The algorithm we developed for `xsact` uses a suffix array to identify all shared substrings, and calculates a similarity score for a pair of sequences by summing the lengths of the shared substrings. To be precise, the set of substrings used for calculating the score are non–overlapping and co–linear in the sequences, and the scoring algorithm finds the maximal such set.

The algorithm was tested on a benchmark data set containing 10,000 human ESTs, and was found to produce clusterings comparable to those produced by `d2_cluster` and `BLAST`–based clustering. `xsact` was also faster.

The worst–case behavior is still quadratic. This is easily seen in the case where all sequences share a sufficiently long common substring, as all pairs of sequences will be considered.

2.3.2 Optimizing match detection[2]

Many suffix array construction algorithms do not perform well if the data contain long, repeated subsequences [20]. Unsurprisingly, the relatively simplistic algorithm used in the early versions of `xsact` suffered from this. In addition, since we want to detect matches against the reverse complement of any sequence, suffixes of the reverse complement of every sequence are also added to the array. This leads to superfluous matches, since any matching substring can be found both in the sequences and in their reverse complement, and the redundancy had to be removed at a later stage.

We first make the observation that for an arbitrary word size k, any maximal exactly matching substring of length $\geq k$ in two sequences contains a left–maximal and a right–maximal matching substring of length k. These must be on the same diagonal in an alignment matrix, and there are no other left– or right–maximal matches between them.

This means that instead of using the suffix array to identify the full–length exact matches, it suffices to identify the *endpoints* of the matches, as the full matches can be reconstructed by sorting the endpoints by diagonal, and pairing them up, as illustrated in Figure 2.1. Left–maximal endpoints are found as

[1] Note that this requirement can be implicit; a requirement of e.g. 95% identity in a window of length 100 implies an exact match of length at least 16.

[2] This technique was presented at the BREW'03 workshop, April 2003.

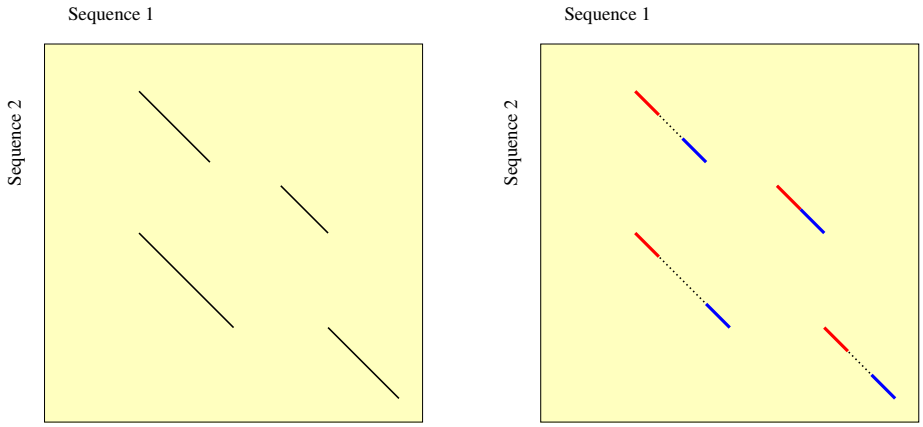

Figure 2.1: Endpoint detection in `xsact`. Instead of finding the maximal match (left), we identify only the left– and right–maximal matches of length k (right). The maximal matches can later be reconstructed from these.

pairs of suffixes from the forward sequences that have different preceding characters, while the right–maximal endpoints are pairs of suffixes of the reverse–complemented sequences. Thus, the suffix array only needs to be sorted to a depth of k.

2.3.3 Subsequent improvements

Sequences frequently contain *simple repeats*, where a short sequence of nucleotides is repeated multiple times (the most common example being poly–A tails). While long repeats slow down suffix array construction, short repeats tend to produce a large number of matches, slowing down the determination of the consistent set of matches. Like other kinds of repeats, simple repeats will also tend to occur in sequences from unrelated genes, and thus merge clusters.

Fortunately, the presence of simple repeats in a sequence can be easily detected by examining how a sequence matches itself, as illustrated in Figure 2.2. When simple repeats have been identified, the offending region is ignored (effectively masked) when matches are generated against other sequences.

There are some facilities in `xsact` for concurrent clustering. The generation of endpoints can be distributed over multiple processes by sequence prefix, so that the first process generates all endpoints starting with `AA..A`, the next `AA..C`, and so on to `TT..T`. Each output file can be individually sorted, and then merged in linear time, constructing the matching scores between sequence pairs.

2.3.4 Related work

The use of a suffix array to filter candidates for alignment was explored in Burkhardt *et al.* [9].

A short while after the publication of this article, Kalyanaraman et al. [43] published a similar algorithm. Their approach traverses a suffix tree in order to determine the longest shared substrings first, and consequently clusters based

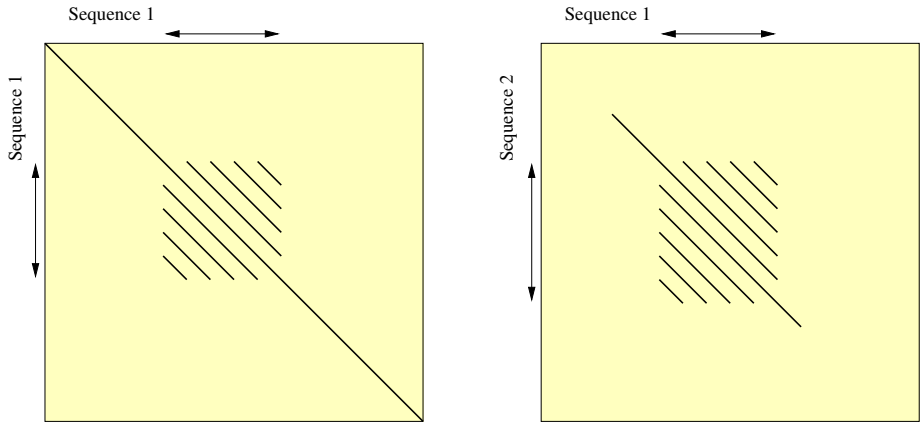

Figure 2.2: Simple repeat detection and elimination in `xsact`. The presence of simple repeats (indicated by an arrow) is detected by comparing the sequence with itself (left), and the region is later masked when comparing to other sequences (right).

Short	**Long option**	**Effect**
`-k` k	`--block-size=`k	matching block size (default is 16)
`-n` n	`--threshold=`n	required clustering threshold
`-p` p	`--prefix-length=`p	prefix length defining suffix blocks
`-x`	`--external-sort`	use external sorting of matches, reducing memory footprint
`-u`	`--input-upper`	accept only upper case characters
`-o` F	`--output=`F	output file name (default is stdout)

Table 2.1: The basic options supported by `xsact`

Short	**Long option**	**Effect**
`-M`	`--output-matches`	output all matches detected
`-N`	`--output-newick`	output a newick-formatted tree
`-U`	`--output-unigene`	output clustering UniGene style
`-L`	`--output-labels`	output clustering as a list of labels
`-P`	`--output-pairs`	output sequence pairs with matches, highlighting the matching blocks in the sequence data

Table 2.2: `xsact` output format options

on a single exact match. Their tool, called `PaCE` is geared towards parallel execution, and can utilize clustered computers.

The `vmatch` suite [2] is a set of tools for constructing and utilizing enhanced suffix arrays [1] for biological sequences. It can also perform sequence clustering, using the suffix array to find seeds, and then extending them to generate local alignments.

It is possible to do sequence clustering in sub–quadratic time by limiting comparisons to a small set of reference sequences. Pedretti [65] describes an algorithm that represents each cluster by a single sequence (normally the longest sequence in the cluster). This is a very fast approach, but the inability to merge clusters means that the resulting quality is not optimal [50].

2.4 INCREMENTAL SINGLE–LINKAGE CLUSTERING

A non–hierarchical single–linkage clustering can be constructed by examining the distances between pairs in arbitrary order, discarding distances that are either redundant (when both objects already are clustered together) or over the clustering threshold. As unused distances are discarded, it is not necessary to store more than one distance for each data point, and this algorithm runs in $O(n)$ space.

A hierarchical clustering, on the other hand, is usually built by scanning the distances in sorted order. This imposes a $O(n^2)$ complexity in space as well as an additional $O(\log n^2)$ factor in time for sorting the data.

Several practical "solutions" to this problem exist, the most commonly employed for sequence clustering is simply to construct non–hierarchical clusterings. Another possibility is to use external storage (i.e. disk).

2.4.1 Incremental clustering

A clustering algorithm is *incremental* if it constructs the clustering for n objects, by first constructing the clustering for $(n-1)$ objects, and then updating this with the nth element.

A single linkage clustering of $(n-1)$ objects contains at most $(n-2)$ distances, while adding the nth object will add another $(n-1)$ distances. An incremental algorithm will at most need to consider $2(n-1)$ distances at a time, and this can potentially reduce space consumption in the hierarchical case to less than twice that for the non–hierarchical case.

In this paper, we explore two methods for constructing single–linkage hierarchical clustering incrementally. Algorithm 1 performs tree rotations (not unlike balanced search trees) when the clustering is updated as a shorter distance between clusters is discovered. Algorithm 2 generates the clustering repeatedly on increasing data sets, but discards from future use the distances not found to participate in a partial clustering.

An unexplored aspect of this work is the possibility to remove any serial bottleneck for parallel clustering implementations. For k processors, k partial clusterings can be generated independently for data sets n/k in size, and the final clustering can be constructed taking only the resulting kn distances into account.

2.4.2 Earlier work

A hierarchical single–linkage clustering is analogous to finding the minimum–cost spanning tree in a graph, and this problem has been studied extensively. While I am not aware of any previous work describing the incremental tree rotations of Algorithm 1, e.g. Eppstein et al. [16] describe an algorithm similar to the transitive closure–based Algorithm 2 presented in this paper.

It is nevertheless worth mentioning these algorithms in this context, since they do not appear to be widely known or used in the bioinformatics community — for instance, `PaCE` [43] parallelizes the discovery of pairs in the clustering, but uses a single node to build the clustering itself.

Partly for reasons of efficiency, sequence clustering uses non–hierarchical algorithms. It is possible, however, that the clustering quality could be improved by increasing the required similarity threshold before merging large clusters.

2.4.3 Implications for average linkage clustering

While hierarchical clustering is used extensively for e.g. gene expression data, it is usually constructed by calculating an explicit distance matrix. Also, average linkage, rather than single linkage is normally used, so at first glance, these algorithms do not seem relevant. However, there exists a relation between single and average linkage clustering.

Theorem: Given a distance measure d, and a threshold t, each cluster in the single linkage clustering with threshold t, is the union of a set of clusters in the average linkage clustering with threshold t.

The proof is by induction. Note that since there is a finite number of objects, there is a finite number of possible distances that can be calculated, and it is possible to order the "interesting" thresholds.

Proof: For $t_0 = 0$, the theorem holds trivially, as each object is in its own cluster in both clusterings. Let t_k be an arbitrary threshold, and assume the theorem holds for all clusterings with lower thresholds.

Let $A = \{a_i | i = 1, 2..n\}$ and $B = \{b_j | j = 1, 2..m\}$ be clusters in the average linkage clustering for thresholds $< t_k$ that are joined at threshold t_k. This implies that $t_k = \frac{1}{nm}\Sigma_{i,j} d(a_i, b_j) \geq \frac{1}{nm}\Sigma_{i,j} \min_{i,j}(d(a_i, b_j)) = \frac{nm}{nm}\min_{i,j}(d(a_i, b_j)) = \min_{i,j}(d(a_i, b_j))$.

Since the final term is the distance between the clusters in single linkage clustering, the clusters will also be merged using single linkage. Thus, the theorem holds for t_k □

Note that for a *weighted* average linkage clustering, the theorem will also hold, on the rather lax condition that the weighted average is larger than the minimum.

It is possible that this result can be used to construct more space–efficient average linkage clusterings, by first calculating the thresholds for the hierarchical single linkage clustering. The resulting clusters at different levels in the hierarchy limit the distances that need to be considered at one time for the average linkage clustering at the same level.

2.5 CONSTRUCTING A REFERENCE CLUSTERING

There is a profound lack of validated reference data sets for EST clustering. Evaluations of new clustering tools or algorithms commonly use UniGene [5, 74] or TIGR Gene Indices [72] as reference, or simply compares clustering output between different tools.[3] There appears to be no available systematic evaluation of the correctness of the UniGene or TGI clusterings.

Synthetic data sets [88] is one alternative. For synthetic data, the correct clustering is known; on the other hand the sequences may lack some of the features, artifacts and errors found in real sequences.

2.5.1 Genome–based clustering

Paper IV describes a method for constructing a reference clustering based on genomic alignment. Genomic mapping of ESTs has been used successfully to investigate gene structure in *Arabidopsis* [87, 31], and alternative splicing in human [19].

As a sample application, we produced a clustering for the ESTs that map to human chromosome 22, and compared it to the clusterings produced by `xsact`, `vmatch`, and `TGICL`. In the process, we experimented with a variety of parameters, and an interesting observation is that approximate matching seems to give *worse* results than matching with more rigid match criteria. It would be premature to draw any conclusions based on a single and limited experiment. One possible explanation, however, could be that exact matching performs better at separating homologous genes (with relatively frequent mutations) while at the same time identifying similarities across small sequence variations.

The paper also discusses the currently used methods for comparing two clusterings, and examines some cases where the indices (Jaccard, Minkowsky, Rand, etc.) may not be measuring an intuitive notion of similarity between clusterings.

While the method described in this paper uses NCBI's alignment of ESTs to the genome, it should be easy to use a different source of alignments. This would give better control over the mapping process.

2.6 GENERATING EST CONSENSUS SEQUENCES

As mentioned in Section 1.3.4, constructing an assembly from sequence overlaps is equivalent to finding Hamiltonian paths in graphs. Unfortunately, this is an NP–complete problem, and thus believed not to have any practical solution. Nevertheless, the heuristics used in contemporary sequence assembly software work well in many cases.

Given the theoretical difficulties with traditional sequence assembly, different approaches to sequence assembly have been explored. One possibility is breaking down sequences into small fragments, and constructing a graph where these fragments represent edges. An assembly can then be constructed by finding an Eulerian path through this graph [32, 67]. Another new development is partial ordered alignment [47, 84].

[3] One exception to this is `PaCE` [43], which evaluates the clustering of *Arabidopsis* against the position of each sequence's alignment to the genome [87].

Short	Long option	Effect
`-k` k	`--word-size=`k	Word size
`-w` w	`--weight-minimum=`w	Ignore edges (words) weighing less than this
`-u`	`--input-upper`	accept only upper case characters
`-o` F	`--output=`F	output file name (default is stdout)
`-G`	`--graph-output`	Output the graph in GraphViz format
`-C`	`--consensus-output`	Output assembled consensus sequences

Table 2.3: Command line options for `xtract`

The tool `xtract` developed in this paper, employs a variation over the Eulerian path approach, but keeps track of the positions each word occurs as well. This extra information is used to guide the greedy algorithm so that it constructs paths that are consistent with actual sequences.

2.6.1 The `xtract` tool

The graph representation of the sequences is stored as a set of k–words, stored with the set of sequence and position pairs. The heaviest word (i.e. the word present in most sequences) is identified, and the consensus sequence is generated by traversing the graph greedily.

While Paper V focuses on the quality of the consensus sequences, the algorithm is also quite fast. The graph building is performed in linear time[4]. Extending the consensus sequence implies looking up the sequences supporting the outgoing edges, and for m sequences and a resulting consensus of length l, the cost is $O(lm)$.

In addition to a set of consensus sequences, `xtract` also has the capability of outputting the cluster as a graph, suitable for rendering with `GraphViz` [15].

As the results show, this is an approach that can perform better than the established competition. The gene index used in Flikka *et al.* [21] was constructed with an early version of `xtract`. The current incarnation of `xtract` does produce an abundance of very similar consensus sequences, and more work is needed in tuning the algorithm to get optimal results (i.e. avoiding the generation of excessive redundancy consensus sequences, while retaining the variety of sequence variations in the data).

[4]Or $O(n \log n)$ if hash table accesses to insert words is a $O(\log n)$ operation.

Future directions

3.1 Improving Sequence Clustering

The worst–case behavior of both the standard transitive closure and of the suffix based clustering algorithms remain quadratic. However, it is an interesting property that while the standard algorithm performs poorly when no sequences match each other, a suffix–based algorithm can run in linear time in that case, as a linear traversal of the suffix array will be sufficient to determine the fact. Conversely, when all sequences match, the standard algorithm runs in linear time, as after matching the first sequence against each of the others, the resulting cluster contains all sequences. It should be possible to develop an algorithm that combines these two properties, and either improves the worst–case complexity, or drastically increases the cases where the algorithm is sub–quadratic.

The automatic detection of repeats shows the potential to become an important technique, even for organisms where the genome is known. The current implementation processes the output of `xsact` to identify the regions to be masked. Masking could be performed more efficiently if the masking process was integrated in the clustering tool.

While Paper I focuses on repeats, it is likely that a similar technique can be used to identify errors like sequencing degradation and chimeric sequences that adversely affect the analysis.

Paper IV indicates that a hierarchical clustering algorithm could be employed to improve clustering quality. Paper III shows how to construct a hierarchical clustering in an efficient manner. Combining these results could produce higher quality clustering without a large increase in computational cost.

3.2 Constructing a Gene Index

One important goal in Bioinformatics is the construction of *gene indices* [42]. There are several gene indices available. RefSeq [70] contains manually verified transcripts from many genes. UniGene [74], the TIGR Gene Indices [72], and STACK [12] are built from EST and cDNA databases using different variations of the pipeline described in Section 1.3. UniGene, TGI, and STACK are compared by Bouck *et al.* [6].

The method described in Paper IV is focused on constructing clusterings, which is a rather "coarse" product. The process could easily be extended to output the positioning of each sequence, giving in effect a reference assembly, as well as a clustering. Such a reference assembly would in its own right be a much–needed supplement for evaluating assembly algorithms.

As part of the process, detailed gene models are constructed, and e.g. exon–intron boundaries, and alternative exons are identified. These gene models provide more information about the gene than single consensus sequences, and it should be relatively straighforward to output and visualize these models.

3.3 IDENTIFYING UNKNOWN GENES

The number of protein coding genes in the human genome is about 20–25,000, according to [37]. The number of EST clusters, on the other hand, range from the current UniGene (build 176), containing approximately 54,000 clusters[1], to the current TIGR Human Gene Index, containing over 200,000 THCs (consensus sequences), and an additional 500,000 singletons. The discrepancy between these numbers is striking, and it is appropriate to ask what the reason for this is.

Clearly, many EST sequences may be the result of genomic or other types of contamination. In the clusters discussed in Paper IV, a large number are singleton clusters. Even if these are disregarded, there remain several times as many clusters as the number of known protein coding genes, and a considerable fraction have a large number of ESTs supporting them, and/or exhibit exon–intron structure.

Genomic tiling is a technique that uses a microarray with probes taken from a chromosome so that as much of the complete sequence is covered as possible. Recent publications using this technique suggests the existence of as much as an order of magnitude more transcriptional units than there are known protein–coding genes [44, 73].

In view of this evidence, it is difficult not to arrive at the conclusion that there still are many unknown genes, and that the EST databases contain many such transcripts. A closer analysis of unidentified clusters could include comparing them to transcribed regions discovered by genomic tiling. In addition, comparative genomics could be used for validation, and the predicted transcripts could be analyzed for classification.

[1] Build 173 contains approximately 107,000 clusters, similar to build 171 used in Paper IV. The reason for this sudden reduction is not known.

BIBLIOGRAPHY

[1] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing Suffix Trees with Enhanced Suffix Arrays. *Journal of Discrete Algorithms*, 2:53–86, 2004.

[2] M.I. Abouelhoda, E. Ohlebusch, and S. Kurtz. Optimal Exact String Matching Based on Suffix Arrays. In *Proceedings of the Ninth International Symposium on String Processing and Information Retrieval*, pages 31–43. Springer-Verlag, Lecture Notes in Computer Science 2476, 2002.

[3] Mark D. Adams, Jenny M. Kelley, Jeannine D. Gocayne, Mark Dubnick, Mihael H. Polymeropoulous, Hong Xiao, Carl R. Merril, Andrew Wu, Bjorn Olde, Ruben F. Moreno, Anthony R. Kerlavage, W. Richard McCombie, and J. Craig Venter. Complementary DNA sequencing: Expressed sequence tags and human genome project. *Science*, 252:1651–1656, 1991.

[4] S. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J. P. Mesirov, and E. S. Lander. ARACHNE: A whole-genome shotgun assembler. *Genome Research*, 12:177–189, 2002.

[5] M.S. Boguski and G.D. Schuler. ESTablishing a human transcript map. *Nature Genetics*, 10:369–371, 1995.

[6] John Bouck, Wei Yu, Richard Gibbs, and Kim Worley. Comparison of gene indexing databases. *Trends in Genetics*, 15:59–62, 1999.

[7] J A Brumbaugh, L R Middendorf, D L Grone, and J L Ruth. Continuous, on-line DNA sequencing using oligodeoxynucleotide primers with multiple fluorophores. *Proceedings of the National Academy of Sciences*, 85(15):5610–5614, Aug 1988.

[8] John Burke, Dan Davidson, and Winston Hide. d2_cluster: A validated method for clustering EST and full-length cDNA sequences. *Genome Research*, 9:1135–1142, 1999.

[9] Stefan Burkhardt, Andreas Crauser, Paolo Ferragina, Hans-Peter Lenhof, Eric Rivals, and Martin Vingron. q-gram based database searching using a suffix array (quasar). In *RECOMB*, pages 77–83, 1999.

[10] C. Caudevilla et al. Natural trans-splicing in carnitine octanoyltranferase pre-mRNA in rat liver. *Proceedings of the National Academy of Sciences*, 95:12185–12190, 1998.

[11] Bastien Chevreux, Thomas Pfisterer, Bernd Drescher, Albert J. Driesel, Werner E.G. Müller, Thomas Wetter, and Sándor Suhai. Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Research*, 14:1147–1159, 2004.

[12] A. Christoffels, A. van Gelder, G. Greyling, R. Miller, T. Hide, and W. Hide. STACKdb: Sequence tag alignment and consensus knowledgebase. *Nucleic Acids Research*, 29(1):234–238, 2001.

[13] Francis H. C. Crick. On protein synthesis. In *Symposia of the Society for Experimental Biology XII: The Biological Replication of Macromolecules*, pages 138–163. Academic Press, 1958.

[14] Francis H. C. Crick. Central dogma of molecular biology. *Nature*, 227:561–563, 1970.

[15] John Ellson, Steven North, et al. The GraphViz web pages. `http://www.graphviz.org/`.

[16] D. Eppstein, Z. Galil, and G. F. Italiano. *Dynamic graph algorithms*. CRC Press, 1997.

[17] Brent Ewing and Phil Green. Base-calling of automated sequencer traces using phred. II Error probabilities. *Genome Research*, 8:185–194, 1998.

[18] Brent Ewing, LaDeana Hillier, Michael C. Wendl, and Phil Green. Base-calling of automated sequencer traces using phred. I Accuracy asessment. *Genome Research*, 8:175–185, 1998.

[19] Eduardo Eyras, Mario Caccamo, Val Curwen, and Michele Clamp. EST-Genes: Alternative splicing from ESTs in Ensembl. *Genome Research*, 14:976–987, 2004.

[20] P. Ferragina and G. Manzini. Engineering a lightweight suffix-array construction algorithm. *Algorithmica*, 40:33–50, 2004.

[21] Kristian Flikka, Fekadu Yadetie, Astrid Laegreid, and Inge Jonassen. XHM: a system for detection of potential cross hybridizations in DNA microarrays. *BMC Bioinformatics*, 5(1):117, Aug 2004.

[22] Electric Genetics. The STACKpack web page, unpb. `http://www.egenetics.com/stackpack.html`.

[23] Robert Giegerich and Stefan Kurtz. A comparison of imperative and purely functional suffix tree constructions. *Science of Computer Programming*, 25(2-3):187–218, 1995.

[24] Robert Giegerich and Stefan Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19:331–353, 1997.

[25] GNU. The General Public License, version 2, 1991. `http://www.gnu.org/licenses/gpl.txt`.

[26] J Gocayne, D A Robinson, M G FitzGerald, F Z Chung, A R Kerlavage, K U Lentes, J Lai, C D Wang, C M Fraser, and J C Venter. Primary structure of rat cardiac beta-adrenergic and muscarinic cholinergic receptors obtained by automated DNA sequence analysis: further evidence for a multigene family. *Proc Natl Acad Sci U S A*, 84(23):8296–8300, Dec 1987.

[27] G. H. Gonnet, R. A. Baeza-Yates, and T. Snider. New indices for text: PAT trees and PAT arrays. In B. Frakes and R. A. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 66–82, Englewood Cliffs, NJ, 1992. Prentice-Hall.

[28] O Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[29] Phil Green. Phrap documentation, 1999. `http://www.phrap.org/phredphrap/phrap.html`.

[30] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge university press, 1997.

[31] Brian J. Haas, Arthur L. Delcher, Stephen M. Mount, Jennifer R. Wortman, Roger K. Smith Jr, et al. Improving the Arabidopsis genome annotation using maximal transcript alignment assemblies. *Nucleic Acids Research*, 31, 2003.

[32] Steffen Heber, Max Alexeyev, Sing-Hoi Sze, Haixu Tang, and Pavel A. Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, pages S181–S188, 2002.

[33] W. Hide, J. Burke, and D. B. Davidson. Biological evaluation of d2, an algorithm for high-performance sequence comparison. *Journal of Computational Biology*, 1(2):199–215, 1994.

[34] X.-Y. Huang and D. Hirsh. RNA trans-splicing. *Genetic Engineering*, 14:211–229, 1992.

[35] Xiaoqiu Huang and Anup Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9:868–877, 1999.

[36] Tadashi Imanishi, Takeshi Itoh, Yutaka Suzuki, et al. Integrative annotation of 21,037 human genes validated by full-length cDNA clones. *PLoS Biology*, pages 0001–0020, 2004.

[37] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–945, 2004.

[38] H. Itoh and H. Tanaka. An efficient method for in memory construction of suffix arrays. In *Proceedings of the Sixth Symposium on String Processing and Information Retrieval*, pages 81–88, Los Alamitos, CA, 1999. IEEE Computer Society Press.

[39] D. B. Jaffe, J. Butler, S. Gnerre, E. Mauceli, K. Lindblad-Toh, J. P. Mesirov, M. C. Zody, and E. S. Lander. Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Research*, 13:91–96, 2003.

[40] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[41] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[42] C. Victor Jongeneel. The need for a human gene index. *Bioinformatics*, 16:1059–1061, 2000.

[43] Anantharaman Kalyanaraman, Srinivas Aluru, Volker Brendel, and Suresh Kothari. Space and time efficient parallel algorithms and software for EST clustering. *IEEE Transactions on Parallel and Distributed Systems*, 14(12):1209–1221, 2003.

[44] Philipp Kapranov, Simon E. Cawley, Jorg Drenkow, Stefan Bekiranov, Robert L. Strausberg, Stephen P. A. Fodor, and Thomas R. Gingeras. Large-scale transciptional activity in chromosomes 21 and 22. *Science*, 296:916–919, 2002.

[45] Stefan Kurtz. Reducing the space requirement of suffix trees. *Software: practice and experience*, 29, 1999.

[46] N. J. Larsson and K. Sadakane. Faster suffix sorting. Technical report, Department of Computer Science, Lund University, Sweden, 1999.

[47] Christoper Lee, Catherine Grasso, and Mark F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18:452–464, 2002.

[48] Feng Liang, Ingeborg Holt, Geo Pertea, Svetlana Karamycheva, Steven L. Salzberg, and John Quackenbush. An optimized protocol for analysis of EST sequences. *Nucleic Acids Research*, 28:3657–3665, 2000.

[49] P. Liang and Pardee A. B. Differential display of eukaryotic messenger RNA by means of the polymerase chain reaction. *Science*, 257:967–971, 1992.

[50] Ketil Malde, Eivind Coward, and Inge Jonassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19 no. 10:1221–1226, 2003.

[51] Ketil Malde, Eivind Coward, and Inge Jonassen. A graph based algorithm for generating EST consensus sequences. *Bioinformatics*, 2004.

[52] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the first annual ACM-SIAM symposium on discrete algorithms*, pages 319–327, 1990.

[53] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22:935–948, 1993.

[54] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of Advanced Computing Machinery (J.ACM)*, 23:262–272, 1976.

[55] Gregor Mendel. Versuche über pflanzen-hybriden. *Verh. Naturforsch. Ver. Brunn*, 4, 1866.

[56] A J Mighell, N R Smith, P A Robinson, and A F Markham. Vertebrate pseudogenes. *FEBS Lett*, 468(2-3):109–114, Feb 2000.

[57] A. Mironov, J. Fickett, and M. Gelfand. Frequent alternative splicing of human genes. *Genome Research*, 9:1288–1293, 1999.

[58] B. Modrek and C. Lee. A genomic view of alternative splicing. *Nature Genetics*, 30:13–19, 2001.

[59] Gordon Moore. Cramming more components onto integrated circuits. *Electronics*, 38, 1965.

[60] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, and K. A. Remington. A whole-genome assembly of drosophila. *Science*, 287:2196–2204, 2000.

[61] ODL. The Open Publication License, version 1.0, 1999. `http://opencontent.org/openpub/`.

[62] R. W. Old and S. B. Primrose. *Principles of Gene Manipulation*. Blackwell Scientific Publications, 5 edition, 1994.

[63] M. Olson, L. Hood, C. Cantor, and D. Botstein. A common language for physical mapping of the human genome. *Science*, 245:1434–1435, 1989.

[64] John Parkinson, David B. Guiliano, and Mark Blaxter. Making sense of ESTs by CLOBBing them. *BMC Bioinformatics*, 3:31, 2002.

[65] Kevin Pedretti. Accurate, parallel clustering of EST (gene) sequences. Master's thesis, University of Iowa, Dept. of Electrical and Computer Engineering, 2001.

[66] 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):651–2, 2003.

[67] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *PNAS*, 98:9748–9753, 2001.

[68] Joan U. Pontius, Lukas Wagner, and Gregory D. Schuler. *NCBI Handbook*, chapter UniGene: A Unified View of the Transcriptome. NCBI, 2003.

[69] J M Prober, G L Trainor, R J Dam, F W Hobbs, C W Robertson, R J Zagursky, A J Cocuzza, M A Jensen, and K Baumeister. A system for rapid DNA sequencing with fluorescent chain-terminating dideoxynucleotides. *Science*, 238(4825):336–341, Oct 1987.

[70] Kim D. Pruitt and Donna R. Maglott. RefSeq and Locuslink: NCBI gene-centered resources. *Nucleic Acids Research*, 29:137–140, 2000.

[71] M. Puttaraju, Sharon F. Jamison, S. Gary Mansfield, Mariano A. Garcia-Blanco, and Lloyd G. Mitchell. Spliceosome-mediated RNA *trans*-splicing as a tool for gene therapy. *Nature Biotechnology*, 17:246–252, 1999.

[72] J. Quackenbush, J. Cho, D. Lee, F. Liang, I. Holt, S. Karamycheva, B. Parvizi, G. Pertea, R. Sultana, and J. White. The TIGR Gene Indices: analysis of gene transcript sequences in highly sampled eukaryotic species. *Nucleic Acids Research*, 29(1):159–64, 2001.

[73] Eric E. Schadt, Stephen W Edwards, et al. A comprehensive transcript index of the human genome generated using microarrays and computational approaches. *Genome Biology*, 5, 2004.

[74] G. D. Schuler, M. S. Boguski, E. A. Stewart, L. D. Stein, G. Gyapay, K. Rice, R. E. White, P. Rodriguez-Tomé, A. Aggarwal, E. Bajorek, et al. A Gene Map of the Human Genome. *Science*, 274(5287):540–546, 1996.

[75] J. Seward. On the performance of BWT sorting algorithms. In *DCC: Data Compression Conference*, pages 173–182, Los Alamitos, CA, 2000. IEEE Computer Society Press.

[76] Jay Shendure and George M. Church. Computational discovery of sense-antisense transcription in the human and mouse genomes. *Genome Biology*, 3, 2002.

[77] A. F. A. Smit and P. Green. The repeatmasker program. `http://ftp.genome.washington.edu/RM/RepeatMasker.html`.

[78] R. Staden. The Staden sequence analysis package. *Molecular Biotechnology*, 5:233, 1996.

[79] G.G. Sutton, O. White, M.D. Adams, and A.R. Kerlavage. TIGR assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19, 1995.

[80] E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14:249–260, 1995.

[81] N. Volfovsky, B. J. Haas, and S. L. Salzberg. Computational discovery of internal micro-exons. *Genome Research*, 13:1216–1221, 2003.

[82] P. Weiner. Linear pattern matching algorithms. In *Proceedings of 14th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–11, 1973.

[83] E. Wingender, X. Chen, E. Fricke, R. Geffers, R. Hehl, I. Liebich, M. Krull, V. Matys, H. Michael, R. Ohnhäuser, M. Prüß, F. Schacherer, S. Thiele, and S. Urbach. The TRANSFAC system on gene expression regulation. *Nucleic Acids Research*, 29:281–283, 2001.

[84] Yi Xing, Alissa Resch, and Christopher Lee. The multiassembly problem: Reconstructing multiple transcript isoforms from EST fragment mixtures. *Genome Research*, 14:426–441, 2004.

[85] Rodrigo Yelin, Dvir Dahary, Rotem Sorek, Erez Y. Levanon, Orly Goldstein, et al. Widespread occurrence of antisense transcription in the human genome. *Nature Biotechnology*, 21:379–385, 2003.

[86] ZhaoLei Zhang and Mark Gerstein. Large-scale analysis of pseudogenes in the human genome. *Current Opinion in Genetics & Development*, 14:328–335, 2004.

[87] Wei Zhu, Shannon D. Schlueter, and Volker Brendel. Refined annotation of the arabidopsis genome by complete expressed sequence tag mapping. *Plant Physiology*, 132:469–484, 2003.

[88] Judith Zimmermann, Zsuzsanna Lipták, and Scott Hazelhurst. A method for evaluating the quality of string dissimilarity measures and clustering algorithms for EST clustering. In *Proceedings of IEEE Fourth Symposium on Bioinformatics and Bioengineering (BIBE)*, 2004.

Papers I–V

MASKING REPEATS WHILE CLUSTERING ESTS

Korbinian Schneeberger[a] Ketil Malde[a]
Eivind Coward[a] and Inge Jonassen[ab]

[a]Department of Informatics, University of Bergen, [b]Computational Biology Unit, Bergen Center for Computational Sciences, University of Bergen

Abstract

A problem in EST clustering is the presence of repeat sequences. To avoid false matches, repeats have to be masked. This can be a time consuming process, and it depends on available repeat libraries. We present a fast and library independent method to eliminate this problem in the process of clustering. We demonstrate that the result is very similar to performing standard repeat masking before clustering.

1 INTRODUCTION

EST sequences are an abundant source of information about gene structure and expression. To organise this information and improve sequence length and quality, EST sequences are often clustered together. Clustering methods differ, but they all rely on grouping overlapping sequences together, based on the idea that overlapping sequences represent the same gene or transcript.

One problem in clustering is the presence of repeats. Repeats are common sequence motifs duplicated hundreds of times in the genome. Since coding sequences do usually not contain repeats, the frequency of repeats is much smaller in ESTs than in genomic sequences. However, repeats can still be present in UTR regions of transcripts, and also in retained intron sequences. Repeat sequences lead to false overlaps of unrelated ESTs, sometimes resulting in oversized clusters.

The usual solution to the problem is to remove repeats by masking. The most commonly used program for masking repeats is RepeatMasker by Smit and Green (http://www.repeatmasker.org). It is based on Smith-Waterman alignment with a library of known repeats, as well as algorithms for detecting low-complexity regions. While this usually works well, there are two problems. One is speed. Repeat masking is sometimes much more time consuming than the clustering itself. Methods for speedup exist [2], but masking of large datasets remains a substantial task.

The second problem is that it can only find known repeats (except for low-complexity regions), which is a major drawback when working with new species. Approaches to repeat identification from genomic sequence alone have been made [3, 4], but this requires a time-consuming all-against-all comparison of the genome to itself.

We present a method to eliminate problematic repeats during the process of clustering. The method is fast and library independent. It is based on the fact that an EST containing a common repeat sequence will generate many more matches in the repeat region than in the rest of the sequence. The repeat can thus be identified and masked. Despite the diversity of many known repeat elements, this seems to work surprisingly well. The method is implemented in a program called RepeatBeater.

We emphasise that this method can not find all repeat sequences like a library search could do, since it will only find common repeats in the actual dataset. However, this is exactly what causes problems in clustering. Our goal is to remove clustering artifacts caused by repeats, and not to identify all repeats. Nevertheless, we demonstrate that the method finds a large proportion of repeats of certain types.

In the following sections we first describe the method and how to tune the parameters, and then we present and discuss a verification on example datasets.

2 DATA SET

Our test data set used for parameter tuning contains 1873 human ESTs, constituting one single cluster. This is the largest cluster obtained when clustering a data set containing over 100,000 human ESTs chosen randomly from the UniGene Human EST collection [7].

After masking for human repeats with RepeatMasker's default options this single cluster containing the 1873 ESTs is separated into 253 clusters plus 186 singletons. We will refer to this clustering as the *reference clustering*.

Additionally, we used the following datasets for verification of the method and parameters: the first 100000 *Arabidopsis thaliana* ESTs from the UniGene build 43 featuring 'cDNA' in the description line, the first 50000 *Oryza sativa* ESTs from UniGene build 51 featuring 'cDNA' in the description line and the first 100000 *C. elegans* ESTs from UniGene build 15.

3 MASKING METHOD

3.1 Terminology and definitions

We define a *matching pair* as two sequences sharing at least one word with the minimum length of 20 nucleotides in a row. A *match* denotes the identical subsequences (longer than 20 nucleotides) a matching pair shares.

Each nucleotide position in a sequence is included in a certain number of matches. We call the distribution of the number of matches over the positions in a sequence the *match distribution*.

The term *repeat* will be used for subsequences which ideally should be masked in the clustering process due to multiple occurrences in different genes.

To calculate all matches in a data set we used the freely available *xsact* tool, which was also used to perform all clusterings [7].

3.2 Outline of the method

The method can be outlined as follows

1. Find all matches and calculate the match distribution for every sequence
2. Identify regions to be considered as repeats
3. Close short gaps between masked regions, and remove short masked regions

The two last steps are detailed below.

3.3 Identification of repeats

Obviously, sequence positions included in repeats are likely to be part of more matches than other positions. Consequently, these regions feature local maxima in the match distribution. However, not every peak in the match distribution describes a repeat. As an additional criterion, we therefore require the number of repeats to exceed the *minimum match threshold* k.

First, the average μ_1 and the standard deviation σ_1 of the match distribution are calculated. Position i is considered as a repeat if

$$m(i) > \max(k, \mu_1 + f\sigma_1) \tag{1}$$

where $m(i)$ describes the number of matches for position i.

In the next section we will discuss the choice of the parameters f (the *multiplier*) and k (the *minimum match number*).

Again the average μ_2 and the standard deviation σ_2 are calculated, but this time disregarding already masked positions. Every remaining position is masked as repeat if it satisfies Eq. 1, with μ_1 and σ_1 replaced by μ_2 and σ_2. We repeat this procedure until no more masking is performed.

3.4 Post-processing of the masking

Because of variability of the repeats and read errors in ESTs, repeat regions will often contain short regions not identified as repeats. Therefore, it is necessary to close gaps between masked regions. This is conceptually an easy task but it is not obvious up to which length a gap shall be closed. This threshold is called the *minimum gap size*. The next section describes the differences in the results gained with different gap sizes.

After connecting neighbouring masked regions there are still very short masked regions left. Though small regions are statistically repeated very often, it does not fit in our definition of a repeat as used in the repeat masking process. We therefore parse the masking again and unmask short masked regions. The next chapter will also discuss different minimum sizes of a masked region, the *minimum length*.

4 TUNING THE PARAMETERS

Our masking method depends on four parameters which have to be adjusted.

4.1 Minimum length of masked region and minimum gap size

The influence of the minimum length threshold on the clusterings is marginal. Figure 1 illustrates the average changes of identical clusters with the reference clustering at different minimum length thresholds. The shown gradient is representative for the whole data set; all 560 test series with the test data set feature a gradient shaped like shown. In fact, the highest number of identical clusters is attained by keeping all the masked regions (that is, setting the minimum length to 0). However, we still recommend to unmask short regions with a threshold of 20 as a useful minimum size of repeats. The impact on the clustering is small, and the masked regions correspond more closely to RepeatMasker's output. RepeatMasker creates 1264 masked regions in the test data set, only 2 of them are shorter than twenty nucleotides.

Figure 2 shows the effect on the number of identical clusters when varying the minimum gap size parameter. RepeatBeater appears very robust against changes of

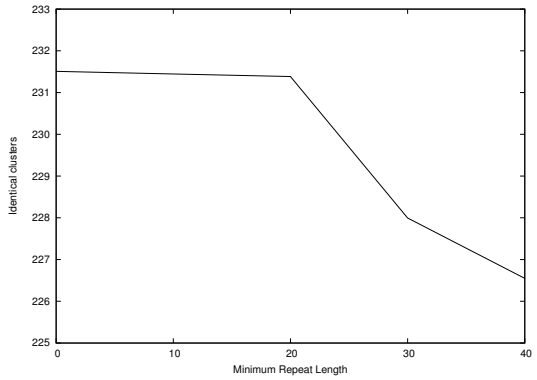

Figure 1: Number of identical clusters as a function of the Minimum Length threshold

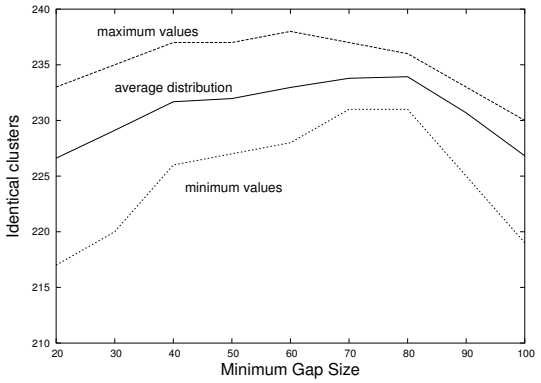

Figure 2: Number of identical clusters as a function of the Minimum Gap Size parameter. Clusters are considered identical if they contain exactly the same set of sequences.

this parameter. In the range from 40 to 80 nucleotides we observe small changes. Based on these curves we choose 70 as the minimum gap size.

4.2 The multiplier f and the minimum match number k

Now that the post-processing of the masking is adjusted, we focus on the parameter involved in the first step of masking: the detection of the masked regions.

Figure 3 visualizes the results of the tests with differing f and k. We set f to 2.15, which is optimal for a large range of values of k. The optimal choice of k depends on the size of the data set. Figure 4 illustrates the influence of k in our original dataset consisting of 105991 ESTs. A raw clustering gives 15938 clusters. After masking with RepeatMasker 16243 clusters and 978 new singletons are formed. 15823 of the clusters are identical. Based on this result we set k to 30.

Table 1 shows the distribution of cluster sizes in the test data in three cases: masking with RepeatBeater, masking with RepeatMasker, and no masking.

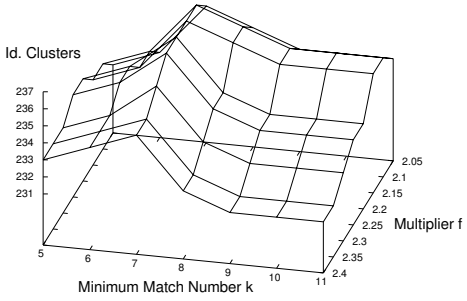

Figure 3: Number of identical clusters as a function of minimum match number and Multipliers

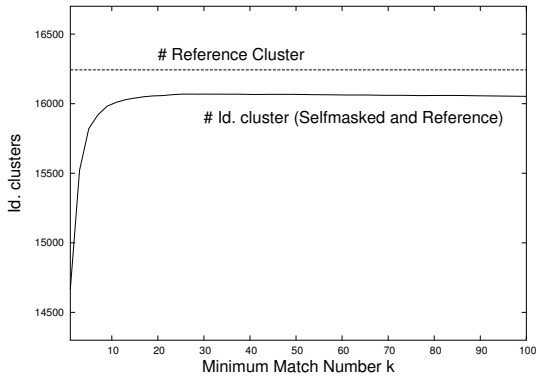

Figure 4: Number of identical clusters as a function of the minimum match number in a large data set

4.3 Verification with different data sets

The method was also tested on other data sets of different sizes, using ESTs of species with existing repeat libraries. We present cluster size distributions of the clusterings after using RepeatMasker, after using RepeatBeater and after clustering the unmasked data set. Additionally, we compared to the corresponding distribution for the UniGene clustering.

We ran RepeatMasker with RepBase update 9.01. For our own masking we used the recommended parameter setting from the previous section.

The datasets used are described in Section 2.

For *A. thaliana* (Table 2), the differences between clusterings appear to be very small concerning the percentage of equal clusters. The effects of repeats in this data set become clear when comparing the largest cluster in each clustering. The clustering produced using RepeatBeater features a much smaller largest cluster than the other clusterings. But compared with the UniGene clustering even our largest cluster is oversized.

For *O. sativa* (Table 3) as well, the clusterings of the rice ESTs feature an oversized cluster compared to the UniGene clustering. Though RepeatBeater does not split this particular cluster, the results are still similar to the clustering after using RepeatMasker.

Cluster size	Number of clusters after masking with		
	RepeatBeater	RepeatMasker	unmasked
1	168	186	
2	65	67	
3 - 4	59	62	
5 - 8	67	64	
9 - 16	40	41	
17 - 32	20	19	
33 - 64			
65 - 128			
129 - 256			
257 - 512			
513 - 1024			
1025 - 2048			1

Table 1: Comparison of different clustering results for our test data set

Cluster size		Number of clusters after masking with		
	UniGene	RepeatBeater	RepeatMasker	*unmasked*
1	204	3017	3010	2988
2	363	1792	1790	1786
3 - 4	918	2243	2240	2232
5 - 8	1798	2161	2152	2147
9 - 16	1905	1433	1408	1403
17 - 32	1012	701	667	654
33 - 64	402	321	298	293
65 - 128	107	110	103	100
129 - 256	39	41	33	27
257 - 512	9	10	9	10
513 - 1024	4	6	5	3
1025 - 2048		1	1	1
2049 - 4096			0	0
4097 - 8192			1	0
8193 - 16384				1
16385 - 32768				

Table 2: Cluster size distribution of clusterings of 100000 *A. thaliana* ESTs

For *C. elegans* (Table 4) we also suspect an insufficient RepeatMasker masking. The differences between the clustering after masking with RepeatMasker and the one resulting out of the unmasked sequences are marginal. Using RepeatBeater reduces again the largest cluster and increases the similarity with the UniGene clustering, but still the differences to the RepeatMasker clustering are small.

4.4 Discussion of the masked regions

While the focus of RepeatBeater is the impact of masking on clustering, we will also take a brief look at the masking itself. In the test data set the maskings of RepeatBeater and of RepeatMasker have 93.5% agreement on the nucleotide level, where 20.9% of the sequence was masked by RepeatMasker.

Further insight is provided when breaking down the masked regions by repeat families, as classified by RepeatMasker (see Table 5). The most common family is SINE/Alu, which is at least partially detected by our tool in 97% of the cases. In 53 cases (5.7%) it is exactly the same region, in many others nearly the same. If we mask only SINE/ALU-repeats with RepeatMasker, we get 96.3% agreement with RepeatBeater on the nucleotide level.

For the other repeat types, RepeatBeater masks them in very few cases (most of these cases being low complexity or simple repeat), and no matches are exact. This is not surprising, since they occur much less frequently than SINE/Alu. Of course, our method can not find repeats which are rare in the actual data set. However, our goal is to remove clustering artifacts caused by repeats, and rare repeats have little impact on clustering.

Cluster size	UniGene	Number of clusters after masking with		
		RepeatBeater	RepeatMasker	*unmasked*
1	505	597	700	576
2	192	229	232	227
3 - 4	445	417	417	412
5 - 8	718	666	678	647
9 - 16	814	719	709	656
17 - 32	502	404	409	353
33 - 64	221	179	176	138
65 - 128	92	76	73	46
129 - 256	18	13	14	7
257 - 512	2	0	0	0
513 - 1024	1	0	1	0
1025 - 2048		0	0	0
2049 - 4096		0	0	0
4097 - 8192		0	0	0
8193 - 16384		1	1	0
16385 - 32768				1

Table 3: Cluster size distribution of clusterings of 50000 *Oryza sativa* ESTs

Cluster size	UniGene	Number of clusters after masking with		
		RepeatBeater	RepeatMasker	*unmasked*
1	785	4688	4666	4642
2	2296	3187	3175	3173
3 - 4	1926	2842	2788	2784
5 - 8	1520	2140	2073	2070
9 - 16	1248	1353	1305	1305
17 - 32	961	643	575	572
33 - 64	397	269	235	233
65 - 128	133	96	74	71
129 - 256	55	52	31	32
257 - 512	9	9	5	5
513 - 1024		2	0	0
1025 - 2048		1	0	0
2049 - 4096			0	0
4097 - 8192			0	0
8193 - 16384			1	1

Table 4: Cluster size distribution of clusterings of 100000 *C. elegans* ESTs

5 CONCLUSION

The presented results show that masking repeats in the clustering process can be done without libraries. The resulting clusterings are very similar to the clustering gained after masking with RepeatMasker, and probably even better when masking with insufficient libraries. In addition the method can be included in the process of clustering itself and avoid the time consuming process of masking.

6 ACKNOWLEDGEMENTS

This work was funded by the Norwegian Research Council through the Salmon Genome Project and the FUGE bioinformatics technology platform.

REFERENCES

[1] Smit,A.F.A., Green,P. RepeatMasker, http://repeatmasker.org

[2] Bedell,J.A., Korf,I., Gish,W. (2000). MaskerAid: a performance enhancement to RepeatMasker. *Bioinformatics*, **16**, 1040–1041.

[3] Kurtz,S., Ohlebusch,E., Schleiermacher,C., Stoye,J., Giegerich,R. (2000). Computation and visualization of degenerate repeats in complete genomes.

Repeat family	total	found
SINE/Alu	919	893
Low complexity	77	17
Simple repeat	59	12
SINE/MIR	53	0
LINE/L2	34	3
LINE/L1	34	1
DNA/MER1 type	23	2
LTR/MaLR	17	1
DNA/MER2 type	14	1
LTR/ERV1	10	0
other	24	0
sum	1264	930

Table 5: Comparison of masked regions. The second column shows how many regions of the given type that were identified by RepeatMasker, and the third column shows how many of these were found by RepeatBeater. A region counts as found if there is at least a partial overlap with a RepeatBeater masked region.

In *Proceedings of the Eight International Conference on Intelligent Systems for Molecular Biology* (eds. R. Altman *et al.*) 228–238. AAAI Press, Menlo Park, CA.

[4] Bao,Z., Eddy,S.R. (2002). Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Res.* **12**, 1269–1276.

[5] Jurka, J. (2000). Repbase Update: a database and an electronic journal of repetitive elements. *Trends Genet.* **9**, 418–420.

[6] Jurka, J. (1998). Repeats in genomic DNA: mining and meaning. *Curr. Opin. Struct. Biol*. **8**, 333–337.

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

BIOINFORMATICS

Vol. 19 no. 10 2003, pages 1221–1226
DOI: 10.1093/bioinformatics/btg138

Fast sequence clustering using a suffix array algorithm

Ketil Malde*, Eivind Coward and Inge Jonassen

Department of Informatics, University of Bergen, HIB, N5020 Norway

Received on September 19, 2002; revised on November 15, 2002 and January 16, 2003; accepted on January 21, 2003

ABSTRACT

Motivation: Efficient clustering is important for handling the large amount of available EST sequences. Most contemporary methods are based on some kind of all-against-all comparison, resulting in a quadratic time complexity. A different approach is needed to keep up with the rapid growth of EST data.

Results: A new, fast EST clustering algorithm is presented. Sub-quadratic time complexity is achieved by using an algorithm based on suffix arrays. A prototype implementation has been developed and run on a benchmark data set. The produced clusterings are validated by comparing them to clusterings produced by other methods, and the results are quite promising.

Availability: The source code for the prototype implementation is available under a GPL license from http://www.ii.uib.no/~ketil/bio/

Contact: ketil@ii.uib.no

INTRODUCTION

Expressed sequence tags (ESTs) constitute a valuable and rapidly growing resource for different kinds of gene analysis; for instance identification of genes, analysis of gene structure (including alternative splicing), and gene expression analysis. For human alone, there are now more than four million EST sequences available.

Because of high redundancy, low quality, and short sequences, clustering of related EST sequences is important in order to extract useful information efficiently. One natural goal for clustering is to group all ESTs originating from the same gene (including possible splice variants). This is the goal of for example, the UniGene clustering (Boguski and Schuler, 1995). Each cluster can then be represented by one or more consensus sequences, assembled from the ESTs (Haas *et al.*, 2000).

Commonly employed methods for sequence clustering (e.g. Holm and Sander, 1998; Burke *et al.*, 1999; Liang *et al.*, 2000) is based on pairwise comparison of sequences, resulting in a similarity score. This score can either be calculated using standard software like BLAST (Altschul *et al.*, 1990) or FASTA (Lipman and Pearson, 1988), or use other more specialized algorithms like d2_cluster's word-based approach.

In general, a transitive closure clustering based on a similarity score can avoid doing an explicit all-against-all comparison, but will in the worst case still need to calculate the similarity between all sequence pairs. Thus, clustering based on pairwise similarity imposes $O(m^2)$ time complexity (where m is the number of sequences to be clustered).

The *UIcluster* program (Pedretti, 2001) uses such a technique, maintaining a global lookup table for known substrings to quickly eliminate many of the comparisons. Additionally it avoids many of the comparisons by comparing sequences only to a representative sequence from each of the existing clusters.

Quadratic scalability can be acceptable for small numbers of sequences, but as databases available contain millions of ESTs, implying on the order of 10^{12} sequence comparisons, faster algorithms are desirable.

The algorithm described in this paper uses a different approach, inspired by suffix trees. Suffix trees, first described by Weiner (1973), allow all occurrences of identical substrings to be identified in $O(n)$ time. This gives us a very quick way to find all *exact* matches between sequences, and we use this as a starting point for EST clustering with sub-quadratic behavior.

TERMINOLOGY

A *suffix array* for a set of strings is a lexicographically ordered array of all suffixes of the strings (Manber and Myers, 1993).

Two sequences have a *matching block* if they have contiguous segments that match exactly. A matching block is *maximal* for a pair of sequences if the sequences differ immediately beyond each end point of the block.

We define a parameter, k, that specifies the length of the shortest matching blocks that the algorithm will detect. A *k-clique* is the set of all sequences that have a specific matching block of length k.

*To whom correspondence should be addressed.

A *score* of a pair of sequences is a numeric measure of their similarity.

Unless otherwise specified, we will in the following define the score of a set of matching blocks to be the sum of the block lengths, and the score of a pair of sequences to be the score of the highest scoring consistent set of matching blocks, where a consistent set is a subset of blocks that are non-overlapping and co-linear (i.e. appearing in the same order) in the sequences.

In order to perform the actual clustering, it is necessary to select the minimal score that will cause two sequences to be clustered together. This value will be referred to as the *clustering threshold*.

We let m denote the number of sequences, and n the total number of nucleotides (i.e. average sequence length is n/m).

ALGORITHM

The whole algorithm is divided into three parts. The first part of the algorithm identifies the pairs that have matching blocks. The second part uses the information generated from this process to calculate a score for pairs of sequences, and finally these scores form the basis for a hierarchical clustering.

(1) Identify all matching blocks of length k:
   (a) construct all suffixes from the data;
   (b) sort the suffixes into a suffix array;
   (c) group the suffixes that share a prefix of length at least k into cliques;
   (d) for each clique, generate the maximal matching blocks between each pair of suffixes in the clique.

(2) Score the resulting sequence pairs:
   (a) for each pair of sequences sharing at least one matching block, collect all matching blocks between the two sequences;
   (b) calculate the largest consistent set of matching blocks, and the corresponding score for each pair.

(3) Generate the clustering:
   (a) starting with the highest scoring sequence pair and working downward, build clusters hierarchically by connecting sequences;
   (b) split the clusters according to the clustering threshold.

Identifying high-scoring pairs

While there exist general data clustering methods that take into account the similarities between multiple objects in order to determine whether to join clusters, single linkage clustering is the most useful approach for clustering of sequence fragments.

The underlying idea for the algorithm is that, while a full similarity matrix contains $O(m^2)$ scores, less than m scores are actually used in single linkage clustering. Thus, one way to achieve significant improvement over algorithms based on a similarity matrix, is to avoid performing the low scoring comparisons.

We observe that since a suffix array is ordered, every clique constitutes a contiguous section of it. Since all sequences that share a matching block of length k are grouped in a k-clique, we can, from each clique in turn, obtain all sequence pairs that share each k-block.

Suffix array generation uses a straightforward, left-wise radix sort, the ith pass sorting the suffixes on the ith nucleotide. While the array could conceivably have been built using a suffix tree algorithm (Weiner, 1973; McCreight, 1976; Ukkonen, 1995), the sorting algorithm is conceptually very simple, and performs well in the expected case. Also, while algorithms for suffix tree construction have good asymptotic behavior, they tend to consume a lot of memory, with poor locality characteristics (Giegerich and Kurtz, 1995).

By first glance, our algorithm hardly does any better, since it produces large intermediate data structures at each level in the recursive sorting algorithm. However, we can get the same result by performing the algorithm on subsets of the data: Selecting a prefix length $l \leq k$, we can for all nucleotide words of length l extract from the data all suffixes that have this word as a prefix. These suffixes can then be sorted into a suffix array, cliques identified, and pairs generated. We retain the pairs, but have no further use for the suffixes in the suffix array, and its space can be reclaimed before generating the suffix array for the next prefix.

While the memory footprint of suffix array generation thus is reduced by a factor of 4^l, up to a maximum of 4^k, the trade-off is increased time to perform the multiple passes over data. The choice for prefix length that is optimal in terms of running time, is thus the smallest value that allows the entire program to execute in available RAM.

The sorting algorithm is also modified to keep track of the match length between subsequent suffixes in the array; this is simply the depth where the sorting terminates (i.e. when there is only a single suffix to sort, or where suffixes have been compared to their full length).

To reduce the number of pairs that are generated, and to simplify book-keeping, we ignore matches that are not maximal by checking that the sequences differ in the preceding nucleotide—if not, the pair is redundant anyway, since the maximal match must be present in a clique elsewhere. Collecting and collapsing multiple matches is achieved by sorting the list of pairs with respect to the ESTs referenced.

Scoring and cluster construction

Sequence scores can be calculated with tools like BLAST, FASTA, or with the d2 algorithm. The present algorithm instead constructs the largest consistent set from the matching blocks already identified, and uses the sum of the block lengths as the score. This calculation is performed efficiently using a dynamic programming algorithm.

Clustering of the scored pairs starts with an empty set of clusters. It takes at each step the highest scoring sequence pair, and checks it against the current clusters to see whether the ESTs are already in the same cluster. If not, we have three possibilities: If the two existing clusters have one each of the sequences, the clusters are merged. If only one of the sequences are clustered, the other sequence is added as a leaf to that cluster. Finally, if none of the sequences are in any clusters, a new cluster is generated with the sequences as leaves. The pair is then removed, and the process is repeated for the next-highest scoring pair, and so on.

The result is a set of clusters, where each cluster is a binary tree, with the sequences as leaves. Each branching node represents a sub-cluster less strongly connected (since it is generated by a lower scoring sequence pair), than sub-clusters below.

ANALYSIS

Generating the suffix array can be implemented in linear time and space using a suffix tree. However, our sorting approach turns out to be no different from constructing a suffix tree using the *lazytree* algorithm by Giegerich and Kurtz (1995) while collapsing the resulting tree. Thus, the efficiency of such a direct sort should be quite good in practice, and run in expected $O(n \log n)$ time, although the worst-case behavior is quadratic.

Generating the suffix array in parts by prefix changes the complexity slightly. With a prefix length of l, the data set is traversed 4^l times, and in each iteration the suffix array for $1/4^l$th of the data set is constructed. However, the depth of the tree does not change, so the resulting time complexity becomes $O(4^l n + n \log n)$, and the space complexity $O(n/4^l)$.

The algorithm for generating pairs from a clique is quadratic in the clique size. This is potentially a costly step, there is a trade off between sensitivity and performance; with k too small, the cliques will grow large and running time will be impacted, with k too large, we may miss some matches, which again may be necessary to produce the correct clustering.

Score calculation will be performed on all generated pairs, and thus for p pairs, the complexity will have a factor p. Different algorithms defining the score metric can have different time complexities; the one in the current implementation uses a dynamic programming algorithm that runs in time linear in the number of blocks as long as the blocks are non-overlapping. In order to handle overlapping blocks, the worst case bounds becomes $O(b^2)$ for b matching blocks, but as the worst case is rare, and the number of blocks matching in two sequences is rarely large for reasonable values of k, scoring is essentially linear in p.

When a pair is to be added to the clustering, the ESTs must be looked up in the existing clusters. Consequently, each cluster stores the ESTs referenced in a searchable structure, which typically has a lookup cost logarithmic in the number of values stored. For p pairs and c clusters, the cost thus becomes $O(pc \log p)$, and, as the clustering stores each sequence exactly once, the space complexity is linear in the number of sequences.

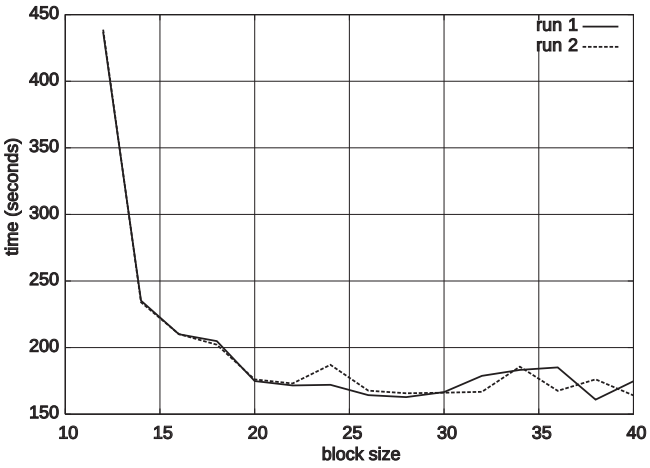

Fig. 1. Times for two complete clustering runs with the block size varying from 12 to 40.

RESULTS

The algorithm was implemented in Haskell (http://www.haskell.org), a high level functional language. The program reads ESTs in FASTA format and produces the clustering as a list of identifiers (usually UniGene accession numbers).

The program was run on a PC equipped with a 1266 MHz Intel Xeon processor and 1 GB RAM, using a benchmark data set available from SANBI (http://www.sanbi.ac.za/benchmarks/). The data set contained 10 000 sequences from human eye tissue, and was masked for repeats and vector sequences using cross_match.

Efficiency

Time consumption is almost solely dependent on the choice of matching block lengths, k. As shown in Figure 1,

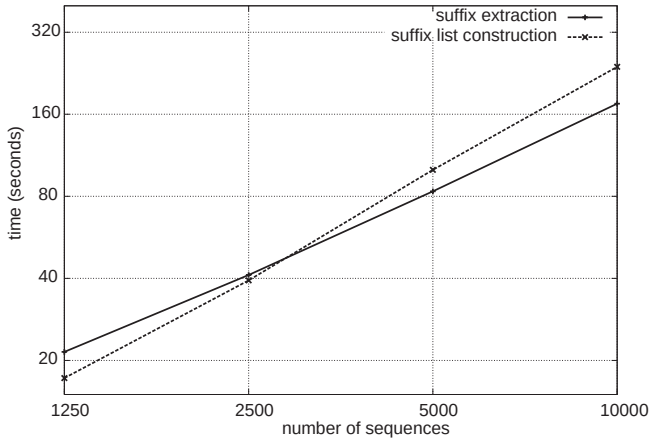

Fig. 2. Time consumed when clustering data sets with different sizes.

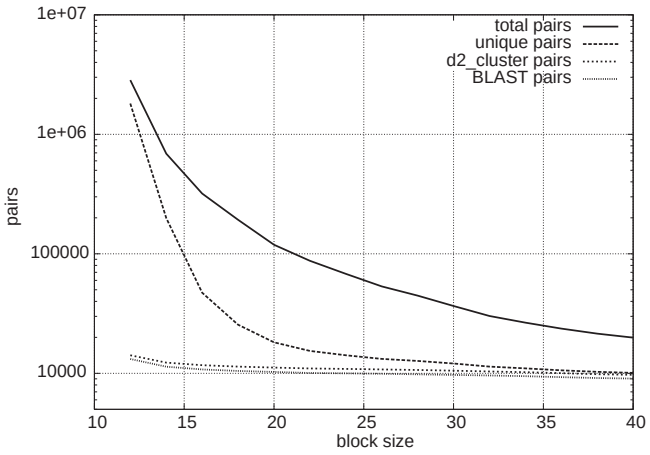

Fig. 3. The total number of matches found, the number of resulting unique sequence pairs, and the number of 'true' pairs produced (i.e. where both sequences are in the same d2 or BLAST cluster).

the speed is approximately constant at around 3 min for block lengths over 20, indicating that pair generation no longer is a significant factor. For block lengths of less than 14, the running time becomes impractical.

For comparison, an all-against-all BLAST search takes 19 min, while d2_cluster uses about 20 min on the same data set on an 866 MHz CPU (Groenewald, personal communication).

In order to measure scalability, the program was run on a data set consisting of 1250, 2500, and 5000 of the sequences, in addition to all 10 000. Timing for the dominating parts, the suffix array construction and suffix extraction (including data reading) is shown in Figure 2. While suffix extraction is almost perfectly linear, suffix array construction is slightly worse than linear.

Running UIcluster on the benchmark set completed in less than 30 s, which is quite impressive. When measuring scalability as described above, it follows a quadratic curve closely; it may scale better for data sets that result in fewer clusters, however.

Sensitivity and specificity in pair detection

We note that for the algorithm to be able to produce a particular (e.g. the 'correct') clustering, we need to generate enough pairs that the correct clusters can be constructed—in other words, there must exist a chain of sequences that match between any pair of sequences within a cluster. More formally stated, for each pair (x, y) of sequences that should be clustered, there must exist a set of pairs (x_i, y_i) for $i = 1, 2..s$, so that $x_1 = x$, $y_s = y$ and $x_{i+1} = y_i$ in the pair generation stage.

Note that this is a necessary, but not sufficient condition; there may be additional pairs generated that span clusters, which again may result in 'over-clustering'.

Similarly, whether the detected pairs are used in the final clustering will depend on their final score. Thus, even if the set of detected pairs is sufficient, the resulting clustering may differ from the correct one.

As a way of measuring sensitivity, we can see whether enough pairs are produced to connect the sequences within each cluster, and as a specificity measure, we can measure the fraction of pairs produced that do not cross cluster boundaries.

In Figure 3, we see that while the number of pairs generated grows faster than exponentially as block size decreases, the number of 'true positives', i.e. pairs that are internal to a BLAST or d2-based cluster, grows much more slowly.

Another interesting property is the relative redundancy of matches, represented by the distance between the two upper graphs. At a block size of 40, we have about twice as many matching blocks as sequence pairs (i.e. each sequence pair has an average of two distinct blocks that match). As the block size decrease, the average number of matching blocks per sequence pair increases as well, until the number of distinct sequence pairs start to rise sharply (at a block length of about 16). This could be interpreted as a measure of quality; a high number of blocks per sequence pair means that the pairs represent real similarities, rather than being due to random matches.

This view is reinforced when taking into account the fraction of detected sequence pairs being inside reference clusters. The 'true' pairs make up a large portion of the detected matches until block sizes decrease below 20 or so, at which point the fraction quickly diminishes.

While we have a measure of 'true positives' detected by our algorithm, a corresponding measure of 'false negatives', i.e. pairs that are similar but fail to be detected,

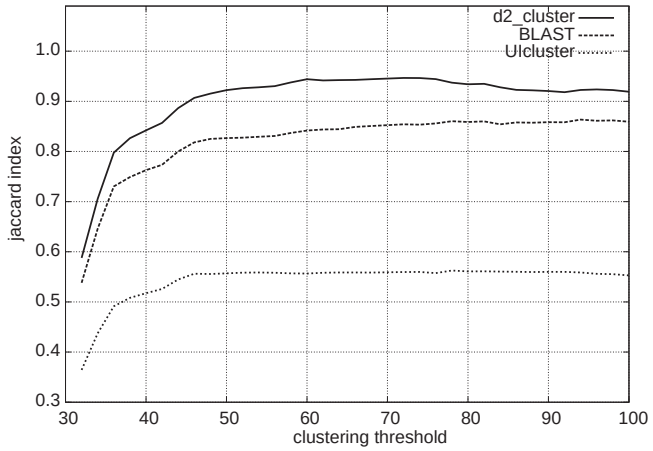

Fig. 4. The Jaccard index with block size 20.

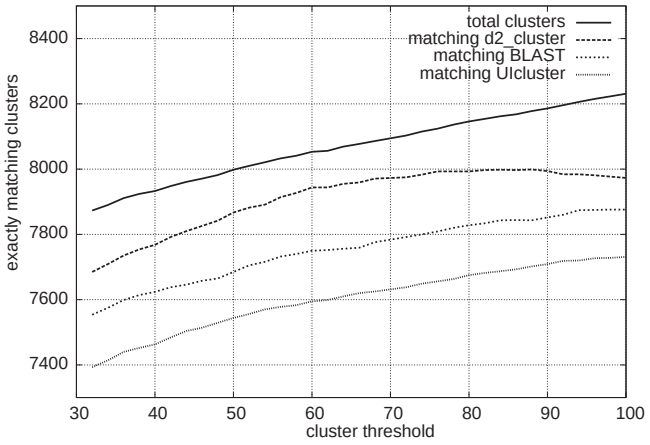

Fig. 5. Counting the number of exactly matching clusters with block size 20.

is harder to calculate. The clusterings contain information about which sequences are clustered together, but not information about which pairs of sequences that match. Thus, we cannot compare the detected pairs directly, and comparing the resulting clusters (see the next section) is at any rate a more relevant measure for determining clustering quality.

We can, however, get an indication on the sensitivity of the pair detection by examining at which block lengths we no longer have enough matches to connect all reference clusters. Experiments show that for the most aggressive choice of parameters tested (block size 12, cluster threshold 32), there are no clusters from neither the d2 nor the BLAST-based clusterings that are split up. Indeed, this holds true for choices of block size up to 16.

Clustering quality

The Jaccard index (Jain and Dubes, 1988) is a measure of cluster similarity. It assigns to a the number of sequence pairs where the sequences are in the same cluster in both clusterings, and to b the number of pairs where sequences are in the same cluster in the first but not the second clustering, and to c the number of pairs where sequences are in the same cluster in the second, but not the first. The Jaccard index is then defined as the number $\frac{a}{a+b+c}$.

In Figure 4, we plot the Jaccard indices for various clustering thresholds with a block size of 20, measured against the output from d2_cluster, BLAST clustering, and UIcluster.

We see that all clusterings reach a plateau for clustering thresholds of 45–50 and up. The closest clustering is d2_cluster, reaching a peak of 0.947 for a threshold of 72. The BLAST based clustering is also close with a maximum of 0.864 for a threshold of 94, with UIcluster less similar, with it's best value, 0.564, at 78.

Table 1. The similarity between clusterings, with the number of clusters on the diagonal, the Jaccard index in the upper half, and the number of exactly matching clusters in the lower half

	20/72	D2	BLAST	UI	UG
20/72	**8103**	*0.947*	*0.854*	*0.560*	*0.503*
D2	7975	**8118**	*0.865*	*0.559*	*0.494*
BLAST	7792	7812	**8298**	*0.537*	*0.473*
UI	7638	7631	7672	**8419**	*0.307*
UG	1629	1619	1578	1458	**2910**

Another measure, used by Burke *et al.* (1999), is the number of exactly matching clusters, as well as clusters in one clustering consisting entirely of a set of clusters in the other. The number of exact clusters compared to the other clusterings is shown in Figure 5. Again, we see that we are closer to d2_cluster than to the BLAST based clustering. The number of exact clusters grows steadily, peaking at 7999 exactly matching clusters for a threshold of 88 for d2, and 7876 clusters for BLAST at 98 and 100. UIcluster's closest value is 7731 for a threshold of 100.

Table 1 shows a comparison between the different clusterings using a block size of 20 and a clustering threshold of 72. UniGene is also included, but as it does not contain all of the benchmark sequences, and as it contains additional sequences that can impact clustering, these values are, at best, only indicative. However, the values give an indication of the level of agreement between the different clusterings.

Again, the proximity of d2 and the present algorithm is underscored, while d2 is marginally closer to BLAST-based clustering.

DISCUSSION AND CONCLUSION

The current prototype implementation runs with performance comparable to or better than d2_cluster and BLAST-based clustering, as long as block sizes are larger than 12.

Short block sizes produce a large number of false positives, and the clustering quality as well as performance appears to be much better for larger block sizes.

While UIcluster is remarkably fast, the comparison to the more well-evaluated methods indicates that the quality is inferior. Pedretti points out some weaknesses of the implementation in his honors project (http://genome.uiowa.edu/pubsoft/clustering/uicluster.doc) that support this observation.

Scalability for our algorithm appears to be close to linear in practice, and implementing the performance-critical parts of the algorithm in a lower level language with more focus on optimization should result in a noticeable constant factor speedup. The critical parts of the algorithm also lend themselves to parallel implementation. The combination of these properties suggests that a tool capable of clustering of the complete set of human ESTs available from public databases using reasonable computing resources may be within reach.

When comparing with the established algorithms, the choice for block size appears to be optimal in terms of clustering quality between 18 and 30. This is probably best interpreted as a consequence of real similarities producing longish matching regions containing matches separated by short gaps, while in some cases, the algorithm identifies short matching blocks separated by large gaps.

The current algorithm does not use gap penalties or other refinements, these options should be explored in the future in order to provide better specificity while maintaining high sensitivity.

ACKNOWLEDGEMENTS

We wish to thank the people at SANBI and Electric Genetics who have been extremely helpful. In particular Liza Groenewald has helped to clarify the details concerning d2_cluster, and provided a version of the benchmark data masked for repeats and vector sequences, as well as the complete clustering output.

Also, we thank the anonymous referees, whose constructive criticism has improved the article

Finally, we are grateful to Stephan Haas and Department Vingron at the Max Planck Institute for Molecular Genetics for their hospitality, as well as for advice, suggestions, and for providing the BLAST-based clustering results.

This work was funded by The Salmon Genome Project, a Norwegian Research Council program.

REFERENCES

Altschul,S., Gish,W., Miller,W., Myers,E. and Lipman,D. (1990) A basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Boguski,M. and Schuler,G. (1995) ESTablishing a human transcript map. *Nat. Genet.*, **10**, 369–371.

Burke,J., Davidson,D. and Hide,W. (1999) d2_cluster: A validated method for clustering EST and full-length cDNA sequences. *Genome Res.*, **9**, 1135–1142.

Giegerich,R. and Kurtz,S. (1995) A comparison of imperative and purely functional suffix tree constructions. *Science of Computer Programming*, **25**, 187–218.

Haas,S.A., Beissbarth,T., Ribals,E., Krause,A. and Vingron,M. (2000) GeneNest: automated generation and visualization of geneindices. *Trends Genet.*, **16**, 521–523.

Holm,L. and Sander,C. (1998) Removing near-neighbour redundancy from large protein sequence collections. *Bioinformatics*, **14**, 423–429.

Jain,A.K. and Dubes,R.C. (1988) *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ.

Liang,F., Holt,I., Pertea,G., Karamycheva,S., Salzberg,S.L. and Quackenbush,J. (2000) An optimized protocol for analysis of EST sequences. *Nucleic Acids Res.*, **28**, 3657–3665.

Lipman,D.J. and Pearson,W.R. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.

Manber,U. and Myers,G. (1993) Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, **22**, 935–948.

McCreight,E.M. (1976) A space-economical suffix tree construction algorithm. *J. Adv. Comput. Machinery*, **23**, 262–272.

Pedretti,K. (2001) Accurate, parallel clustering of EST (gene) sequences, Master's Thesis, University of Iowa, Department of Electrical and Computer Engineering.

Ukkonen,E. (1995) On-line construction of suffix-trees. *Algorithmica*, **14**, 249–260.

Weiner,P. (1973) Linear pattern matching algorithms. *Proceedings of 14th IEEE Symposium on Foundations of Computer Science (FOCS)*. pp. 1–11.

SPACE-EFFICIENT INCREMENTAL SINGLE-LINKAGE CLUSTERING

Ketil Malde

Department of Informatics, University of Bergen

Abstract

Single linkage clustering is an important technique in many fields of bioinformatics. One important application is EST clustering, where the data sets can contain millions of sequences. The standard algorithm suffers from large memory requirements, making it impractical in this case. This paper describes a methodology for constructing single linkage hierarchical clustering requiring only linear space. The algorithms are also incremental, so that new information can be added at a relatively small marginal cost.

1 INTRODUCTION

Data clustering is an important and commonly applied technique in Bioinformatics as well as in many other fields. Grouping objects according to similarity is an intuitive way to deal with large data sets, and grouping the groups leads to hierarchical clustering.

While there are many different approaches to clustering, one of most straightforward is *single linkage agglomerative hierarchical clustering* [3], which only requires that we can measure the distance between any two objects in the data set, with the requirement that for any two objects a and b, $dist(a,a) = 0, dist(a,b) = dist(b,a)$ and $dist(a,b) \geq 0$ holds[1].

The standard algorithm starts with each object in a separate cluster, finds the shortest distance between two objects, and merges their clusters. The next shortest distance between two objects (in separate clusters) is determined, and those clusters are again merged, and so on, until either all objects reside in a single cluster, or some distance threshold is reached. The result is a (usually binary) tree with each leaf representing an object, and each internal node representing a subcluster, labeled with the shortest distance between objects in the subclusters.

This method is commonly applied to a variety of data in bioinformatics, from protein sequences and ESTs to and gene expression measurements. While it is simple, it does have a few drawbacks. The set of distances (often stored in a *distance matrix*) needs to be calculated and sorted, this requires $\Theta(n^2)$ space, and normally $\Theta(n^2 \log n)$ time [5], where the $\log n$ factor arises from the sorting process. In the general case a lower time bound on a single linkage clustering algorithm is $\Omega(n^2)$, since we must examine the distance for every pair of objects before we can eliminate it from the clustering. It is, however, possible to perform this clustering faster if we can make specific assumptions about the data space. E.g. [2] shows how to cluster two dimensional data in $O(n \log^2 n)$, but in particular for sequence data, such assumptions cannot be made.

I propose instead to construct a single linkage clustering in an incremental manner. This obviates the need to retain the discarded values from the distance matrix,

[1] The distance is often the Euclidean distance between points in a vector space, but this is not the case for clustering based on e.g. sequence similarity

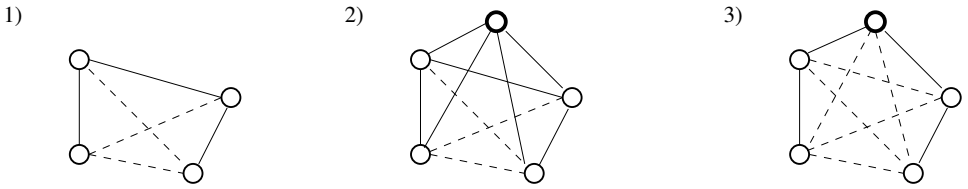

Figure 1: Incremental algorithm for finding the minimum cost spanning tree in a graph. 1) Given a graph G with the MCST, we 2) add a node with all the edges between the new node and the nodes in G. We can then 3) construct the MCST for the new graph by examining only the newly added edges and the edges in the previous MCST.

and we can thus construct the clustering in a space efficient manner. I also show that single linkage hierarchical clustering can be implemented in $\Theta(n)$ space.

The algorithm will be described in two variants; the first is a variation that uses the standard agglomerative algorithm outlined above on increasing subsets of objects, the second incrementally modifies the clustering hierarchy as new distances are calculated in arbitrary order. An implementation of the second variant is provided as part of the EST clustering tool, *xsact* [6].[2]

2 AN INCREMENTAL CLUSTERING ALGORITHM

While agglomerative hierarchical clustering is not incremental, it is possible to do incremental clustering by modifying the algorithm somewhat. We observe that to link n objects, the single linkage hierarchical clustering only needs $(n-1)$ distances, making up the internal nodes in the tree.

Given the correct clustering C for n objects, we can add an object by first calculating its distances to objects in C. We then extract the $(n-1)$ distances used in the construction of C, and sort them together with the the n distances to the new object. We can now rebuild the clustering for the $(n+1)$ objects from the $(2n-1)$ distances, discarding $(n-1)$ of them. Similarly, new objects can be added, giving us an incremental algorithm with time and space complexity $\Theta(n^2 \log n)$ and $\Theta(n)$, respectively.

A single-linkage hierarchical clustering is similar to finding the minimum cost spanning tree, or MCST, in the complete graph consisting of the objects as vertices, with the distances as weights on the edges [1], and the standard algorithm, described in the introduction is quite analogous to Kruskal's algorithm for the MCST problem [9]. Solving the MCST problem for a graph with vertices V and edges E is usually considered to require $O(|V| + |E|)$ space and $O(|E| \log |V|)$ time, although recent publications improve the time bound somewhat [8]. From a graph-theoretic view, the space consumption is optimal, since we need to store the graph itself.

However, the present algorithm trivially leads to an $O(|V|^2 \log |V|)$ time and $O(|V|)$ space solution to the MCST problem. Assume we know the MCST M for a complete subgraph G with n nodes. We then construct a new graph G' by adding a new node v and n new edges to G, making G' complete also. The MCST of G' consists of n edges, taken from M and the n new edges.

This is easily seen by observing that the MCST can be constructed by removing the longest edge from every cycle in the graph. Since the construction of G' only adds to G, any cycle in G will still be present in G', and the same edge can be removed when the MCST for G' is constructed.

[2] I plan to have implemented both algorithms for the final version of the article.

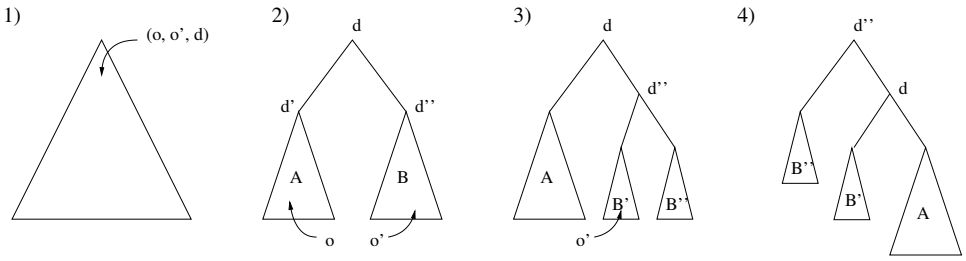

Figure 2: Rotating the hierarchical clustering after insertion of a new distance. 1) a new pair of objects o and o' are inserted with distance d. 2) the smallest existing subcluster containing both o and o' is found, and d replaces the previous distance at its root (assuming d is larger than the previous value). 3) the child B with the smallest distance is identified, and the grandchild B'' not containing any of the objects is also identified. 4) A new parent is created, containing B'' and the new child containing A and B' (and thus o and o' as branches can be rotated similarly, until d is larger than the distances in the children.

3 INCREMENTAL TREE ROTATION

The previous section describes how we can build a hierarchical clustering incrementally, adding one node at a time to the clustering. It is also possible to increment by inserting a single distance at a time.

We define the distance of an internal node of the clustering tree as the shortest distance between a pair of objects, so that one of the objects is in the left branch and the other is in the right branch. We then have for any hierarchical clustering that the distance at each internal node is greater than or equal to the distances in its children (with leaves having a distance of zero by definition). I will refer to this fact as the *hierarchy invariant*.

In addition, we make the rather trivial observation that for the single linkage clustering, for all pairs of objects, the subclusters containing them are connected by a distance smaller or equal to the distance between the objects.

Together, these two observations constitute necessary and sufficient conditions for a correct single linkage hierarchical clustering.

The algorithm is defined inductively: Assume we have a correct hierarchical clustering for part of the data. (The usual starting configuration with each object in its own cluster satisfies this condition). When inserting a new distance between a pair of objects in the clustering, we have one of two possibilities: Either the objects are in different clusters in which case the clusters are *merged* and the new distance inserted at the root. Alternatively, they are in the same cluster. In the latter case, the cluster is traversed until the smallest subcluster containing both objects is found. The new distance is compared to the distance in the existing root of this subcluster, and the smaller distance is retained.

In either case, if the new distance is inserted into the clustering, the hierarchy invariant of the resulting cluster may be violated, and we need to reorganize the hierarchy to maintain the invariant. This is achieved similarly to traditional algorithms for rebalancing trees (see e.g. [9, 7]).

We first observe that the hierarchy above the insertion point is consistent, since we have reduced the distance at the insertion point, it is still smaller than its parent. Thus, we only need to concern ourselves with the subcluster represented by the insertion point node. I will refer to this subcluster as the *parent* and the subclusters below it as its *children*.

We examine the two children A and B, and find the child (B, say) with the

larger distance. If this distance is shorter than the inserted distance d (currently residing in the parent), the tree is correct, and we are done. If the distance in the child is greater, this node belongs higher up in the tree, and it will be rotated up to become the new parent.

Since each of A and B contains exactly one of the objects between which the new distance was measured, B will have two children B' and B'', one of which will not contain either of the objects. The new parent will retain this grandchild (in this case B''), in addition to a new node consisting of the old parent's distance d, and having as children the grandchild B' (that contains one of the objects), and the child A (containing the other object).

Now we have successfully rotated the new distance one step down, the hierarchy invariant is maintained down to the new node, and we can apply the procedure again on this node, until the new distance is at the correct place in the hierarchy.

All pairs are eventually processed, and whenever we encounter a pair that connects already connected subclusters, we discard the pair with the longest distance. Thus, each subcluster is connected by the smallest distance possible.

Let us again examine the situation when we discover a pair of objects that already are in the same cluster. This is analogous to finding a cycle in the graph in the MCST problem, and we resolve it by removing the longest edge in the cycle, which is stored at the top of the hierarchical clustering.

In other words, this algorithm again directly gives us a method for constructing a MCST, but this time by discovering all cycles in the graph, and eliminating the longest edge from each as they are discovered.

A straightforward implementation of this variant stores the objects as leaves and distances in each internal node. While this is clearly $\Theta(n)$ space, the time complexity will depend on how fast one can search for an object in a subcluster. It seems likely that a good caching scheme will be required to get a time-efficient implementation.

4 DISCUSSION

Both variations obviate the need to retain the elements of the distance matrix. The distances still need to be sorted, but this is done implicitly while the clustering is being built.

The clustering is done incrementally, so if a new object is to be added to an existing clustering, it is straightforward to calculate its distances to the previous objects, and insert the resulting pairs in the existing clustering hierarchy. While the first variation is incremental in the objects, the second variation is also incremental in the distances, so the elements of the distance matrix can be calculated.

Both variations will probably entail moderate constant factors, since intermediate clusterings are constructed from pairs that are later discarded as the clustering is updated. While the second variation only requires the distances to be retained, the first algorithm also needs to retain the actual pairs of objects.

The second variation was implemented as part of *xsact*, and run on a variety of data. As expected, it performs somewhat slower than the standard implementation, varying from hardly noticeable on data with many small ("shallow") clusters, to about twice the time on data with a single "deep" cluster. For simplicity, the current implementation is performed in a naïve way, and could certainly be improved.

Of the family of agglomerative hierarchical clustering methods, only single linkage clustering has been examined. While it is perhaps the most commonly used approach for sequences, [4] asserts that complete linkage, using the maximum rather than the minimum distance between objects in the subclusters, is more useful in general. Another popular method is average linkage clustering. It remains an open

question whether similar approaches can be adopted to include other clustering algorithms in the same family.

5 ACKNOWLEDGMENTS

I would like to thank friends and colleagues who supported me with help and encouragement, and in particular Eivind Coward, Inge Jonassen, Pinar Heggernes, Trond Hellem Bø and Nils Olav Handegard.

This work was funded by the Norwegian Salmon Genome Project, a Norwegian Research Council project.

REFERENCES

[1] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 252–257. ACM Press, 1988.

[2] Sabyasachy Choudhury and M. Narasimha Murty. A divisive scheme for constructing minimal spanning trees in coordinate space. *Pattern Recogn. Lett.*, 11(6):385–389, 1990.

[3] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[4] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[5] Takio Kurita. An efficient agglomerative clustering algorithm using a heap. *Pattern Recogn.*, 24(3):205–209, 1991.

[6] Ketil Malde, Eivind Coward, and Inge Jonassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19 no. 10:1221–1226, 2003.

[7] Udi Manber. *Introduction to Algorithms*. Addison-Wesley, 1989.

[8] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

[9] Mark Allen Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings Publishing Company, 1995.

A METHOD FOR CONSTRUCTING EST CLUSTERING BENCHMARKS

Ketil Malde[a], Alexander Sczyrba[b], and Inge Jonassen[ac]

[a]Department of Informatics, University of Bergen, [b]Praktische Informatik, University of Bielefeld, [c]Computational Biology Unit, Bergen Centre for Computational Sciences, University of Bergen

Abstract

While there exist many algorithms for EST clustering, there are no standard data sets for validation of the algorithms. We present a method that derives gene models from the genomic alignment of the ESTs, and constructs a reference clustering by grouping the ESTs that match each gene model. In addition, the clustering is validated by comparing the gene models to the NCBI gene annotations. The resulting reference is exemplarily compared to UniGene, and a subset is used to compare some available clustering tools. The implementation and sample data sets are available at `http://bibiserv.techfak.uni-bielefeld.de/estref/`.

1 INTRODUCTION

Although the number of sequenced genomes is steadily increasing, there remains a number of organisms for which no genome sequence will become available in the near future. Due to the low costs of single read sequencing of the 5' and 3' ends of mRNAs [2], for these organisms ESTs will continue to constitute the most important source of genetic knowledge. But even for organisms where the genome is well known, ESTs can give valuable information about the transcriptome e.g. by identifying or confirming genes, alternative splice forms, and polymorphisms.

As ESTs are only fragments of the complete mRNA sequence, an important step in the analysis pipeline is the clustering of the ESTs into gene-oriented clusters. If the sequences are incorrectly clustered, the subsequent assembly stage is likely to produce incorrect results.

ESTs are strictly speaking the result of single-read sequencing of cDNA clones. However, EST data bases and clustering processes usually include full-length cDNA sequences as well. In the following, we will use the term EST to refer to both types of sequence.

1.1 Clustering methods

Aligning an EST to the genome will often reveal the location of the originating gene. If such genomic alignments for the ESTs are available, a clustering can be constructed by grouping ESTs that are aligned to overlapping or proximate positions in the genome.

Unfortunately, the genome is available only for relatively few organisms. Another obstacle is that the available tools for mapping sequences to the genome, e.g. SIM4 [7] and GeneSeqer [24] are generally too slow to map large data sets, such as all human ESTs against the genome [6].

ESTs are commonly clustered based on pairwise sequence similarities, usually by selecting a similarity threshold and constructing a transitive closure. In general, this process is easy to perform, but it does not scale well to very large data sets. While likely to be less reliable than genomic alignment, it is used either because it is faster (when the data sets are small), because the genome is unavailable, or simply because it is more convenient.

1.2 Data sets for clustering validation

To explore the accuracy of clustering algorithms, a large number of accurate EST clusters is required which can be used as reference sets. New clustering algorithms, such as d2_cluster [5], CLOBB [15], or xsact [14], are frequently evaluated on a small number of test cases, often comparing the results to UniGene [3, 21] or TIGR Gene Indices [13, 18, 16][1]. UniGene is often used as reference data set because of its wide acceptance. While Bouck *et al.* [4] compare UniGene, STACK and HGI, no recent comprehensive evaluation is available.

Both the TIGR Gene Indices and UniGene are, at least partly, constructed using sequence similarity. It is therefore not unlikely that artifacts introduced by a clustering tool will affect the reference as well, giving an artificially good score.

Another possibility is to use a synthetic data set [28], but unless the mechanisms involved – both *in vivo* and *in vitro* – are understood to a level where the different errors that arise in practice can be correctly modeled, the synthesized sequences may not be representative of real data.

So while a synthetic data set can give us the correct clustering with certainty, the data may not be representative for what is encountered in reality, using real data sets implies uncertainty about the correct clustering. There is, however, a possibility of a middle ground, using a subset of a real data set, where the clustering is validated against other resources.

1.3 A reference data set

In the following, we describe a method for constructing reference data sets for EST clustering, which allows evaluation and optimization of clustering tools. The implementation and sample data sets are available at `http://bibiserv.techfak.uni-bielefeld.de/estref/`.

For human, the sequence of the genome is well established, and NCBI provides as part of their Mapview[2] the spliced alignment of most available ESTs against the genome. We will therefore use this genomic alignment of ESTs as the basis for a reference clustering, called B3C, and compare this to UniGene and the output from automatic clustering tools.

The use of reference data sets is a common approach to evaluate new algorithms, and has been used successfully in many related fields: *BAliBASE* [23] is a database of manually-refined multiple sequence alignments specifically designed for the evaluation and comparison of multiple sequence alignment programs, *CASP* (Critical Assessment of Techniques for Protein Structure Prediction, [1]) aims at establishing the current state of the art in protein structure prediction, *GASP* (Genome Annotation Assessment Project, [19]) tried to obtain an in-depth and objective assessment of the current state of the art in gene and functional site predictions in genomic DNA of *D. melanogaster*, and Gardner *et al.* [8] published a comprehensive comparison of comparative RNA structure prediction approaches.

[1] One exception is the PaCE [10, 11] clustering algorithm, which was evaluated by a comparison of mapped *A. thaliana* ESTs [27].

[2] `http://www.ncbi.nlm.nih.gov/mapview/`, `http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=handbook.chapter.1562`

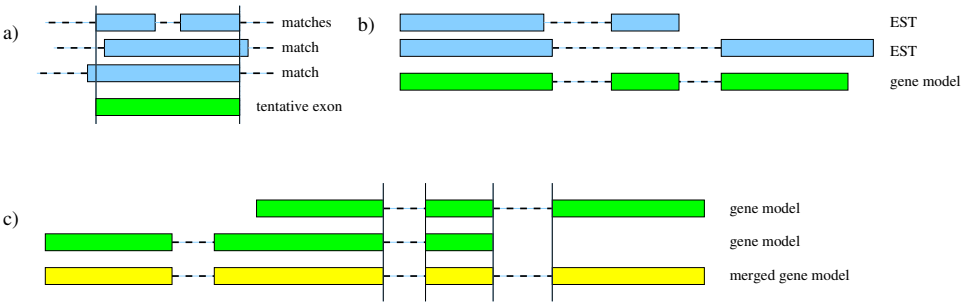

Figure 1: The construction of gene models from matches. The process starts by determining the boundaries of the tentative exons, based on coverage and conserved end points (a). Then, the tentative exons constructed from matches in the same ESTs are joined (b). Finally, compatible gene models are merged (c).

2 METHODS

2.1 Genome-mapped clustering

The NCBI Mapview files (build 34.3) were downloaded from the NCBI FTP site[3]. We used the file `hs_esttrn.md`, where each line contains one contiguous match, described by EST accession, genomic contig, strand, and start and end positions in the contig, and start and end positions in the EST.

Mapview contains 3,855,378 ESTs, mapped to 11,553,008 matches in the whole genome. There is some redundancy in the genomic contigs. The *DR51 haplotype* is a variant of chromosome 6, the X and Y chromosome share many, but not all genes, and TCAG provides an alternative assembly of chromosome 7 [20]. This resulted in 189,141 sequences being mapped to more than one contig. This was resolved by removing the contigs from DR51, chromosome Y and the TCAG assembly. This left 451 contigs containing 11,035,360 matches to 3,851,143 ESTs, where each EST mapped to a unique contig.

The matches were first grouped by transitive closure of overlaps, that is, a group consisted of all matches in a contiguous region of the genomic contig, so that each position in the region is covered by at least one EST, and no match crosses the endpoints of the region.

Within each region, tentative exon boundaries were determined based on coverage and conserved starting and ending points in the genomic contig. Each matching fragment was assigned to a unique tentative exon.

The tentative exons containing matches from the same EST were then grouped, resulting in sets of linked exons, which we will refer to as *tentative gene models*.

Due to sequence truncation, masking, or incorrect matching, ESTs sometimes contain only part of an exon. Since a very accurate match is required when the tentative exons are constructed, this can lead to a gene being represented as several distinct models at this point.

The models were therefore compared, and similar models were merged based on strand, the number of overlapping exons, and how closely each exon in one model matches an exon in the other. However, gene models were allowed to be positioned in the introns of other models, have interleaved exons, and overlap to a small degree without being merged. The ESTs supporting each of the resulting merged models were then output as clusters.

[3] `ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/maps/mapview/BUILD.34.3/`

RefSeq type	genes	transcripts
NM	16,443	20,330
XM	6,333	6,334
NR	128	192
XR	19	26
pseudogene	1,317	1,317
none	2,843	2,843

Table 1: The distribution of annotated genes by RefSeq transcript type.

2.2 The NCBI gene annotations

The NCBI provides annotated assemblies of the human genome. Data of various types and numerous sources is assimilated and made available for download or through the Mapviewer. For a detailed description of the annotation process see [12]. Here we give a short summary.

RefSeq RNA sequences, mRNA sequences from Genbank and ESTs from dbEST are aligned to the genomic sequence using MegaBLAST. These alignments are processed to produce candidate gene models as follows: (1) putative exons are identified by taking splice sites near the ends of the alignments into account, (2) a compatible set of exons for the model is selected, and (3) BLAST may be used to identify additional exons that were missed in the initial transcript alignments.

All gene models based on a particular RefSeq RNA are compared and the best one is selected. Overlapping gene models that are compatible with each other are combined into an extended model. Additionally, the *ab initio* gene prediction program GenomeScan [25] is used to build gene models from protein homologies. The models are consolidated, attributed to known genes whenever the correspondence is clear and predicted genes named. Multiple models based on alternative transcripts for some genes may be produced and are annotated only if the models are based on RefSeq mRNAs representing alternative transcripts from the same gene.

Each RNA sequence contained in the RefSeq database [17] represents a distinct transcript produced from a particular gene. The accessions distinguish between manually curated sequences (with accession prefix NM for mRNA and NR indicating other RNA) and computed sequences (XM and XR, respectively).

For the human genome build 34.3 the NCBI annotations contain 26,669 different genes. The distribution of transcripts is shown in Table 1. Genes without a RefSeq transcript appear to have each exon annotated individually, so the gene count for this category is probably inflated. ENSEMBL provides an independent gene annotation, containing 24,261 annotated genes [6].

2.3 Comparing B3C to the NCBI annotations

We define the overlap of a match between a tentative exon and an annotated exon to be the length of the overlapping region divided by the total length of the union of both exons. More precisely, if a tentative exon x is described by genomic positions P_x, and an annotated exon y is described by genomic positions P_y, we define overlap$(x, y) = |P_x \cap P_y|/|P_x \cup P_y|$ Thus, the overlap will reach its maximal value of 1.0 if and only if the tentative exon and the annotated exon have the exact same sets of positions.

In the following, we will consider a gene model to match an exon annotation when at least one exon in the gene model overlaps an exon in the annotation by 0.9 or more. Using this definition, 179,049 tentative exons are annotated, and 18,869

clusters/gene	genes	genes/cluster	clusters
1	19,054	1	17,699
2	353	2	1,038
3	17	3	101
4	6	4	15
		5–8	11
		9+	5

Table 2: The number of annotated genes that are split over multiple clusters, and the number of clusters containing multiple gene annotations.

clusters have one or more annotated exons. (Clusters that match annotations without explicit exon structure are not taken into account.)

Some of the clusters have multiple annotations, while some of the annotations match exons in multiple clusters. Table 2 summarizes this. We see that very few ($<$2%) of the annotations are split over multiple clusters. Six of the annotated genes are spread over four clusters, these are all long genes with many exons, and with poor coverage in the EST data. A larger percentage (6%) of the annotated clusters contain multiple gene annotations.

There are several known cases where genes on opposite strands of the genome overlap, causing antisense transcription [22, 26]. Combined with the unreliability of strand annotation, this causes difficulty in separating overlapping genes, and there are regions where it is difficult, even with manual inspection, to determine the endpoints, and even the number, of transcribed regions involved.

In some cases, incorrect clusters seem to be caused by incorrectly mapped sequences. One of the clusters matches three gene annotations, and overlaps several others. Examination of one of the sequences, AF067420, reveals that it is mapped as nine matches spanning most of the cluster. Although RepeatMasker fails to find any known repeats, closer inspection reveals that BLAST aligns this sequence to 77 different locations in the genome. It is therefore likely that at least parts of the sequence are incorrectly mapped to a conserved region of a nearby, related gene.

2.4 Extracting a reference subset of the clusters

While the exon matching described above ensures a relatively strong correspondence between the annotated and the tentative exons, it does not take into account the structure of the annotated gene vs. the model representing the cluster. While sensitivity is high, specificity is low.

For a reference clustering, we wish to only have clusters that have good matches between the cluster's gene model, and the corresponding gene annotation. We often see that while exon-intron boundaries are well conserved, the starting position of the gene tends to be poorly conserved in the EST data. Consequently, gene models not matching the annotation in the first exon is a relatively common case. In addition, there seems to be a large number of cases of unannotated splice variants.

We therefore extract clusters matching gene annotations using the following criteria:

- the cluster matches one, and only one, gene annotation (as defined above)
- the gene annotation matches no other clusters
- there exist a maximum of two exons at either end not matching annotated exons

- no more than two of the gene's annotated exons are unmatched to exons in the model

Note that we do not compare the individual sequence against the annotation, but the model built from a consensus of sequences. This lets us retain sequences that only partly fit the model, and thus includes a more realistic amount of noise.

Using these criteria, we extracted 13,257 clusters, containing 1,778,791 sequences.

2.5 The comparison of clusterings

It is common to provide histograms or tables displaying the distribution of cluster sizes. While this is sometimes useful to compare the stringency of the clustering, one obviously cannot in general assume a similar clustering from similarity in the cluster size distribution.

The most commonly used methods for comparing clusterings are based on counting the status for all pairs of objects. Let $A(x)$ be the cluster containing x in clustering A. Then, given two clusterings A and B, we calculate:

$$\begin{aligned} a &= |\{(x,y)|A(x) = A(y), B(x) = B(y)\}| \\ b &= |\{(x,y)|A(x) = A(y), B(x) \neq B(y)\}| \\ c &= |\{(x,y)|A(x) \neq A(y), B(x) = B(y)\}| \\ d &= |\{(x,y)|A(x) \neq A(y), B(x) \neq B(y)\}| \end{aligned}$$

Several measures of clustering similarity are based on these pair counts. The most common ones are perhaps the Jaccard index ($a/(a+b+c)$), the Rand index ($(a+d)/(a+b+c+d)$), and the Minkowski index ($\sqrt{(b+c)/(a+c)}$) [9].

Sometimes, in particular when we are comparing against a presumably correct reference clustering, a is referred to as *true positives*, d are *true negatives*, and b and c are *false positives* and *false negatives*, respectively. This leads to the *specificity* ($a/(a+b)$), *sensitivity* ($a/(a+c)$), and *correlation coefficient* as correctness measures [10].

It is also common to comparing clusterings A and B by counting the number (or percentage) of exactly matching clusters, the number of clusters in A that are subsets or supersets of clusters in B, and the number of clusters that are in complex relationships (e.g. [5, 14]).

In the following, we will provide the Jaccard index, together with cluster size histograms and relationship counts for comparison purposes.

3 RESULTS

3.1 UniGene

The set of sequences in UniGene (build 171) is not the same as in the NCBI Mapview. Of the 4,579,560 sequences contained in UniGene, 3,563,016 are present in the Mapview data when redundant contigs are removed. Thus, 1,016,544 sequences are unique to UniGene, while 292,362 are unique to the set of mapped sequences. In order to make a valid comparison, we computed an *intersection* between the clusterings. Sequences not present in both clusterings were removed from either clustering, and any resulting empty clusters were deleted. Any sequence not present in both clusterings is thus ignored for the purpose of this comparison. The histogram for cluster sizes before and after the intersection, is shown in Table 3.

cluster size	UniGene	B3C	intersected UniGene	intersected B3C
1	8126	109158	21258	24944
2	38207	26283	21241	21657
3-4	23295	16375	13049	13564
5-8	11978	8715	6854	7422
9-16	5588	4475	3791	3979
17-32	3733	3333	3228	3289
33-64	3292	3254	3218	3234
65-128	3822	3756	4023	3773
129-256	4365	3938	4089	3785
257-512	3003	2312	2246	2148
513-1024	1043	834	738	775
1025-2048	332	218	203	201
2049-4096	106	74	70	75
4097-8192	33	36	26	30
8193-16384	12	12	9	12
16385-32768	2	1	1	1
total clusters	106937	182774	84044	88889

Table 3: The cluster sizes in UniGene and in the clustering arising from genomic positions, and for the clusterings intersected to their common subset.

	B3C		UniGene
exact matches	54,390	$=$	54,390
overclustering	2,291	$<$	5,066
underclustering	11,122	$>$	4,798
complex	21,086	$\times$	19,790

Table 4: Comparing B3C to the UniGene clustering. “Overclustering” indicates that one B3C cluster is a union of multiple UniGene clusters, “underclustering” is the converse, and “complex” denotes the cases where the union of sets of multiple clusters from each clustering contain the same sequences. The Jaccard index is 0.69. The exactly matching clusters include 16,170 singletons.

size	reference	xsact	vmatch	TGICL
1	27	84	106	304
2	8	15	21	35
3–4	16	12	17	35
5–8	17	20	21	29
9–16	18	18	18	19
17–32	12	12	12	12
33–64	14	13	13	14
65–128	32	31	31	29
129–256	21	20	20	20
257–512	13	14	14	14
513–1024	5	5	5	5
total	183	244	278	516

Table 5: The distribution of cluster sizes for the automatic clustering tools. The xsact clustering was performed using a word size of 25 and a clustering threshold of 45, while the parameters for vmatch was clustering sequences with regions of 90% or better similarity, a length of at least 60, and a gap penalty of 2. The distributions are very similar, in particular for the large cluster sizes.

UniGene augments the clustering by merging clusters joined by annotated clone ID. This technique will be able to merge both ends of a gene, even when no transcript bridges the 3' and the 5' ends. Thus we expect to see cases where one UniGene cluster is a superset of B3C clusters mapped to genomic positions relatively close. Indeed this seems to be the case for a majority of the "underclustering" cases; of the 4,798 superclusters, only 184 contains sequences that map to different parts of the genome.

However, for the 19,790 clusters in complex relationships, 5,366 are split into clusters that are separated by a considerable distance in the genome. E.g. Hs.1857 (208 sequences) is mapped to nine different locations, spread over seven chromosomes. Hs.1933 (89 sequences) maps to four distinct locations.

3.2 Evaluation of automatic clustering tools

The full data set contains close to two million sequences, and is too large for clustering based on pairwise similarity to be practical with reasonable computational resources. In order to get a smaller benchmark data set, we extracted from the full set the clusters that are mapped to chromosome 21. The data set was further reduced in size by retaining only clusters that map to regions of the chromosome corresponding to a single NCBI gene annotation. (This is also likely to remove many "difficult" clusters). The resulting data set consisted of 16,027 sequences from 183 clusters.

We then masked the sequences for repeats with RepeatMasker and vector sequence using vmatch, and generated new clusterings with xsact, vmatch and TGICL, using a variety of parameters. The results for vmatch are shown in Figure 2.

In Table 6, the different clusterings are compared using the Jaccard index, and Table 5 shows the cluster distributions with the highest Jaccard score for each tool.

While xsact calculates similarity as the sum length of exactly matching blocks, vmatch and TGICL start with an exact block (seed), and try to extend it, tolerating small errors. The different parameters explored for vmatch are displayed in Figure 2. It is interesting to note that the best results are achieved with relatively short

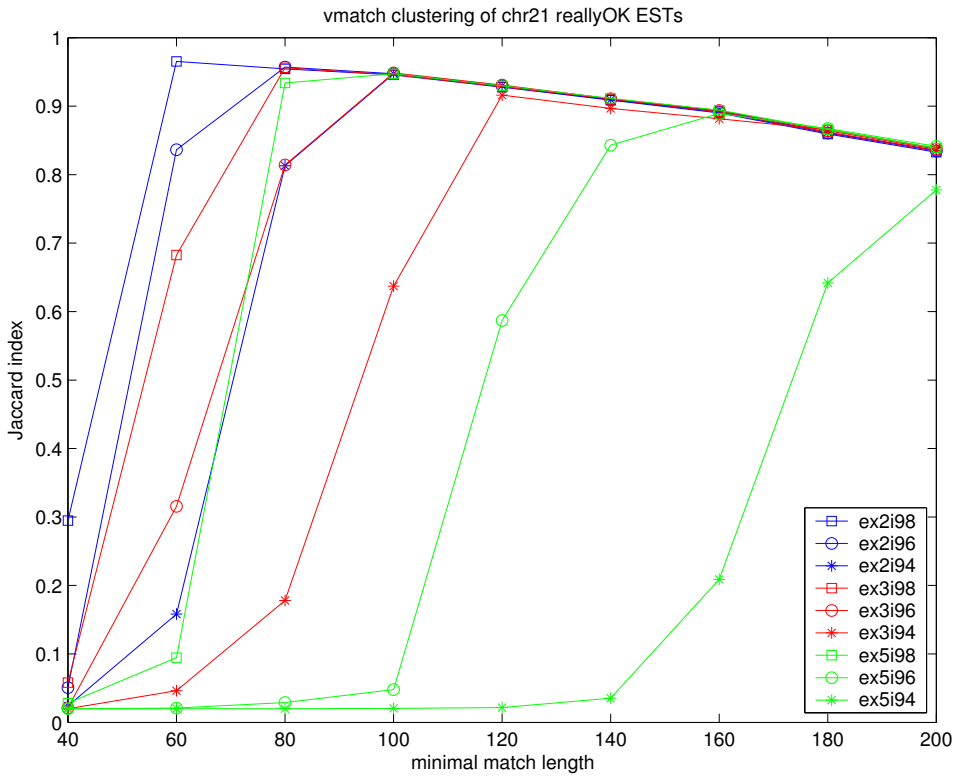

Figure 2: The Jaccard index for clusters produced by vmatch with a variety of parameters. The plot shows the Jaccard index (ordinate) for the vmatch clusters compared to the reference set. On the abscissa the minimal match length necessary to cluster two sequences into the same cluster is shown. Different curves correspond to different matching parameters (gap penalty 2,3 and 5, percent identity 94, 96 and 98). With longer match lengths, the other parameters become less relevant, but the overall score decreases slowly. The best results are achieved using shorter, but more accurate matches. A similar behavior (not shown) is exhibited by xsact.

	xsact	vmatch	TGICL
reference	0.968	0.965	0.957
xsact		0.970	0.950
vmatch			0.973

Table 6: The Jaccard scores between the clusterings from Table 5. The clusterings are quite similar, but while the vmatch clustering is closer than TGICL to the reference, they are even more similar to each other.

matches and low error tolerance.

4 DISCUSSION AND CONCLUSIONS

We have developed a method for constructing reference clusterings based on genomic alignment, and optional refinement using annotated genes. This method is a good compromise between wholly synthetic data sets, which may fail to take into account the variety of error conditions in real data, and references constructed from sequence similarity, which may fail because of similarities between genes, or short overlaps between transcripts.

According to the NCBI documentation, UniGene is constructed primarily based on genomic alignment, and secondarily using sequence similarity and annotation to augment the clustering. It is therefore surprising that we find several cases where UniGene clusters contain sequences from different chromosomes. Without intimate knowledge of the UniGene construction process, it is difficult to identify the cause of this; it is something that should be investigated more closely, however.

It is interesting to contrast the two comparisons of the genome-based clustering to UniGene and the NCBI gene annotations. The NCBI annotations indicate that the genome-based clusters are too large, and each cluster contains multiple genes more often than a gene is split over multiple clusters. On the other hand, the UniGene comparison seems to show that clusters are too small, with a relatively higher number of UniGene clusters being split up in the genome-based clustering.

There can be several reasons explaining this; one is the relative scarcity of annotated genes compared to EST clusters. This affects the average distance between clusters in the genome (if there are fewer clusters, they are less likely to be close enough to interfere with each other). Another possible reason is that many of the annotated genes are highly expressed, and thus less likely to be split up due to partial coverage.

There is an important discrepancy between the Jaccard index and an accurate number of clusters. All of the automatic clustering tools fail to connect many of the smaller clusters, producing a relatively large number of singletons, and to a lesser degree, smaller clusters. This could reflect that larger clusters increase the probability of random matches, and suggests that an improved clustering could be produced by using dynamic clustering threshold (i.e. requiring a stricter threshold for merging larger clusters).

5 ACKNOWLEDGMENTS

We would like to thank Professor Robert Giegerich and the Praktische Informatik group at the University of Bielefeld for their hospitality and support.

This work was funded in part by the Salmon Genome Project, which is a project under the Norwegian Research Council.

REFERENCES

[1] *CASP5. Proceedings of the 5th Meeting on the Critical Assessment of Techniques for Protein Structure Prediction*, volume 53. Proteins, 2003.

[2] M.D. Adams, J.M. Kelley, J.D. Gocayne, M. Dubnick, M.H. Polymeropoulos, H. Xiao, C.R. Merril, A. Wu, B. Olde, R.F. Moreno, A.R. Kerlavage, W.R. McCombie, and J.C. Venter. Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–6, 1991.

[3] M.S. Boguski and G.D. Schuler. ESTablishing a human transcript map. *Nature Genetics*, 10:369–371, 1995.

[4] John Bouck, Wei Yu, Richard Gibbs, and Kim Worley. Comparison of gene indexing databases. *Trends in Genetics*, 15:59–62, 1999.

[5] John Burke, Dan Davidson, and Winston Hide. d2_cluster: A validated method for clustering EST and full-length cDNA sequences. *Genome Research*, 9:1135–1142, 1999.

[6] Val Curwen, Eduardo Eyras, T. Daniel Andrews, Laura Clarke, Emmanuel Mongin, Steven M. J. Searle, and Michele Clamp. The Ensembl automatic gene annotation system. *Genome Research*, 14:942–950, 2004.

[7] Liliana Florea, George Hartzell, Zheng Zhang, Gerald M. Rubin, and Webb Miller. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Research*, 8:967–974, 1998.

[8] P. Gardner and R. Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5(1):140, 2004.

[9] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[10] Anantharaman Kalyanaraman, Srinivas Aluru, Volker Brendel, and Suresh Kothari. Space and time efficient parallel algorithms and software for EST clustering. *IEEE Transactions on Parallel and Distributed Systems*, 14(12):1209–1221, 2003.

[11] Anantharaman Kalyanaraman, Srinivas Aluru, Suresh Kothari, and Volker Brendel. Efficient clustering of large EST data sets on parallel computers. *Nucl. Acids. Res.*, 31(11):2963–2974, 2003.

[12] P. Kitts. *NCBI Handbook*, chapter Genome Assembly and Annotation Process. NCBI, 2003.

[13] Feng Liang, Ingeborg Holt, Geo Pertea, Svetlana Karamycheva, Steven L. Salzberg, and John Quackenbush. An optimized protocol for analysis of EST sequences. *Nucleic Acids Research*, 28 No 18:3657–3665, 2000.

[14] Ketil Malde, Eivind Coward, and Inge Jonassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19 no. 10:1221–1226, 2003.

[15] John Parkinson, David B. Guiliano, and Mark Blaxter. Making sense of ESTs by CLOBBing them. *BMC Bioinformatics*, 3:31, 2002.

[16] 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):651–2, 2003.

[17] Kim D. Pruitt and Donna R. Maglott. RefSeq and LocusLink: NCBI gene-centered resources. *Nucl. Acids Res.*, 29(1):137–140, 2001.

[18] J. Quackenbush, J. Cho, D. Lee, F. Liang, I. Holt, S. Karamycheva, B. Parvizi, G. Pertea, R. Sultana, and J. White. The TIGR Gene Indices: analysis of gene transcript sequences in highly sampled eukaryotic species. *Nucleic Acids Res*, 29(1):159–64, 2001.

[19] M.G. Reese, G. Hartzell, N.L. Harris, U. Ohler, J.F. Abril, and S.E. Lewis. Genome Annotation Assessment in Drosophila melanogaster. *Genome Res.*, 10(4):483–501, 2000.

[20] SW Scherer, J Cheung, JR MacDonald, LR Osborne, K Nakabayashi, JA Herbrick, AR Carson, Parker-L Katiraee, J Skaug, R Khaja, et al. Human chromosome 7: DNA sequence and biology. *Science*, 300:767–772, 2003.

[21] G. D. Schuler, M. S. Boguski, E. A. Stewart, L. D. Stein, G. Gyapay, K. Rice, R. E. White, P. Rodriguez-Tom, A. Aggarwal, E. Bajorek, et al. A Gene Map of the Human Genome. *Science*, 274(5287):540–546, 1996.

[22] Jay Shendure and George M. Church. Computational discovery of sense-antisense transcription in the human and mouse genomes. *Genome Biology*, 3, 2002.

[23] J.D. Thompson, F. Plewniak, and O. Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999.

[24] Jonathan Usuka, Wei Zhu, and Volker Brendel. Optimal spliced alignment of homologous cDNA to a genomic DNA template. *Bioinformatics*, 16(3):203–211, 2000.

[25] Ru-Fang Yeh, Lee P. Lim, and Christopher B. Burge. Computational Inference of Homologous Gene Structures in the Human Genome. *Genome Res.*, 11(5):803–816, 2001.

[26] Rodrigo Yelin, Dvir Dahary, Rotem Sorek, Erez Y. Levanon, Orly Goldstein, et al. Widespread occurrence of antisense transcription in the human genome. *Nature Biotechnology*, 21:379–385, 2003.

[27] Wei Zhu, Shannon D. Schlueter, and Volker Brendel. Refined annotation of the arabidopsis genome by complete expressed sequence tag mapping. *Plant Physiology*, 132:469–484, 2003.

[28] Judith Zimmermann, Zsuzsanna Lipták, and Scott Hazelhurst. A method for evaluating the quality of string dissimilarity measures and clustering algorithms for EST clustering. In *Proceedings of IEEE Fourth Symposium on Bioinformatics and Bioengineering (BIBE)*, 2004.

BIOINFORMATICS ORIGINAL PAPER

Vol. 00 no. 0 2005, pages 1–6
doi:10.1093/bioinformatics/bti184

A graph based algorithm for generating EST consensus sequences

Ketil Malde[1,*], Eivind Coward[1] and Inge Jonassen[2]

[1]Department of Informatics, University of Bergen and [2]Computational Biology Unit, Bergen Centre for Computational Sciences and Department of Informatics, University of Bergen, Norway

Received on June 28, 2004; revised on September 15, 2004; accepted on November 24, 2004
Advance Access publication November 30, 2004

ABSTRACT

Motivation: EST sequences constitute an abundant, yet error prone resource for computational biology. Expressed sequences are important in gene discovery and identification, and they are also crucial for the discovery and classification of alternative splicing. An important challenge when processing EST sequences is the reconstruction of mRNA by assembling EST clusters into consensus sequences.

Results: In contrast to the more established assembly tools, we propose an algorithm that constructs a graph over sequence fragments of fixed size, and produces consensus sequences as traversals of this graph. We provide a tool implementing this algorithm, and perform an experiment where the consensus sequences produced by our implementation, as well as by currently available tools, are compared to mRNA. The results show that our proposed algorithm in a majority of the cases produces consensus of higher quality than the established sequence assemblers and at a competitive speed.

Availability: The source code for the implementation is available under a GPL license from http://www.ii.uib.no/~ketil/bioinformatics/

Contact: ketil@ii.uib.no

INTRODUCTION

ESTs are produced by one-shot sequencing of cDNAs produced by cloning mRNA. The sequences are easy to produce, and are a useful and versatile resource of gene sequence data. However, they are prone to sequencing errors and vector sequence contamination, and methods for EST analysis need to take this into account.

In order to reconstruct the original mRNA, EST analysis normally entails a clustering stage—attempting to group ESTs according to originating gene, followed by an assembly or consensus stage—aiming to determine for each cluster one or more consensus sequences to which the sequences can be globally aligned. Ideally, one consensus sequence is produced for every mRNA isoform of the gene, but in practice, low coverage regions and high error rates make this goal difficult to attain.

Recent works (Mironov *et al*., 1999; Modrek and Lee, 2001) indicate that alternative transcripts are very common, and consequently, we must expect ESTs in a cluster to contain fragments, not only from a single correct mRNA sequence, but from several equally correct, related sequences.

The presence of multiple correct assemblies contributes to making the assembly of EST data inherently different from assembling genomic DNA.

In addition, EST data are generally considered to be more prone to sequencing errors. However, compared to genome assembly, repeats are less of a problem for assembling an individual gene, since the coding sequence of a gene is unlikely to contain repeats. (It is still an important consideration for EST *clustering*, of course.) The EST assembly problem is thus different enough from genomic assembly to warrant specialized tools (Pertea *et al*., 2003).

The classical approaches to sequence assembly use the 'overlap–layout–consensus' approach; that is, they use pairwise sequence alignments to find a best fit, and build contigs by merging sequences that overlap.

In the ideal case, we can construct a directed graph where each sequence is a vertex, and edges represent overlaps between sequences; i.e. there is an edge from s_1 to s_2 if and only if a suffix of s_1 matches a prefix of s_2. A correct assembly of the set of sequences is then constructed by finding a Hamiltonian path through this graph.

There exist many sequence assemblers that can be used for constructing consensus sequences of ESTs. Some of the more popular are Phrap (Green, 1996), the TIGR Assembler (Sutton *et al*., 1995) and CAP3 (Huang and Madan, 1999). Liang *et al*. (2000) provide a thorough comparison of these tools for the EST clustering problem. However, common to all of these is that they are primarily designed for assembling genomic sequences, rather than ESTs, and recent efforts to improve the state of the art in sequence assembly tend to specialize even more for the genomic assembly problem (Batzoglou *et al*., 2002; Jaffe *et al*., 2003; Myers *et al*., 2000).

*To whom correspondence should be addressed.

A different approach to sequence assembly is based on finding Eulerian paths in a graph. Based on work on 'sequencing by hybridization', the idea is to break down sequencing data into fixed length, overlapping fragments, or k-tuples (Idury and Waterman, 1995). It is then possible to extract a consensus sequence as a path through a graph over these fragments. However, this graph is complicated by read errors, and effective algorithms to eliminate these errors are essential (Pevzner *et al*., 2001).

The advantage of this method is partly efficiency, since it computes the assembly by finding an Eulerian path through a graph, which is computable in linear time, while computing the assembly from the overlap graph is NP-complete (the Hamiltonian path problem).

Since the ESTs in a cluster corresponding to a gene originate from multiple but related mRNA sequences, the graph resulting from applying this approach will partly be determined by the splice structure of the gene, and is consequently sometimes referred to as a *splice graph* (Heber *et al*., 2002).

A splice graph is thus in itself a very descriptive way to present a cluster of ESTs, and can be more revealing of the underlying splice structure than assembled consensus sequences. However, as most tools deal with sequences rather than graphs, it is for many purposes still preferable to represent clusters as consensus sequences.

In the following we present an alternative algorithm for constructing EST consensus sequences. Similarly to the other graph based approaches, we construct a graph over sequence fragments representing the data set. However, we use a different algorithm that does not explicitly try to construct an Eulerian path, nor calculate sequence alignments or overlaps. Instead we traverse the graph greedily, trying to pursue a path through the graph so that each branch followed is as consistent with the previous one as possible.

A complete EST analysis process incorporates many stages, from base calling, quality and vector clipping, and repeat masking, through sequence clustering and assembly, to analysis and detection of transcriptional features like SNPs and splice variants (Staden, 1996; Chevreux *et al*., 2004). We therefore wish to emphasize that the present algorithm and tool only deals with the sequence *assembly* stage, and that other tools must be used for the other parts of the EST analysis tool chain.

BACKGROUND

We define a *word graph* for a data set $S \subset \Sigma^*$ and with word length k as a directed graph (V, E) where the set of nodes V is the set of all $(k-1)$-words in S, the edges $E = \{(xw, wy) | x, y \in \Sigma, xwy \text{ is a } k\text{-word in } S\}$.

For our purposes, S is a set of sequences over the alphabet $\Sigma = \{A, C, G, T\}$, and then nodes then represent all length $(k-1)$-substrings, while edges represent the length k substrings of the sequences in S. We also define the *weight* of an edge as the number of sequences in S containing the word represented by the edge. [The word graph is a subgraph of the *de Bruijn graph*; see, e.g. Skiena and Sundaram (1993) for a more elaborate description.]

Different features of the data set will give rise to corresponding properties of the word graph. For example, a single read error will cause the word graph to branch into paths that merge again after k nodes. Since the chance for the same random error occurring in the same position in two sequences is very low, we expect one of the paths to have a weight of one, and given good coverage, the path representing the correct sequence to have a higher weight. For non-random polymorphisms, it is more likely that the different paths will be supported by several sequences. In addition, if any k-word occurs more than once in a sequence, it will lead to a cycle in the graph.

Word graphs for real data will be complicated by the combination of these features, and the 'raw' graph often becomes a complex tangle of edges, something that moderated the success of the early Eulerian graph sequence assemblers (Pevzner *et al*., 2001).

The present algorithm does not try to construct an Eulerian graph, but instead takes into account the weight of edges to eliminate read errors.

ALGORITHM

Based on the word graph for a set of sequences, we construct consensus sequences by finding *greedy paths* through the graph. The graph is traversed, trying to follow edges so that the path is consistent with at least a subset of the sequences.

The algorithm takes as a parameter the minimum weight threshold, which indicates the number of sequences that need to support a particular edge in the graph in order for us to consider it reliable—or in other words, if the number of sequences supporting an edge is lower than this threshold, we suspect it is due to sequence errors, rather than actual sequence differences. This threshold is used as a stop criterion; when no edges with weights exceeding the threshold remain, the algorithm terminates.

Given a word length k, the algorithm starts out by constructing the corresponding k-word graph. The graph is then examined to identify the heaviest edge—i.e. the k-word occurring in most sequences—and the set of sequences supporting it.

We can now construct the *greedy path* in each direction by traversing the graph until we reach a junction. For each branch out of the junction, we examine the set of sequences supporting it. We select the branch supporting most of the sequences that also support the heaviest edge.

At the next junction, we again examine the set of sequences supporting each branch, and select the one supporting most of the sequences that supported the previously taken branch.

This process is repeated, until we reach a node with no outgoing edges.

One problem that arises is the presence of cycles in the graph. If a word is repeated in a consensus sequence, we risk the danger of looping infinitely. This situation is avoided by maintaining a map of the current position in the sequences. When the consensus is extended, only sequences where the extension would increase the current position are taken into account. This allows the consensus sequence to contain multiple occurrences of the same word, but only if it can be represented by different positions in the sequence data. The map is also used in branch selection, gradually increasing the weight of sequences that align well to the consensus. If a word occurs multiple times in a sequence, its position cannot be unambigously determined in that sequence. In order to avoid this ambiguitiy, the selection of starting point penalizes the weight of multiply-occurring words.

When the greedy path cannot be extended in either direction, it is returned as a consensus sequence, and the next path can then be constructed, starting from the heaviest edge not in any previous sequence.

We used this algorithm to implement an experimental transcript assembly/consensus tool, *xtract*. In this implementation, the word graph is represented by a finite map (a.k.a. dictionary), mapping k-words to sequence and position pairs. This makes it fast [an $O(\log n)$ operation] to check whether a given k-word exists, and to retrieve the list of occurrences in the data set.

Nodes and edges in the word graph can be extracted from this structure; each k-word entry represents an edge from the $(k-1)$-prefix to the $(k-1)$-suffix of the word; the nodes can be inferred implicitly from the edges.

The implementation was written in Haskell and compiled with the GHC compiler.

RESULTS

Since UniGene (Boguski and Schuler, 1995) contains both cDNA ESTs as well as full-length mRNAs, we decided to test the accuracy of our assembly algorithm on the ESTs and use the mRNA as reference. In most cases, sequences are annotated as either mRNA or cDNA, sequences with neither designation being treated as cDNA.

We extracted the 20 427 sequences contained in the first 100 UniGene clusters (Boguski and Schuler, 1995), and masked them using RepeatMasker. Since UniGene uses annotation in addition to sequence information when producing the clustering, the sequences were reclustered based on sequence information only, using xsact (Malde *et al.*, 2003), with a block size of 20 and a match threshold of 75 (i.e. sequences are clustered together if they have exactly matching blocks of length 20 or more, whose sum of lengths is at least 75).

The full length mRNAs (274 sequences, identified by annotation) were then removed from the clusters, leaving only the EST sequences. The largest cluster was comprised of approximately 7500 sequences and 64 mRNAs. The complete distribution of cluster sizes is shown in Figure 1.

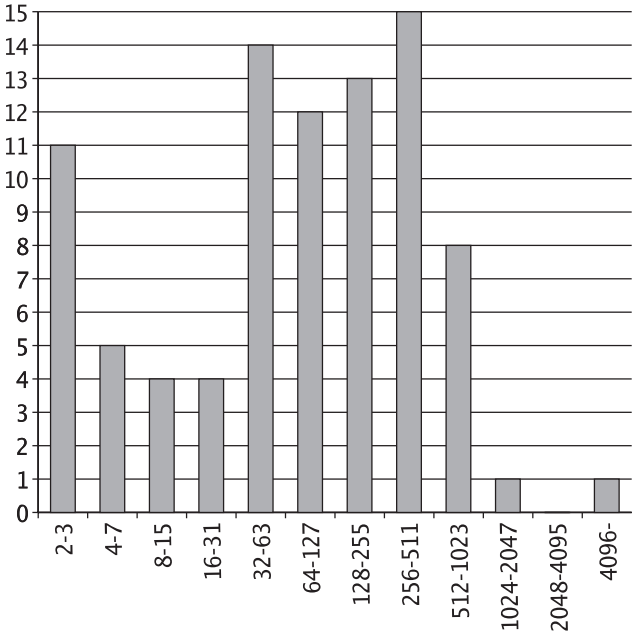

Fig. 1. The distribution of cluster sizes.

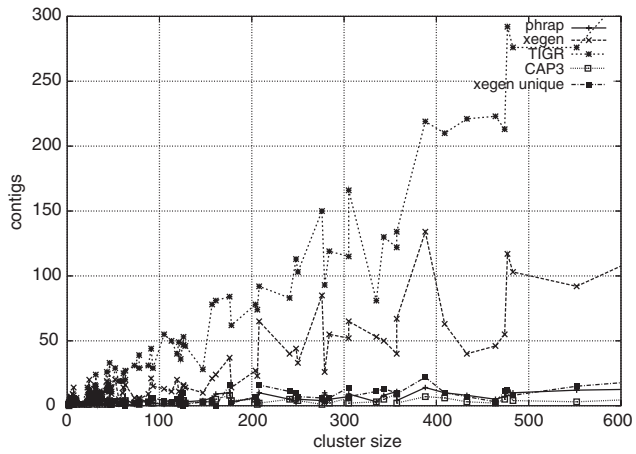

Fig. 2. The number of contigs produced by each sequence assembler plotted against cluster sizes. The line labeled *unique* is the number of sequences after redundancy has been eliminated with *cd-hit* (see Discussion). The largest clusters are not shown in the diagram due to space constraints.

We then ran Phrap, TIGR and CAP3, as well as our algorithm on each of the resulting clusters. For easy comparison, all computations were performed on a Sun Fire 880 computer.

For each cluster, the number of contigs produced by the different programs is presented in Figure 2. The running time

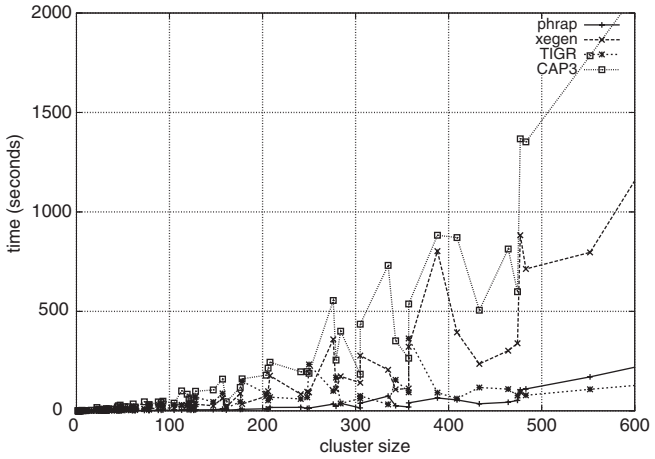

Fig. 3. Running times for each sequence assembler plotted against cluster sizes. The largest clusters are not shown due to space constraints.

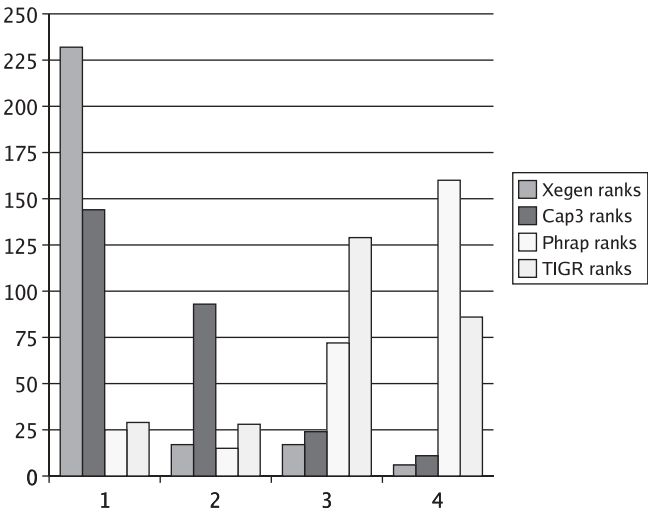

Fig. 4. The ranks of best contig matching each mRNA for each of the sequence assemblers.

of each sequence assembler was measured, and the results are presented in Figure 3.

For each cluster, we compared all the produced contigs against the mRNAs in the cluster. We extracted from the output of each assembler the contig with the highest BLAST bitscore to the mRNA and ranked them. In the case of ties, contigs producing the same score were given the same rank. The ranks were counted for each assembler, and the results are displayed in Figure 4.

The aligment of the consensus sequences for one of the clusters in the data set is shown in Figure 5. Only CAP3 and xtract manage to reconstruct most of the transcript, and CAP3 has a gap of 195 bases. The xsact matches the reference exactly, CAP3 has 2 single-nucleotide errors, TIGR has 13, and Phrap has 17 single-nucleotide errors in the matched regions (not shown).

DISCUSSION AND CONCLUSIONS

We see from Figure 2 that Phrap and CAP3 produce a relatively low number of consensus sequences, while our algorithm and even more so, the TIGR assembler, produces a much larger amount. In many cases, TIGR is close to one contig for every two sequences in the data set. This likely reflects different goals; in particular, xtract will often generate different contigs for SNP variants in the data.

From Figure 3 we see that TIGR and Phrap are very fast, CAP3 is very slow, and our algorithm fluctuates in the middle, generally about twice as fast as CAP3.

The histogram in Figure 4 shows that in a majority of the cases (232 out of 272, or 85% of the total), our algorithm either produced the best contig or tied for the first place.

CAP3 is a clear second with highest-scoring contigs in 144 cases (53%). Among these were a surprisingly high number of ties (128 cases) where it scored the same as our algorithm, so it only succeeded in producing a *better* scoring contig in 16 cases. Both TIGR and Phrap score relatively poorly in our benchmark, with 25 (9%) and 29 (11%) first places, respectively (and roughly half of these were ties with another assembler, sometimes with all four, indicating clusters that are relatively easy to assemble correctly). This agrees well with the results published in Liang *et al*. (2000).

Since our algorithm will produce a set of transcripts so that every word in the input data with sufficient weight is contained in at least one consensus sequence, the number of sequences produced can be large. In order to get a set of consensus sequences more in line with the output of CAP3 and Phrap, we need to reduce the redundancy in the sequences produced by our algorithm. One way to do this is to use cd-hit (Li *et al*., 2001), which reduces a set of sequences to a set of representative sequences, eliminating sequences that have a high degree of similarity to the representatives.

As an experiment, we clustered the consensus sequences with *cd-hit*, and measured the quality and the number of contigs, as above. The results are presented in Figures 6 and 2. Running *cd-hit* was quite fast, and it processed the whole data set in about a minute.

We see that while the number of cases where our algorithm produces the best-scoring contig is reduced, our algorithm still outperforms CAP3, while the number of remaining contigs is more within the range of CAP3 and Phrap.

While *cd-hit* is effective in removing a large amount of redundancy from the data set, it would probably be worthwhile to integrate this functionality in the algorithm itself, in order to be able to better identify potential SNPs, alternative splice variants and other transcript features.

The running time of the algorithm scales proportionally to the output sequence size (each graph traversal is linear in its

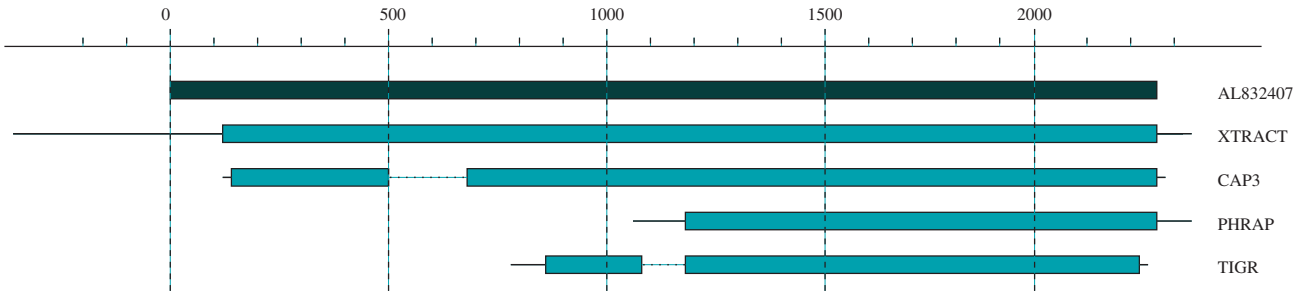

Fig. 5. Alignment of the best-matching consensus sequences against AL832407. The thick regions indicate matches, solid lines indicate non-matching regions, and stippled lines (in the CAP3 and TIGR outputs) indicate parts of the transcript that are missing from the consensus sequences.

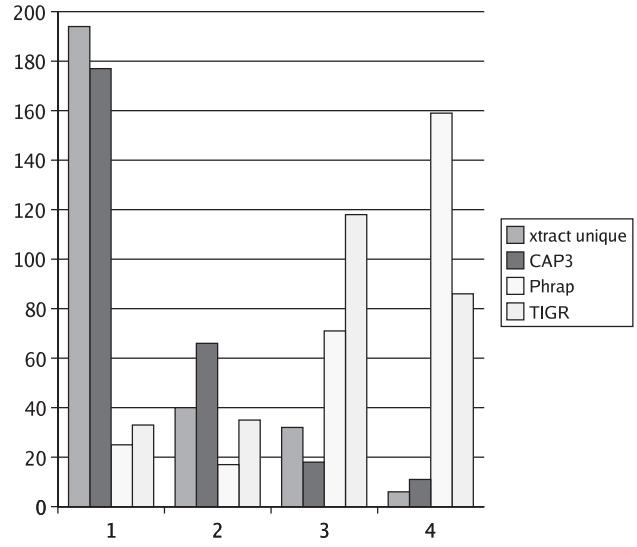

Fig. 6. The ranks of best contig, after processing redundant contigs with *cd-hit*.

length, with logarithmic factors in the graph size). Thus, generating sequences that later will be eliminated is costly, and a modified algorithm that ignores small sequence differences to produce a less redundant set of consensus sequences, should be much faster. As the current algorithm keeps track of where the generated consensus sequence matches the input data, it should be relatively easy to detect potential SNP sites as words occurring in multiple sequences matching a consensus well, but not themselves being part of the consensus.

We believe this experiment shows that this algorithm achieves better quality assemblies than more traditional approaches, with competitive running times.

ACKNOWLEDGMENTS

Phrap is available under an academic license. CAP3 is freely available for academic use, and was provided pre-compiled, along with advice on using it for EST clustering, by its author, Xiaoqiu Hang (pers. comm.). The TIGR Assembler can be downloaded from the TIGR web pages (http://www.tigr.org) as source code.

This work was funded by the Norwegian Salmon Genome Project, a Norwegian Research Council project.

REFERENCES

Batzoglou,S., Jaffe,D.B., Stanley,K., Butler,J., Gnerre,S., Mauceli,E., Berger,B., Mesirov,J.P. and Lander,E.S. (2002) ARACHNE: A whole-genome shotgun assembler. *Genome Res.*, **12**, 177–189.

Boguski,M. and Schuler,G. (1995) ESTablishing a human transcript map. *Nature Genet.*, **10**, 369–371.

Chevreux,B., Pfisterer,T., Drescher,B., Driesel,A.J., Miller,W.E., Wetter,T. and Suhai,S. (2004) Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Res.*, **14**, 1147–1159.

Green,P. (1996) Phrap documentation, http://www.phrap.org/phredphrap/phrap.html

Heber,S., Alexeyev,M., Sze,S.-H., Tang,H. and Pevzner,P.A. (2002) Splicing graphs and EST assembly problem. *Bioinformatics*, **18**, S181–S188.

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

Idury,R. and Waterman,M.S. (1995) A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, **2**, 291–306.

Jaffe,D.B., Butler,J., Gnerre,S., Mauceli,E., Lindblad-Toh,K., Mesirov,J.P., Zody,M.C. and Lander,E.S. (2003) Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Res.*, **13**, 91–96.

Li,W., Jaroszewski,L. and Godzik,A. (2001) Clustering of highly homologous sequences to reduce the size of large protein database. *Bioinformatics*, **17**, 282–283.

Liang,F., Holt,I., Pertea,G., Karamycheva,S., Salzberg,S.L. and Quackenbush,J. (2000) An optimized protocol for analysis of EST sequences. *Nucleic Acids Res.*, **28**, 3657–3665.

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

Mironov,A., Fickett,J. and Gelfand,M. (1999) Frequent alternative splicing of human genes. *Genome Res.*, **9**, 1288–1293.

Modrek,B. and Lee,C. (2001) A genomic view of alternative splicing. *Nature Genet.*, **30**, 13–19.

Myers,E.W., Sutton,G.G., Delcher,A.L., Dew,I.M., Fasulo,D.P., Flanigan,M.J., Kravitz,S.A., Mobarry,C.M., Reinert,K.H. and Remington,K.A. (2000) A whole-genome assembly of drosophila. *Science*, **287**, 2196–2204.

Pertea,G., Huang,X., Liang,F., Antonescu,V., Sultana,R., Karamycheva,S., Lee,Y., White,J., Cheung,F., Parvizi,B., Tsai,J. and Quackenbush,J. (2003) Tigr gene indices clustering tools (tgicl): a software system for fast clustering of large est datasets. *Bioinformatics*, **19**, 651–652.

Pevzner,P.A., Tang,H. and Waterman,M.S. (2001) An Eulerian path approach to DNA fragment assembly. *PNAS*, **98**, 9748–9753.

Skiena,S.S. and Sundaram,G. (1993) Reconstructing strings from substrings. *Lecture Note. Comput. Sci.*, **709**, 565–576.

Staden,R. (1996) The Staden sequence analysis package. *Mol. Biotechnol.*, **5**, 233.

Sutton,G., White,O., Adams,M. and Kerlavage,A. (1995) TIGR assembler: A new tool for assembling large shotgun sequencing projects. *Genome Sci. Technol.*, **1**, 9–19.

INDEX