

Chapter 2

Sequence Comparison Tools

Michael Imelfort

2.1 Introduction

The evolution of methods which capture genetic sequence data has inspired a parallel evolution of computational tools which can be used to analyze and compare the data. Indeed, much of the progress in modern biological research has stemmed from the application of such technology. In this chapter we provide an overview of the main classes of tools currently used for sequence comparison. For each class of tools we provide a basic overview of how they work, their history, and their current state. There have been literally hundreds of different tools produced to align, cluster, filter, or otherwise analyze sequence data and it would be impossible to list all of them in this chapter, so we supply only an overview of the tools that most readers may encounter. We apologize to researchers who feel that their particular piece of software should have been included here. The reader will notice that there is much conceptual and application overlap between tools and in many cases one tool or algorithm is used as one part of another tool's implementation. Most of the more popular sequence comparison tools are based on ideas and algorithms which can be traced back to the 1960s and 1970s when the cost of computing power first became low enough to enable wide spread development in this area. Where applicable we describe the original algorithms and then list the iterations of the idea (often by different people in different labs) noting the important changes that were included at each stage. Finally we describe the software packages currently used by today's bioinformaticians. A quick search will allow the reader to find many papers which formally compare different implementations of a particular algorithm, so while we may note that one algorithm is more efficient or accurate than another we stress that we have not performed any formal benchmarking or comparison analysis here.

The classes of tools discussed below are *sequence alignment*, including sequence homology searches and similarity scoring, *sequence filtering* methods, usually used for identifying, masking, or removing repetitive regions in sequences, and

M. Imelfort (✉)
University of Queensland, Queensland, Australia
e-mail: m.imelfort@uq.edu.au

sequence assembly and *clustering* methods. Sequence annotation tools including gene prediction and marker discovery have been covered elsewhere in this volume and are not discussed here.

2.2 Sequence Alignment

At the most basic level a sequence alignment is a residue by residue matching between two sequences, and algorithms which search for homologous regions between two sequences by aligning them residue by residue are arguably the most fundamental components of sequence comparison. It is also biologically relevant to consider situations where nucleotides have either been inserted into or deleted from DNA; most, but not all, sequence alignment algorithms allow the matching of a residue with a gap element or simply a gap.

Consider two sequences which are identical except that the first sequence contains one extra residue. When we view the alignment of these two sequences, the extra residue will be matched to a gap. This corresponds to an insertion event in the first sequence or a deletion event in the second. On the other hand, if we note that an insertion event has occurred in the first sequence (with respect to the second) then we know how to match that residue to a gap in the second. Thus one way to build a sequence alignment is to find a series of insertions, deletions, or replacements, collectively called *mutation events*, which will transform one sequence into the other. The number of mutation events needed to transform one sequence into the other is called the *edit distance*. As there will always be more than one series of possible mutation events which transform the first sequence into the second, it makes sense to rate each set's likelihood of occurrence. Greater confidence is placed in alignments which have a higher likelihood of occurring. Each alignment can be rated by considering both the cumulative probabilities and biological significance of each mutation event. For example, an alignment which infers a lesser amount of mutations to transform one sequence into another is almost always considered more likely to have occurred than an alignment which infers many more mutations, therefore many alignment algorithms work by minimizing the edit distance.

To resolve the issue of biological significance, information about the distribution of mutation events is used. This information is most commonly stored, in a scoring matrix. Each pair of matched residues (or residue – gap pairs) can be scored and the *similarity score* is the sum of the scores of the individual residues. Most alignment algorithms seek to produce meaningful alignments by maximizing the similarity score for two sequences. Traditionally, sequence alignment algorithms have been called *global* if they seek to optimize the overall alignment of two sequences. Often the resulting alignment can include long stretches of residues which are matched with different residues or gaps. Conversely, if the algorithm seeks to align highly conserved subsequences while ignoring any intervening unconserved regions then it is called a *local* alignment algorithm. A local alignment of two sequences can produce a number of different subsequent alignments. So far, only the case where

two sequences are being compared has been described. This is called a *pairwise* alignment. The case where more than two sequences are being compared concurrently is called a *multiple* alignment.

Alignment algorithms can be broadly classified as taking a heuristic or dynamic programming approach. Generally, dynamic programming based approaches are guaranteed to produce the best alignments but are very often computationally and memory expensive, while heuristic based algorithms sacrifice guaranteed quality for speed. Note that a heuristic algorithm can produce an optimal alignment; there is just no guarantee that it will. Often dynamic programming approaches are used to *finish* or perfect alignments made using heuristics.

2.2.1 Substitution Matrices

When two sequences are aligned there are often residues in one sequence which do not match residues in the other. There is usually more than one way to align two sequences, so a scoring system is needed to decide which of the possible alignments is the best. For a nucleotide alignment, a simple scoring system could award one point for every match and zero points for a mismatch or a gap. This information can be stored in a matrix called a *substitution* matrix. An example of such a matrix is shown as the first matrix in Fig. 2.1 below. This is the substitution matrix employed with good results in the original Needleman–Wunsch algorithm (Needleman and Wunsch 1970). However, this matrix was criticized as lacking in both biological relevance and mathematical rigor and there have been a number of attempts to improve on both this and some earlier methods resulting in the scoring systems used today. In 1974, Sellers introduced a metric which could be used to describe the evolutionary distance between two sequences (Sellers 1974) and this method was generalized by Waterman et al in 1976 (Waterman et al. 1976). The idea behind these scoring systems was to minimize the number of mutations needed to transform one sequence into the other while also taking into account the differing probabilities of distinct mutation events. For example, a more sophisticated scoring system could award negative scores for gaps (a *gap penalty*) and for mismatches the scores

	A	C	G	T
A	1	0	0	0
C	0	1	0	0
G	0	0	1	0
T	0	0	0	1

	A	C	G	T
A	3	-2	-1	-2
C	-2	3	-2	-1
G	-1	-2	3	-2
T	-2	-1	-2	3

Fig. 2.1 Two examples of nucleotide similarity matrices. The first matrix implements a binary scoring scheme awarding one point for a match and one for a mismatch. The second matrix introduces a more sophisticated method where biological observations such as the unequal probabilities of transitions and transversions influence the score

could differ according to whether the mismatch was a transition event or a transversion event. An example of this is the second matrix in Fig. 2.1 below.

For protein alignments, each row and column in the substitution matrix S corresponds to a particular amino acid, where each entry S_{ij} contains a value representing the probability of substituting the residue in row i for the residue in column j . The most widely used examples of such matrices are the point accepted mutation (PAM) matrices (Dayhoff 1978) and Block substitution matrices (BLOSUM) (Henikoff and Henikoff 1992). Both matrices share many similarities. They are both 20×20 matrices and in both cases identities and conservative substitutions are given high scores while unlikely replacements are given much lower scores. Both matrices are assigned numbers which identify when they should be used, for example PAM30 or BLOSUM62. However, one should use a higher numbered BLOSUM matrix when comparing more similar sequences while for PAM matrices lower numbers should be used. A more important difference is the way the matrices are built. PAM matrices are derived from an explicit model of evolution and based on observations of closely related protein sequences, while BLOSUM matrices are based directly on observations of alignments of more distantly related sequences using a much larger dataset than for PAM. As a result the BLOSUM matrices tend to produce better results than PAM matrices, particularly when aligning distantly related sequences.

Work on the PAM matrix model of protein evolution was undertaken by Dayhoff in the late 1970s (Dayhoff 1978). The main idea behind the PAM matrices is that of all possible mutations; we are going to observe only those which are accepted by natural selection. PAM1 was calculated using the observed relative frequencies of amino acids and 1,572 observed mutations in multiple alignments for 71 families of closely related proteins. Each entry in PAM1 represents the expected rates of amino acid substitution we would expect if we assume that on average only 1% of the residues in one sequence have mutated (Dayhoff 1978). By assuming that further mutations would follow the same pattern and allowing multiple substitutions at the same site, one can calculate the expected rates of substitution if we assume on average that 2% of the residues have mutated. This is the PAM2 matrix. Thus all the PAM matrices are calculated from the PAM1 matrix and are based on an explicit model of evolution based on point mutations. Matrices were calculated by Dayhoff up to PAM250.

The PAM approach performs well on closely related sequences but its performance declines for more distantly related sequences. The BLOSUM matrices were derived by Steven and Jorja Henikoff in the early 1990s to address this problem. To build a BLOSUM matrix, local alignments are made using sequences obtained from the BLOCKS database. Sequences with a similarity greater than a given cut off are combined into one sequence producing groups with a given maximum similarity. This reduces any bias caused by large numbers of highly similar sequences (Henikoff and Henikoff 1992). The value for the cut off is appended to the name of the matrix, thus the BLOSUM62 matrix is effectively made by comparing sequences with less than 62% similarity. As a result BLOSUM80 is a better matrix to use when aligning closely related sequences than BLOSUM30 which is better

suited to aligning highly diverged sequences. BLOSUM62 is the default matrix used in the BLAST algorithm described below.

2.2.2 *Pairwise Sequence Alignment Algorithms*

At the base of many sequence comparison tools are pairwise sequence alignment algorithms. Beginning in the mid 1960s a large number of heuristic algorithms were suggested for the pairwise alignment of protein sequences. The era of modern sequence alignment techniques began in 1970 with the publication by Needleman and Wunsch of a dynamic programming method which could be used to make a global pairwise alignment of two protein sequences (Needleman and Wunsch 1970). In 1981, Smith and Waterman extended the ideas put forward by Needleman and Wunsch to create the local alignment algorithm known as the Smith–Waterman algorithm (Smith et al. 1981). Both the Needleman–Wunsch and Smith–Waterman methods belong to a class of algorithms called dynamic programming algorithms. This class of algorithms can find optimal solutions to problems but can take a long time to run, especially in complicated cases or for large data sets. These two algorithms are the most accurate pairwise alignment algorithms in existence. Nearly all of the newer local pairwise alignment algorithms use a two step approach to reduce the running time. The first stage uses heuristics to search for areas which have a high probability of producing alignments. Next, these areas are passed to a dynamic programming algorithm such as the Smith–Waterman algorithm for true alignment. The most commonly used two step approaches are FASTP/FASTA, the BLAST family of algorithms, Crossmatch/SWAT, and BLAT, although there are many others.

Higher order sequence comparison tools often employ pairwise alignment algorithms to judge similarity for use in clustering or assembly, so it is important to understand how these basic algorithms work and which sequences they are better suited to. We provide below an overview of the most common pairwise alignment algorithms.

2.2.2.1 *The Needleman–Wunsch Algorithm*

This is a highly accurate, dynamic programming based, global pairwise alignment algorithm. It was originally developed for aligning protein sequences but can also be used to align DNA sequences. This algorithm aligns two sequences A and B with lengths m and n residues respectively, by finding a path through a two dimensional $m \times n$ array; S . As all $m \times n$ values in S must be calculated for every alignment, the work needed to align two sequences becomes intractable for large m and n . For the following example, we assume the use of the simple nucleotide similarity matrix in Fig. 2.1. First the bottom right cell $S_{m,n}$ is assigned the value 1 or 0 depending on whether the base in position m of A matches the base in position n of B. The cell diagonally above and to the left of this cell; $S_{m-1, n-1}$, is given a value

of 2 if the base in position $m - 1$ of A matches the base in position $n - 1$ of B or a value of 1 otherwise. This is because a match will produce a maximum run of matches of length 2 for all bases from this point on, whereas a mismatch will produce a run of at most one match. The algorithm continues working backwards until every cell has been assigned a value. Finally the algorithm starts from the highest scoring cell in the array, and finds a path through the array which maximizes the cumulative sum of the values in the cells visited in the path. The resulting path represents a *maximally matching* global alignment.

2.2.2.2 The Smith–Waterman Algorithm

In 1981 Smith and Waterman extended the ideas presented by Needleman and Wunsch to create an algorithm which is capable of finding optimal local pairwise alignments. The method uses a distance metric introduced by Sellers in 1974 which can be summarized by a matrix similar to the second example in the Fig. 2.1 (Smith et al. 1981). This algorithm uses a method similar to that of Needleman and Wunsch; first filling in all the values for an $m \times n$ matrix based on the score for a maximum length run of matches and then finding a path through the matrix. There are two main differences between the Smith–Waterman algorithm and the Needleman–Wunsch algorithm. The first is that the matrix is completed from the top left cell downwards as opposed to the backtracking done by Needleman and Wunsch. The second is that the path is built by finding the maximal valued cell in the matrix and then backtracking until a zero is found. The resulting path represents an alignment of two segments, one from each sequence. Note that while not all the bases in both sequences are aligned, there can be no other pair of segments which will produce a higher score. The algorithm was modified by Gotoh to include affine gap penalties (Gotoh 1982) and is sometimes called the Smith–Waterman–Gotoh algorithm. This algorithm is without doubt the cornerstone of modern sequence comparison.

2.2.2.3 SWAT and CrossMatch

Unlike many other *fast* pairwise algorithms, SWAT does not employ first stage heuristics to speed up the Smith–Waterman algorithm. Instead, the authors of SWAT focused on speeding up the code itself by revising recursion relations and making efficient use of word-packing. This resulted in a significant reduction in the number of machine instructions executed per Smith–Waterman matrix cell. Thus they have produced a raw implementation of the Smith–Waterman–Gotoh algorithm which is about one tenth as fast as BLAST. SWAT is normally used to search query sequences against a sequence database or as an engine in other sequence comparison tools.

CrossMatch is a general-purpose sequence comparison utility based on SWAT and is used for comparing sets of DNA sequences. CrossMatch uses the same

algorithm as SWAT, but allows the use of heuristics to constrain the comparison of pairs of sequences to bands of the Smith–Waterman matrix that surround one or more matching words in the sequences. This step reduces the running time for large-scale nucleotide sequence comparisons without significantly compromising sensitivity. CrossMatch and SWAT form the kernel of the Phrap assembly program and CrossMatch is used as the comparison engine in RepeatMasker. Both Phrap and RepeatMasker are described in more detail below. SWAT and CrossMatch are unpublished software; however, information can be found at: www.genome.washington.edu/UWGC/analysisistools/Swat.cfm.

2.2.2.4 The BLAST Family of Algorithms

BLAST (Altschul et al. 1990) and its many derivatives are arguably the most widely used pairwise local alignment algorithms. The BLAST algorithm attempts to heuristically optimize a measure of local similarity called the maximal segment pair (MSP). The MSP is defined as the highest scoring pair of identical length segments chosen from two sequences. To enable the reporting of multiple local alignments BLAST can also return other *locally* maximal segment pairs. Put simply, the speed of the BLAST algorithm is mainly due to its ability to identify and divert resources away from areas in the query sequences which have very little chance of producing high scoring alignments. Most BLAST implementations enable the user to search a pre-compiled database for high scoring segments in a set of query sequences. The database is created by running the program `formatdb` which produces a set of files that have been optimized for size and speed of searching. The algorithm has three distinct steps. First, using the information in the database and the query sequence, the algorithm compiles a list of high scoring words of a set length k (k -mers) from the query sequence. The database is scanned for matches to the words and where these hits occur, the algorithm tries to extend the hit to the left and right. BLAST uses a minimum score cutoff when assessing word hit quality to filter out any hits which could have occurred due to random chance. Note that the BLAST algorithm is characterized by the creation of k -mer lists for each query sequence and a linear search of the entire database for words in these lists. The BLAST algorithm has been highly successful and there are many different implementations available which have been adapted to better suit particular applications.

2.2.2.5 BLAT

BLAT stands for BLAST-Like Alignment Tool and was developed by James Kent for use in the annotation and assembly of the human genome (Kent 2002). Kent was given the task of aligning many millions of mouse genomic reads against the human genome. He found that when using BLAST, the need to calculate high scoring k -mer lists for each query sequence and the linear nature of the database search proved too slow. To solve this problem, BLAT creates an indexed list of all possible

non-overlapping k-mers from sequences in the database. BLAT then compiles a list of all overlapping k-mers from each query sequence and attempts to find these in the database. In the regions where multiple perfect hits occur, BLAT performs a Smith–Waterman alignment of the two sequences. This allows BLAT to maintain relatively high sensitivity, although it must be noted for example that TBLASTX can be configured to be more sensitive to distant relationships than BLAT. The reduced sensitivity is compensated for by the fact that BLAT can be run up to 50 times faster than TBLASTX (Kent 2002).

2.2.3 Multiple Sequence Alignment Algorithms

It is often necessary to produce an alignment of a group of three or more sequences. Examples include the comparison of the evolutionary distances between protein sequences, the evaluation of secondary structure via sequence relationship, or the identification of families of homologous genes. Efforts have been made to extend dynamic programming pairwise alignments to handle three or more sequences (Murata et al. 1985). However, the computational complexity of handling more than 4 sequences proved too much for the available computing power. Many modern multiple alignment algorithms use a method first suggested in 1987 by Feng and Doolittle called the progressive method (Feng and Doolittle 1987). The underlying assumption used in constructing the progressive method is that sequences with a high level of similarity are evolutionarily related. Given a set of sequences to be aligned, Feng and Doolittle use the Needleman–Wunsch pairwise alignment algorithm to calculate rough evolutionary distances between every pair of sequences and these are used to create a reference phylogenetic tree. Starting from the two closest branches on the tree, a pairwise alignment is made and a consensus sequence is produced which is used as a substitute for the branch. This is continued for the next closest pair of branches until all the sequences have been added and the alignment is complete.

The intermediate pairwise alignments may include two of the query sequences, one query sequence and one consensus sequence or two consensus sequences. It is important to note that the order in which sequences are added will affect the ultimate alignment and it is very difficult to repair the damage caused to the overall quality of an alignment if a less than optimal choice is made early on. However, algorithms such as MUSCLE attempt to do this. The use of a reference tree helps ensure that closely related sequences are aligned before distantly related sequences. Thus the progressive method utilizes a greedy algorithm. Feng and Doolittle stressed the point that any gaps added to the alignment in earlier stages must remain, creating the rule “once a gap, always a gap” (Feng and Doolittle 1987). This ensures that distantly related sequences cannot disturb meaningful alignments between closely related sequences, however some implementations of the progressive method do not follow this rule. Finally it is important to note that the reference tree should not be used to infer phylogenetic relationships, as there is a high probability that the tree is erroneous

(in that sense). However, a new tree (or set of trees) can be made with the resulting multiple alignment and this can be used to study phylogeny. There are a number of multiple alignment algorithms based on the progressive method. The most widely used are the CLUSTAL family of algorithms, MUSCLE and T-Coffee.

2.2.3.1 The CLUSTAL Family of Algorithms

The CLUSTAL family of multiple alignment algorithms includes the original program CLUSTAL as well as CLUSTAL V and CLUSTAL W. All of the CLUSTAL derivatives are based on the progressive method (Higgins and Sharp 1988; Higgins et al. 1992; Thompson 1994). The original CLUSTAL package was released as a collection of different pieces of software with each one performing one stage of a progressive alignment. CLUSTAL V was a rewrite of this system which combined all the packages into one program. CLUSTAL W is a further update to CLUSTAL V which incorporates sequence weighting, position-specific gap penalties, and weight matrix choice. For the rest of this section we describe only the features of CLUSTAL W.

Highly similar sequences will be positioned very closely on the reference tree and consequently will be added to the alignment much earlier than divergent sequences. Too many highly similar sequences in the query set can create bias in the topology of the reference tree which can lead to future alignment errors (Higgins and Sharp 1988). Sequence weighting attempts to reduce this bias by down-weighting groups of similar sequences and up-weighting divergent sequences. This feature reduces the negative impact that the topology of the reference tree can have on the final alignment (Thompson 1994). When the algorithm starts, it can use gap penalties and substitution matrices as supplied by the user. CLUSTAL W provides a choice of PAM or BLOSUM matrices with the default being BLOSUM. As the algorithm progresses, CLUSTAL W adjusts the gap penalties according to the position, content (hydrophilic or hydrophobic regions) and length of the sequences. CLUSTAL W also adjusts the weight of the substitution matrix based on the estimated evolutionary distances obtained from the reference tree. These additions to the CLUSTAL algorithm reduce the negative impact of sub-optimal parameter choices made by the user.

CLUSTAL W is the most widely used multiple sequence alignment algorithm and represents an acceptable balance between speed and accuracy. The next two algorithms are faster and more accurate respectively. The first, MUSCLE, sacrifices some accuracy for significant gains in speed, while the second, T-Coffee, makes significant gains in accuracy for a modest sacrifice in speed.

2.2.3.2 MUSCLE

MUSCLE is a very fast multiple sequence alignment algorithm based on the progressive method. The algorithm is split into three phases. The first is typical of a

progressive algorithm except that instead of using an alignment algorithm to generate the reference tree and evolutionary distances, MUSCLE employs the faster method of k-mer counting to judge similarity (Edgar 2004). Once the preliminary tree has been built, MUSCLE progressively adds sequences to the multiple alignment following the branching order, with closer branches being added first. At this stage, a new tree can be constructed and the progressive alignment can be returned to the user. The second phase seeks to improve the results of the first by iteratively constructing progressive alignments in the same manner as the first stage but using the most recent tree generated from the previous progressive alignment. At the end of each iteration, a new tree is made for use in the next round or phase. The third and final phase performs iterative refinement on the tree produced in the second phase. At each iteration, the tree is first separated into two pieces by removing an edge. Superfluous indels (insertions or deletions) are removed from each of the partial multiple alignments and then the tree is rejoined by re-aligning the partial multiple alignments. MUSCLE can produce multiple alignments achieving accuracy similar to CLUSTAL W but two to three orders of magnitude faster. Thus MUSCLE is suited to fast alignment of large sequence datasets.

2.2.3.3 T-Coffee

Nearly all progressive based multiple alignment algorithms employ a greedy algorithm for adding sequences to the alignment. Unfortunately, errors can occur if the sequences are added in a less than ideal order. T-Coffee is an implementation of the progressive method which attempts to rectify some of the problems associated with the greedy approach to progressive alignment while minimizing speed sacrifices. To achieve this, T-Coffee first builds a library of both global and local pairwise alignments between all the query sequences. T-Coffee uses the progressive method, but in contrast to the algorithms described above, it attempts to consider the effects on every query sequence for each sequence being added. This approach seems to have worked as, on average, T-Coffee produces more accurate alignments than the competing algorithms (Notredame et al. 2000). However, this comes at the cost of increased running time, so T-Coffee may not be suited to the task of aligning large datasets.

2.3 Filtering, Clustering, and Assembly

This section covers the area of sequence filtering and the related areas of sequence clustering and sequence assembly. There is a great deal of overlap in the methods used for both sequence assembly and clustering. Pre-filtering reads and masking low complexity areas can improve the performance of assembly and clustering algorithms and is often a first step in many assembly/clustering pipelines.

2.3.1 Filtering and Masking

The first phase for many sequence comparison algorithms is filtering or masking regions whose presence will reduce the efficacy of tasks further down the pipeline. For example, consider the process of automated sequence annotation. One task involves querying the sequence to be annotated against a database of sequences with high confidence annotations (usually performed by making pairwise alignments). If the query sequence contains a substring which is common to many, largely unrelated or loosely related sequences, then the algorithm may return a large number of matches to sequences in the database which do not reflect meaningful annotations. These common elements are usually called *repetitive*, *repeats* or *low complexity sequences*. For sequence assembly, finding overlaps between reads is a fundamental task, and spurious overlaps caused by low complexity sequences can severely impede an assembly program's ability to produce accurate contigs.

Masking repetitive regions usually involves replacing all of the nucleotide bases in the repetitive region with another generic character, usually an "X" or an "N." The majority of assembly and alignment programs ignore these characters by default. In this way, results made by comparing masked sequences are usually more accurate than those where masking has not been performed. Masking can also decrease the running time of sequence comparison algorithms by reducing the number of possible alignments.

Another form of pre-filtering is sequence trimming. Often DNA sequences will begin, and possibly also end, with nucleotide bases from the vector used in the cloning stage, and for many different types of reads, the quality of the data decreases towards the end of the sequence read. An easy way to overcome these problems is to trim the ends of the sequence reads. There are a number of programs which can be used to trim sequences, however they are not discussed here. Finally, reads which contain low amounts of information can simply be removed from the data set, for example if the majority of the read consists entirely of As or Ns. When raw quality values are available, it is also common to simply discard reads whose overall quality is below a certain threshold.

2.3.1.1 RepeatMasker

RepeatMasker screens DNA sequences for interspersed repeats and low complexity DNA sequences. The output of the program is a detailed annotation of the repeats present in the query sequence as well as a modified version of the query sequence where all the annotated repeats have been replaced by Ns or Xs. RepeatMasker draws information about which regions are repetitive by comparing the query sequences to a curated database of repeats. RepeatMasker uses CrossMatch for this task (Smit et al. 1996)

2.3.2 Sequence Clustering

With the quantity of sequence data contained in online repositories increasing at an accelerating pace, tools that can cluster related sequences into meaningful groups provide a way for researchers to efficiently sort through and make sense of this mountain of data. Many researchers are interested in clustering reads from expressed sequence tag (EST) datasets in the hope of identifying the full length genes which the ESTs represent. Another application of clustering is the identification of single nucleotide polymorphisms (SNPs). Clustering is often used to reduce redundancy in a dataset. For example, the BLOSUM substitution matrices use clustering of similar sequences as a first step to reduce the negative effect caused by including too many highly similar sequences. Clustering can also be useful as a first step in sequence assembly pipelines. Sequence assembly programs will often perform significantly better when run multiple times on sets of closely related sequences than when attempting to assemble the whole data set as one chunk. This approach can also significantly reduce the running time of the assembler. Clustering algorithms typically take as input a set of reads to be sorted and input parameters specifying the degree of similarity required for reads to be grouped together. The output is a grouping of the reads that match these criteria.

Most clustering algorithms use an agglomerative approach. At the start of the algorithm, each sequence is effectively in its own group. The algorithm successively merges groups if the similarity criteria are met, and repeats this process until no more merges are possible. These final merged groups are then returned to the user. Depending on user input or the algorithm itself, two groups will be merged when there exists a single pair of sequences (one sequence from each group) which match the similarity criteria. This is referred to as *single linkage* clustering or *transitive closure*. It is sometimes possible to raise the minimum number of pairs needed for merging to occur. If every possible pair of sequences from both groups must match the similarity criteria for merging to occur then this is called *complete linkage* clustering. Complete linkage clustering typically produces many small, high quality clusters, whereas single linkage clustering typically produces fewer, larger, lower quality clusters. Depending on the application, one approach may be more favorable than the other.

The similarity criterion for clustering is usually stated in terms of the minimum overlap and minimum percentage identity. This is sometimes augmented by limiting the maximum number of mismatches allowable. There are two main approaches available to find overlaps. The first uses information gathered from successive pairwise alignments, effectively looking at the edit distance, the number of mutation events needed to describe the distance. The second uses a k-mer counting approach, where the presence of multiple identical words is used to infer an overlap. Both methods perform well, but the k-mer counting approach has been proved to handle sequencing errors better. Furthermore, the k-mer counting approach can be implemented in linear time whereas the edit distance approach can only be as fast as the underlying alignment algorithm, which in the case of Smith–Waterman is quadratic,

and slightly better for BLAST-like algorithms. Two popular clustering algorithms are WCD (Hazelhurst et al. 2008) and d2_cluster (Burke et al. 1999).

2.3.3 Sequence Assembly Overview

The greatest challenge to sequencing genomes is the vast difference in scale between the size of the genomes and the lengths of the reads produced by the different sequencing methods. While there may be a 10–500-fold difference in scale between the short reads produced by next generation sequencing and modern Sanger sequencing, this still dwarfs the difference between the Sanger read length and the lengths of complete chromosomes. For example, human chromosomes vary between 47 and 245 million nucleotides in length, around 50,000–250,000 times longer than the average Sanger reads. For all technologies, the challenge is the assembly of sequence reads to produce a representation of the complete chromosomes. Whether this challenge is significantly greater for short reads is being hotly debated.

The first sequence fragment assembly algorithms were developed in the early 1980s. Early sequencing efforts focused on creating multiple overlapping alignments of the reads (typically the Sanger sequence reads) to produce a layout assembly of the data. A consensus sequence is read from the alignment and the DNA sequence is inferred from this consensus. This approach was referred to as the *overlap-layout-consensus* approach and culminated in a variety of sequence assembly applications such as CAP3 and Phrap. Previous generation DNA sequencing has produced relatively long, high quality reads which were amenable to assembly using the overlap-layout-consensus approach.

From the late 1980s to the mid 1990s research began to focus on formalizing, benchmarking, and classifying fragment assembly algorithm approaches. Three papers, (Pevzner 1987, Myers, 1995, Idury and Waterman 1995) formalized the approach of placing sequence reads or fragments in a directed graph. Myers focused on formalizing the traditional overlap-layout-repeat method while Pevzner and Idury and Waterman developed a new method for solving the assembly problem. While both methods involved the construction of graphs, they differed in that whereas the fragments in Myers graph (Myers 1995) are represented as nodes, the fragments in Pevzner's and Idury and Waterman's graphs (Pevzner 1987; Pevzner 2001; Idury and Waterman 1995) are represented as edges.

2.3.3.1 The Overlap Graph Method

The overlap graph method was formalized by Myers in 1995 (Myers 1995) and is referred to here as the Myers method. In this graph, each vertex represents a read or fragment, and any two vertices are joined by an edge if the two fragments overlap (often imperfectly with some level of significance set by the user). Next, the graph is simplified by the removal of transitive edges and contained nodes

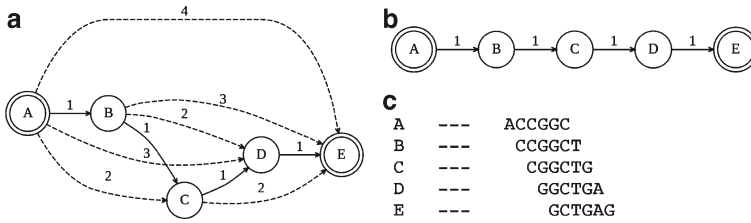


Fig. 2.2 Removal of transitive edges from the overlap graph. (a) The original graph with numbers depicting offsets in the alignment of equal length reads. The dashed lines are transitive edges. (b) The simplified graph. (c) The short reads and their alignment as given by the graph yields the sequence ACCGGCTGAG

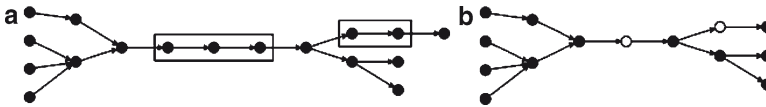


Fig. 2.3 The collapsing of linearly connected sub graphs into single nodes greatly reduces the complexity of the overlap graph. (a) The original graph which contains linearly connected sub graphs. (b) The simplified graph with white nodes representing “chunks”

which add little or no information. The removal of transitive edges is shown in Fig. 2.2. Contained nodes occur when the graph is made from reads of different lengths and one read is completely contained within another.

Finally, chains of nodes or linearly connected sub graphs are collapsed into “chunks” which themselves are treated as single nodes in the graph. This is shown in Fig. 2.3. These graph simplification methods are very effective for reducing the computational complexity of assembly and many modern day algorithms employ these methods. Once the graph has been simplified, the Myers method finds a maximum likelihood non-cyclic (Hamiltonian) path through the graph and infers the layout of the fragments from this path.

2.3.3.2 The Eulerian Path Method

Idury and Waterman proposed an algorithm which could be used to assemble data generated in sequencing by hybridization (SBH) experiments (Idury and Waterman 1995). Although the mathematics for it was developed by Pevzner in 1989 (Pevzner 1989), this is the first algorithm developed using this approach. We refer to the combined ideas of Idury and Waterman and Pevzner as the IWP method. The main application of SBH is now gene chips and not genome sequencing, but the ideas described in the IWP model can be seen in a number of sequence assembly algorithms, most notably the EULER algorithms developed by Pevzner in 2001 (Pevzner 2001). In Idury and Waterman’s algorithm, sequence fragments are broken down

into every possible read of some length k (k is very small, approximately 10 bases) referred to as k -tuples. The set of all k -tuples found is often referred to as the spectrum of reads (Pevzner 2001). In Idury and Waterman's model, assembled sequences are represented as paths through a de Bruijn graph where each node in the graph is a k -1 tuple. Two nodes X and Y are joined by a directed edge if there exists a read R in the spectrum where the first $k-1$ bases of R match X and the last $k-1$ bases of R match Y . Thus it follows that if two edges are adjacent in the graph they will have a perfect overlap of $k-1$ bases. It is important to note that this model only finds perfect overlaps, while the Myers method can accept imperfect overlaps. An example of such a de Bruijn graph is shown in Fig. 2.4. Here the graph is for the sequence CAGTCGAGTTCTCTG with k equal to 4. Erroneous reads cause the inclusion of extra edges which can cause "tangles" in the graph. The dashed edge from TTC to TCG is due to the erroneous read TTCG being included in the spectrum. Idury and Waterman (Idury and Waterman 1995) describe a number of graph simplifications which can remove errors from the graph.

Assembly is achieved by finding an Eulerian path in the de Bruijn graph. That is a path which visits every edge exactly once. It is well known that the problem of finding Hamiltonian paths in a graph is NP hard whereas the problem of finding Eulerian paths is relatively easy. However, the theoretical advantage that the IWP method seems to have over the method of Myers has not translated into great computational or time savings. This is mainly due to the fact that heuristics have been employed to speed up the latter. The main problem with the Eulerian Path approach is that errors in real data cause the inclusion of extra edges, causing tangles. When there are too many errors in the data, the graph becomes entangled, and as a result the algorithm cannot be scaled up. An example of how erroneous reads cause graphs to become entangled is given in Fig. 2.4. In 2001, Pevzner successfully applied the method of Idury and Waterman to read sets with errors by developing an error correction algorithm which could reduce the number of errors by approximately 86% (Pevzner 2001). Pevzner introduced a number of transformations which simplify the graph and these transformations have a conceptual overlap with Myers simplifications. One transformation replaces a number of consecutive edges by one edge, in a way which mimics the collapse of linearly connected sub graphs described above. This process of edge formation/simplification is performed

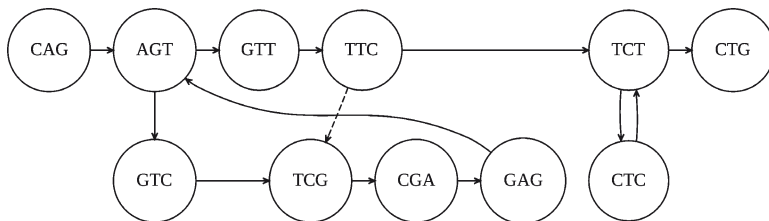


Fig. 2.4 A de Bruijn graph for the sequence CAGTCGAGTTCTCTG. The erroneous read TTCG has been included in the spectrum causing the inclusion of the *dashed edge*. Erroneous edges cause the graph to become entangled

at the beginning of the assembly so that only the minimal number of edges possible need be processed.

The two methods described above have formed the foundation for modern assembly approaches, and all modern sequence fragment assemblers include variations of these concepts, and in some cases algorithms may borrow from both methods.

2.3.3.3 Problems of Assembling Complex Genomes

One challenge of genome sequencing lies in the fact that only a small portion of the genome encodes genes, and that these genes are often surrounded by repetitive DNA which is comparably information poor. Large repeats can cause ambiguity with fragment assembly and thus pose the greatest challenge when assembling genomic data. In Fig. 2.5, we know that regions B and C are surrounded by identical repetitive regions X and that both regions lie between regions A and D, but without more information, it is impossible to know the correct ordering of B and C.

The traditional method to overcome the problems created by large repeats when assembling sequence reads is to increase the read length to such a point that every repeat is spanned by at least one read. In practice however, this is simply not possible as these repeats are frequently longer than the current Sanger read length. Modifications to the original shotgun method that attempt to overcome this problem try to increase the “effective” read length. These include using paired end sequencing, where DNA fragments of known approximate size are generated and sequenced from both ends. Information about these pairs such as average fragment size and the orientation of reads with respect to the read pair is included in the assembly process (Pevzner 2001). If the distance between the paired ends, known as the insert size, is large enough, then there is a high probability that repeats will be spanned by a pair of reads (or mates) which can remove ambiguity from the assembly. For example, if paired end data is analyzed and region B is found to have mates in regions A and C but not D, while region C has mates in regions B and D but not A, then an ordering can be inferred. This is shown in Fig. 2.6. Note that if the insert size was too large and paired ends from B reached over to region D while the paired ends of C reached over to region A there would still be doubt as to how these reads should be arranged. There is also a problem if the insert size is too small and paired reads do not reach past the repetitive region. To address these issues, a number of different size fragment libraries are often used.

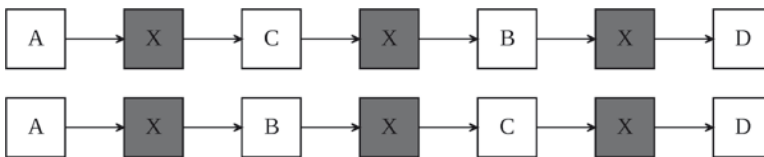


Fig. 2.5 An example of how repeats cause ambiguity in assembly. Because both fragments B and C are surrounded by repetitive region X, there is no way to know their ordering in the assembly

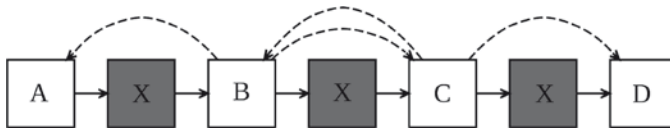


Fig. 2.6 Resolution of ambiguities using paired end data. *Solid edges* indicate overlaps while *dashed edges* show links between reads in a region and the region(s) containing the paired mate

Errors in data further exacerbate the problem of resolving repetitive regions as it is often difficult to differentiate between reads from slightly different repetitive regions and reads from the same region that contain errors. This can cause a problem called over-collapsing, where multiple copies of a repeat will be assembled on top of each other. Although we would expect a significant increase in read depth for contigs, which are made from over collapsed regions, the read depth is frequently variable across the genome and is therefore an unreliable indicator of repeat regions. Both paired end data and various statistical methods have been applied in an attempt to solve the problem of assembling short read sequence data and these are described in more detail below.

2.3.4 Traditional Fragment Assembly Algorithms

For many years, the vast majority of DNA sequence data has been produced using variations of the chain termination method first introduced by Sanger in 1977 (Sanger et al. 1977). The Sanger sequence reads are typically 700–1,000 bases long and of high quality. The individual nucleotide bases in a sequence file is *called*, based on information found in a chromatogram, a *trace* file which is produced by the automatic sequencing machines. Phred is the most commonly used base calling software package (Ewing et al. 1998), and the two most commonly used programs for assembling the Sanger sequence data are Phrap and CAP3. Both programs make use of Phred generated quality scores when performing the assembly, although this data can be omitted if it is not available. Aside from being used to assemble data generated in large scale genome sequencing projects, these programs have also been used to assemble EST sequence data. Both Phrap and CAP3 use variations of a Myers-like approach to fragment assembly, though Phrap deviates from this standard template in the final consensus phase.

2.3.4.1 Phrap

Phrap stands for “phragment assembly program” or “Phil’s revised assembly program” and is used for assembling shotgun DNA sequence data. Unlike many other assemblers, Phrap makes use of the whole read and not just the trimmed high quality portion. Phrap can be provided with a mixture of machine generated and

user supplied quality data to assist in making reliable contigs. One aspect which sets Phrap apart from many other Myers type algorithms is that Phrap returns contig sequences which are mosaics of the highest quality parts of reads rather than a consensus or majority vote.

Phrap searches for reads with matching words and then does a SWAT comparison between pairs of reads with this property. This allows Phrap to efficiently make use of the very accurate Smith–Waterman algorithm encoded in the SWAT algorithm. This first stage identifies all potential overlaps. The next stage effectively masks vector sequences. This stage also identifies near duplicate reads, reads with self matches, and reads which have more than one significant overlap in any given region. These steps help Phrap to deal with repetitive elements. Phrap then constructs contig layouts based on strong pairwise alignments using a greedy algorithm and from these layouts produces the contigs. Finally, Phrap aligns reads to the contigs identifying inconsistencies and possible misassembled sites. Phrap returns Phred-like quality scores for each base in the contig based on the consistency of the pairwise alignments at that position (<http://www.phrap.org/phredphrap/phrap.html>).

2.3.4.2 CAP3

CAP3 is the third generation of the CAP assembly algorithm and was released in 1999 (Huang and Madan 1999). CAP3 uses a Myers-like method which makes extensive use of quality values and paired read data. The overlap stage begins by using a BLAST-like algorithm to identify areas where detailed local alignments are produced using a modified version of the Smith–Waterman algorithm which weights the substitution matrix at each position using the quality scores at the bases concerned. Where CAP3 differs from other algorithms is in the way that these overlaps are then validated. First, CAP3 identifies *good* regions in a read. A good region is a run of nucleotide bases with high quality scores and which share an overlap with a region in another read which also has high quality scores. CAP3 uses the good regions to identify which bases to trim from the ends of the reads. Once the good regions have been identified, CAP3 produces a global alignment of the reads previously identified as having local alignments and attempts to identify inconsistencies in the global alignments between good regions. There are a number of criteria each overlap must satisfy and any overlaps which do not meet all the criteria are discarded. This completes the overlap stage. CAP3 then uses a greedy algorithm to produce a layout of the reads which is validated by checking whether the paired read data (if supplied) produces any inconsistencies. Finally, the reads are aligned to the layout and a consensus produced. CAP3 also produces Phred-like quality scores which are returned to the user. In benchmarking, CAP3 generally produces better quality, shorter contigs than Phrap due to the strict methods for creating contigs (Huang and Madan 1999). However, CAP3 relies heavily on paired end data and even more so on quality values, and may not perform as well if given raw sequence data alone.

2.3.5 Short Read Fragment Assembly Algorithms

Several assemblers have been developed for short sequence reads, these include Edena, Velvet, EULER SR, SASSY and ALLPATHS. All of these algorithms borrow from the Myers or IWP models described above either implicitly or explicitly, and there are many similarities between the different algorithms in terms of their overall structure. Most algorithms are divided into up to five stages which include some or all of the following procedures: read error correction, read filtering, naïve assembly, refining of naïve assembly (using paired end data if available), and finishing. To understand what a naïve assembly is we need to define the terms *consecutive* and *linearly connected*. Two reads A and B are *consecutive* if they overlap (either the first k bases in A match the last k bases in B or vice versa) and for a graph with no transitive edges, two reads A and B are *linearly connected* if they are consecutive and there exists no read C which is consecutive with A on the same side as B or with B on the same side as A. For naïve assembly, we mean that starting with some read R we can try to extend that read (on one side) by examining the consecutive reads (on that side). If there is only one candidate, then the read can be extended in the direction of the overlap by the bases which are overhanging. This process mimics the collapsing of linearly connected sub graphs in the Myers model or edge formation in the IWP model. Thus, any string of linearly connected reads can be concatenated into one long read. For a given read there may be more than one candidate to extend with, and in this case the extension stops. Similarly, the extension stops when there are no candidates. The case where there is more than one candidate can be caused when the extension reaches the boundary of a duplicated or repetitive region in the genome or as happens much more frequently, it can be caused by errors in the data. If all possible extensions have been made for all available reads, then the resulting set of extended reads represents a naïve assembly. The accuracy of this assembly declines rapidly as both the error rate and the complexity of the organism being sequenced increase (Chaisson et al. 2004; Whiteford et al. 2005).

2.3.5.1 Edena

Edena, released in 2008 (Hernandez et al. 2008), is the first short read assembly algorithm to be released which uses the traditional overlap-layout-consensus approach. Edena does not include an error correction phase before graph production, which leads to the formation of a messy sequence graph; however it does include a three step error correction phase which cleans the graph before assembly begins. The first phase of the algorithm removes duplicate reads, keeping only the original read and the number of times it has been seen. Next it uses the reads to construct an overlap graph where the reads are represented as nodes. Two nodes are joined by an edge if there is an overlap between them larger than a set minimum (defined by the user). Once this graph has been built, it contains many erroneous edges which have to be removed. First it removes transitive edges in the graph in the same manner as described by Myers (Myers 1995). Following this, all dead end paths are removed.



Fig. 2.7 An example of a P-bubble most likely caused by an error. The reads making up the lower sequence will typically have a low copy number and the overlaps are very short. However, this phenomenon can also be caused by low copy number repeats

A dead end path is a series of consecutive nodes shorter than 10 reads in length which is attached to the main body of the graph on one side and to nothing on the other. These paths are caused when areas are sequenced with very low coverage, causing breaks in the sequence of consecutive reads, or when a series of errors combine to make a series of consecutive reads. Finally, the algorithm removes what are called P-bubbles. These occur when there are two regions which are identical except for a one base difference. In the case where this is caused by single nucleotide polymorphisms (SNPs) in repetitive regions, we would expect each side of the bubble to have a similar topology and copy number. Where a P-bubble is caused by an error, we would expect to see one side of the bubble with a very sparse topology and significantly lower copy number. Figure 2.7 gives such an example.

When P-bubbles are found, Edena removes the side with the lowest copy number/sparsest topology. Hernandez points out that P-bubbles may be caused by clonal polymorphisms which would account for the low coverage and sparse topology observed (Hernandez et al. 2008). However, as Edena does not take paired end information into account, the method used for eliminating P-bubbles will most certainly cause over-collapsing of low copy-number repetitive regions. Once the graph has been cleaned using the three operations described above, a naïve assembly is formed and the resulting contigs are returned to the user.

2.3.5.2 Velvet

Velvet is the name given to the collection of algorithms that assemble short read data which were released by Zerbino in 2008 (Zerbino and Birney 2008). Velvet uses an IWP model to make the initial graph. Like Edena, Velvet does not include an initial error correction phase but instead uses a series of error correction algorithms to clean up the resulting graph. These algorithms work in a method analogous to the error correction phase in Edena (Hernandez et al. 2008), where tips are removed and then bubbles. In Velvet, tips are removed only if they are shorter than $2k$, where k is the read length. Unlike Edena, Velvet uses an Eulerian path approach, which although highly efficient in terms of memory use, appears to further complicate the P-bubble removal step. Velvet includes an algorithm called Tour Bus which traverses the graph looking for P-bubbles, and when they are found, uses a combination of copy number and topographical information to remove the erroneous edges. Velvet then assumes that all low copy number edges

that remain must be errors and removes them from the graph. Like many of the algorithms described here, Velvet does not make use of paired read information and therefore has an increased probability of over-collapsing repetitive regions.

2.3.5.3 Euler SR

There have been many iterations of the original EULER algorithm developed by Pevzner in 2001 (Pevzner 2001). The latest addition to the EULER family is EULER SR which is a version of EULER optimized to handle short reads (Chaisson and Pevzner 2008). The algorithm described by Idury and Waterman did not include a step for filtering or correcting errors, however it did include a number of graph simplifications which could be used to reduce the impact of errors. Unfortunately, this method could not scale up to handle the large amounts of error present in real data. The original EULER algorithm was designed as an implementation of the Idury and Waterman algorithm, but included a novel method for error correction. A short read is broken down into a number of even shorter k -tuples which are stored in a database. In the case when the dataset contains no errors then we would expect that the k -tuples generated for a particular read R would appear a number of times in the database, as all reads overlapping with R would also contain some number of these k -tuples. Pevzner describes a read as “solid” if all of its k -tuples appear at least n times (where n is set by the user) or “weak” otherwise. When used with real data, if a read has been classified as weak, the algorithm tries to find the minimum number of base changes which will change its classification to strong. If that number is less than d (where d has been set by the user) then the changes are made, otherwise the read is discarded. Pevzner shows that this method corrects over 86% of errors with very few false positives for the dataset he analyzed (Pevzner 2001). This represents the most sophisticated and efficient approach for error correction of short reads that has been developed thus far. EULER SR builds on the original EULER algorithm and contains optimizations to make it more memory efficient, a property which is necessary for the vast amount of data produced by short read assemblers. Interestingly, in testing EULER SR, a hybrid approach was assessed where short read data was combined with longer Roche 454 read data. It was found that there was no significant improvement in assembly for the majority of reads (Pevzner 2001), which is contrary to most of the current opinion in this field. After the errors have been removed, a graph is built and a set of contigs produced by naïve assembly is returned to the user.

2.3.5.4 SASSY

We are currently developing an assembly algorithm called SASSY which is based on a Myers like method that incorporates paired end data. SASSY is being developed primarily to assemble eukaryotic sequences of around 100–200 Kbp in length cloned into BACs. While SASSY shares many similarities with the software described

above, there are a number of key differences. We have developed a novel iterative approach to graph construction which removes the need for some of the simplification steps typically needed for this type of implementation. We aggressively filter the data set, flagging up to approximately 90% of the reads which are set aside to be used only in later stages. With the remaining reads we construct a first round naïve assembly using only reads which have an overlap of at least $k - t$ nucleotide bases, where k is the read length and t is very low (usually 2 or 3). The advantage of using this approach is that erroneous areas of the graph usually have a sparse topology and the number of common bases between any two reads in these areas is usually much lower than for reads in correct areas of the graph. Thus, assemblies generated in the first round represent high confidence assemblies, however their length is typically very short, with an N50 of less than 50 bases for Applied Biosystems SOLiD reads and slightly longer for Illumina Solexa reads. It should be noted that the longest contigs produced from this preliminary assembly are typically 4,000–12,000 bases long. These longer contigs are used to identify *stable* areas in the overlap graph. The next stage involves building a new overlap graph which explicitly combines the overlap data in the original graph with the paired read data. Normally this would be difficult because of the repetitive nature of the data, but by starting the graph building in stable areas, many of the problems associated with repeats are resolved. Thus we use the naïve contigs only as a guide instead of trying to extend them, which is the case for the other algorithms described in this section. Following the construction of the overlapping graph, we align all the reads flagged in the filtering stage to the assembled contigs. We examine the distribution of the insert size for the mapped reads to identify erroneous assemblies which are repaired where possible or flagged as conspicuous in the case when there is no obvious resolution. Finally, new contigs are built from the new overlap graph and these are returned to the user. SASSY is being developed to make optimal use of local topology and paired end data in order to avoid the problems of over-collapsing repetitive regions or unnecessarily breaking contigs when errors are present in the data. This software is currently still in a developmental stage; however, initial test versions promise to overcome many of the limitations inherent in current small read assembly software.

2.3.5.5 ALLPATHS

ALLPATHS is another recent addition to the collection of short read assemblers based on an IWP model (Butler et al. 2008). ALLPATHS begins by correcting errors using an EULER like method, and then makes a large set of naïve assemblies which are referred to as “unipaths.” At this stage, ALLPATHS leaves the model followed by EULER, and uses paired read information to sort unipaths into localized groups which can be worked on individually and in parallel. For each localized set of unipaths, ALLPATHS chooses paired reads which lie in the set, and proceeds to work out every path of consecutive unireads which could possibly be followed from the read to its mate. Once this is completed, the number of paths is trimmed

down using localization information and other statistical methods until, ideally, only one path remains. This method reduces the complexity in the overall sequence graph by making local optimizations, allowing many shorter unipaths to be condensed into longer unipaths. Once the long unipath generation has been completed separately, the results from the local optimizations are stitched together to produce one long sequence graph. One limitation of this algorithm is its sensitivity to the standard deviation of the fragment length used to make the paired sequence reads. Butler notes that in some cases, a large number read-mate pairs generate over 10^3 possible paths, and in some cases more than 10^7 possible paths are generated, which causes ALLPATHS to return erroneous unipaths (Butler et al. 2008). The final phase incorporates both read pair information and statistics to identify erroneous assemblies, and if possible it tries to fix them. The most unique aspect of ALLPATHS is that no information is discarded at any stage in the algorithm which improves the ability to repair errors in the final phase. Unlike every other algorithm described here it returns the entire graph to the user as opposed to just the contigs.

2.4 Discussion

There are many branches of research into sequence comparison (more than have been covered here) with varying levels of complexity. The amount of effort being spent on solving different branches has continuously shifted as computational power has increased and the nature of the data being produced has changed. For example, multiple sequence alignment algorithms only started to receive widespread attention from the mid 1980s, almost 20 years after the merits of different phylogenetic tree making algorithms were being heavily debated, and many years after efficient algorithms had been produced for pairwise alignments. Pairwise sequence alignment has long been the base currency of sequence comparison, but graph theoretical methods; in particular k-mer distance methods and k-mer grouping/sorting have been demonstrated to be valuable for increasing the speed at which analysis can be performed. The typically long and accurate sequence reads produced using the Sanger sequencing method have been largely replaced (in terms of volume of data being produced) by next generation sequencing methods which produce copious amounts of largely error laden data, and the current focus of bioinformatics in this area has been to develop algorithms that can accurately assemble this data into long stretches of sequence. Again, graph theoretical approaches have proved valuable. Progress in the area of sequence assembly has only been feasible using computing power developed in recent years, although it should be noted that more than 40 years after the birth of comparative algorithms, the lack of ever greater computing power remains the main hindrance to progress. As an example, the original implementation of CLUSTAL was tested on a 10 MHz microcomputer (PC) with only 640K of memory, while the current iteration of the program SASSY was developed using an 8 core (1.8 GHz per core) cluster with access to 16 GB of memory and almost unlimited hard disk space. There is a clear

trend that advances in computing hardware continue to spur development of ever more sophisticated comparison algorithms, allowing researchers greater insight into comparative genomics and the workings of the biological world.

References

- Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. *J Mol Biol* 215:403–410
- Bentley DR (2006) Whole-genome re-sequencing. *Curr Opin Genet Dev* 16(6):545–552
- Burke J, Davison D, Hide W (1999) d2_cluster: A validated method for clustering EST and full-length cDNA sequences. *Genome Res* 9:1135–1142
- Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES et al (2008) ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Res* 18(5):810–820
- Chaisson MJ, Pevzner PA (2008) Short read fragment assembly of bacterial genomes. *Genome Res* 18:324–330
- Chaisson M, Pevzner PA, Tang HX (2004) Fragment assembly with short reads. *Bioinformatics* 20(13):2067–2074
- Dayhoff Mo, ed., 1978, Atlas of protein Sequence and Structure, Vol 5
- Dohm JC, Lottaz C, Borodina T, Himmelbauer H (2007) SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res* 17:1697–1706
- Edgar RC (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res* 32:1792–1797
- Ewing B, Hillier L, Wendl MC, Green P (1998) Base-calling of automated sequencer traces using phred. 1. accuracy assessment. *Genome Res* 8:175–185
- Feng DF, Doolittle RF (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol* 25:351–360
- Gotoh O (1982) An improved algorithm for matching biological sequences. *J Mol Biol* 162:705–708
- Hazelhurst S, Hide W, Liptak Z, Nogueira R, Starfield R (2008) An overview of the wcd EST clustering tool. *Bioinformatics* 24(13):1542–1546
- Henikoff S, Henikoff JG (1992) Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA* 89(22):10915–10919
- Hernandez D, Francois P, Farinelli L, Osteras M, Schrenzel J (2008) De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Res* 18(5):802–809
- Higgins DG, Sharp PM (1988) CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene* 73:237–244
- Higgins DG, Bleasby AJ, Fuchs R (1992) CLUSTAL V: improved software for multiple sequence alignment. *Bioinformatics* 8(2):189–191
- Huang X, Madan A (1999) CAP3: A DNA sequence assembly program. *Genome Res* 9:868–877
- Idury RM, Waterman MS (1995) A new algorithm for DNA sequence assembly. *J Comput Biol* 2:291–306
- Jeck WR, Reinhardt JA, Baltrus DA, Hickenbotham MT, Magrini V, Mardis ER et al (2007) Extending assembly of short DNA sequences to handle error. *Bioinformatics* 23:2942–2944
- Kent JW (2002) BLAT – the BLAST-like alignment tool. *Genome Res* 12:656–664
- Murata M, Richardson JS, Sussman JL (1985) Simultaneous comparison of three protein sequences. *Proc Natl Acad Sci USA* 82(10):3073–3077
- Myers EW (1995) Toward simplifying and accurately formulating fragment assembly. *J Comput Biol* 2:275–290

- Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* 48:443–453
- Notredame C, Higgins DG, Heringa J (2000) T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol* 302:205–217
- O'Connor M, Peifer M, Bender W (1989) Construction of large DNA segments in *Escherichia coli*. *Science* 244:1307–1312
- Penzner PA (2001) Fragment assembly with double-barreled data. *Bioinformatics* 17:S225–S233
- Pevzner PA (1989) l-tuple DNA sequencing: computer analysis. *J Biomol Struct Dyn* 7:63–73
- Pevzner PA, Tang HX, Waterman MS (2001) An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci USA* 98(17):9748–9753
- Sanger F, Nicklen S, Coulson AR (1977) DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci USA* 74(12):5463–5467
- Sellers PH (1974) On the theory and computation of evolutionary distances. *J Appl Math (siam)* 26:787–793
- Smit AFA, Hubley R, Green P RepeatMasker Open-3.0. 1996-2004. <http://www.repeatmasker.org>
- Staden R (1979) A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res* 6:2601–2610
- Thompson JD, Higgins DG, Gibson TJ, Clustal W (1994) Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*. Nov 11;22(22):4673–4680
- Warren RL, Sutton GG, Jones SJM, Holt RA (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics* 23(4):500–501
- Waterman MS, Smith TF, Beyer WA (1976) Some biological sequence metrics. *J Adv Math* 20:367–387
- Wheeler DA, Srinivasan M, Egholm M, Shen Y, Chen L, McGuire A et al (2008) The complete genome of an individual by massively parallel DNA sequencing. *Nature* 452(7189):U872–U875
- Whiteford N, Haslam N, Weber G, Prugel-Bennett A, Essex JW, Roach PL et al (2005) An analysis of the feasibility of short read sequencing. *Nucleic Acids Res* 33(19):e171
- Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res* 18(5):821–829

Bioinformatics

Tools and Applications

Edwards, D.; Stajich, J.; Hansen, D. (Eds.)

2009, XII, 451 p., Hardcover

ISBN: 978-0-387-92737-4