PHAST (PHAGE ASSEMBLY SUITE AND TUTORIAL):

A WEB-BASED GENOME ASSEMBLY TEACHING TOOL

D. Leland Taylor

Computational Biology

Center for Interdisciplinary Studies

Davidson College

May 6, 2012

Dr. Laurie Heyer

Dr. A. Malcolm Campbell

Dr. Scott Denham

Dr. David Wessner

Funding provided by the Davidson Research Initiative and the Mimms Summer Research Fellowship. Special thanks to those with the HHMI PHIRE program: Dr. Graham Hatfull, Dan Russell, Dr. Tuajuanda Jordan, Melvina Lewis, and Dr. Lucia Barker. Thank you to the Davidson College staff who helped with this project: Bill Hatfield and Paul Brantley. Thanks to Phillip Compeau for his advice on various genome assembly topics. Finally, special thanks to my advisors, Dr. A. Malcolm Campbell and Dr. Laurie Heyer, for their guidance throughout this process!

TABLE OF CONTENTS

1 ABSTRACT

Next generation sequencing technologies have greatly reduced the cost of sequencing genomes. With the current sequencing technology, a genome must be broken into sections and then sequenced, producing "reads." A computer pieces these reads back together in a process known as genome assembly. PHAST (Phage Assembly Suite and Tutorial) is an online set of modules designed to teach the genome assembly process. The website includes tutorials detailing the complexities of genome assembly and allows users to assemble small phage genomes of their own. With PHAST, entry-level biology students learn concepts that affect genome assembly (coverage, read lengths, read errors, etc.) and come to appreciate the use of mathematics and graph theory to solve biological problems such as the genome assembly problem.

2 INTRODUCTION

As defined by the National Institutes of Health (NIH), computational biology is the "development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral, and social systems" (Huerta *et al.*, 2000). Modern scientific tools, such as genome sequencing and microarray analysis, produce overwhelming amounts of data that make manipulating, storing, and interpreting such data difficult. To meet these challenges, the field of computational biology blends mathematics, biology, and computer science into a single area of study. Through computational biology, scientists develop the methods, tools, models, and skills required to facilitate scientific inquiry in a data saturated world.

Genome assembly is a sub-field of computational biology in which scientists focus on building tools and methods to solve the genome assembly problem. Modern DNA sequencers produce gigabytes of data per run. These data are the sequences of small DNA fragments called "reads." The genome assembly problem is to correctly stitch these gigabytes of reads together to form a genome. Computational biologists have approached the genome assembly problem in a variety of ways; however, the most successful approaches are based on graph theory. Mathematical methods developed to construct theoretical graphs, store graphs in computer memory, and efficiently traverse graphs are now frequently used to solve the problem of decrypting an organism's genome, thanks to computational biologists.

As a capstone to my undergraduate study in computational biology, I designed a website, called PHAST (Phage Assembly Suite and Tutorial), to teach undergraduates about

the genome assembly process. PHAST includes tutorials detailing complex issues in genome assembly and allows users to assemble small genomes of their own based on some choices they can make. Because PHAST describes many of the issues included in this thesis, some of the text in this document is very similar to the PHAST tutorials.

2.1 MOTIVATION

I was motivated to produce PHAST to fill a void in the national genomics education program called Phage Hunters Integrating Research and Education (PHIRE), funded by the Howard Hughes Medical Institute's (HHMI) Science Education Alliance (SEA). PHIRE engages young scientists in authentic research by identifying and isolating mycobacteriophages (called "phages") – viruses that infect *Mycobacterium smegmatis* (Phanning the Phlames, 2012). PHIRE is a yearlong curriculum that targets first year undergraduates and students who are often left behind in genomics curriculum innovations. In the first semester, students follow a series of lab protocols to isolate and characterize phages from local soil samples of their choosing. Over winter break, genomes of a few phages from each institution are sequenced at a sequencing facility. In the second semester, students receive a fully assembled genome, which they annotate and compare to other phage genomes (Mycobacteriophage Database, 2012). To date, the PHIRE program has been very successful – reaching students at 97 institutions (Mycobacteriophage Database, 2012), as well as producing several papers (Hatfull *et al.,* 2006; Pope *et al.,* 2011).

The depth in which PHIRE institutions explore genome sequencing and assembly varies. Some PHIRE institutions detail the next generation sequencing and assembly process, and even have their own sequencing facilities available to the institution's PHIRE program. In general, most institutions use video tutorials provided on the PHIRE website

that describe the overall sequencing workflow (Mycobacteriophage Database, 2012). These video tutorials are informative and useful, but primarily describe how to manually edit and validate a genome assembly. PHIRE offers little information on the process that produced the genome assembly or the complexities and frequent complications associated with the genome assembly problem.

PHAST builds on PHIRE's model of an interactive learning process and is a "one-stop" teaching tool where students learn about the genome assembly process in an interactive environment. To that end, PHAST is designed as a website to allow simple, cross-platform availability with no installation required on the user's computer. The only available alternative requires the installation of a virtual Linux machine and familiarity with UNIX, which is beyond the capacity of most biology faculty. At the PHAST web site, each user can assemble selected genomic data (raw output of high-throughput sequencers). After assembly, a user's genomes are available on every page of the site, along with basic statistics on each assembly. No other user has access to these genomes, so each student must do the work for him or herself. Once two or more genomes are assembled, a user can compare the structure of different assemblies using a fully integrated platform-and browser-independent dotplot comparison tool.

Phage genomes are ideal for PHAST, given the computational requirements of a web-based assembly tool. The genome assembly process is very time consuming and can take days to assemble a complicated, large, eukaryotic genome on a powerful, dedicated computer (Miller *et al.*, 2008). Because PHAST is web-based, it needs to be able to perform several assemblies at once, which is not feasible with complex genomes, even on a super computer. However, phage genomes are small, ranging from 41,441 to 164,602 base pairs,

and have little genomic complexity, such as repetitive sections of DNA (Mycobacteriophage Database, 2012). These properties make it feasible to assemble multiple phage genomes at once on a standard web server. Using a MacPro3 computer with two 2.8 GHz Quad-Core Intel Xeon processors and 4 GB 800 MHz of DDR2 RAM, most phages in the PHAST database take no more than several hours to completely assemble. For these longer assemblies, PHAST has an email feature that will email a user when an assembly is finished.

3 GENOME SEQUENCING TECHNOLOGY

The term "genome" refers to all genetic material in viruses, prokaryotic cells, and eukaryotic organelles. In haploid organisms, the term refers to the genetic material in a haploid set of the organism's chromosomes (Russell, 2010). Sequencing is the process of reading or decoding the nucleotides of the DNA (or RNA) composing a genome. The genome sequencing process can be divided into three general steps (1) sequencing, (2) finishing, and (3) annotating (Campbell and Heyer, 2007). In the sequencing process, genomic DNA is isolated from the target organism, read by a sequencer, and converted into digital information that can be stored on a computer. Because current sequencing technologies read at most a few thousand contiguous base pairs and the smallest known genome of a cellular organism is 159,662 base pairs (Nakabachi *et al.*, 2006), a genome must be broken up and sequenced in small segments (even a small Mycobacteriophage is roughly 40,000 base pairs). The resulting digital information of sequenced nucleotides are called reads. Through computer programs, these reads are assembled into a genome.

The goal of a genome-sequencing project is to obtain a digital copy of the target organism's genomic DNA in the exact order it appears inside the target genome. Genomes that contain at most 1 error per 10,000 bases are considered finished genomes (Campbell and Heyer, 2007). Due to inherent sequencing errors and the complexity of the target genome, the results of a first sequencing run are rarely this similar to the actual target genome – especially for eukaryotic organisms. Such genomes are described as draft genomes and inevitably have DNA inversions, insertions or deletions (indels), and incorrect base determination when compared to the target organism's actual genome.

Fixing these sequencing errors is very expensive, requiring many hours of work by highly trained technicians. Because of the cost, most genomes are never finished and remain in a draft form.

Genome annotation is the final stage of a genome-sequencing project. Annotation involves identifying functional segments of DNA in the genome. Often, these segments are genes, but they may also be intergenic regions that perform some other function such as gene regulation (Campbell and Heyer, 2007).

Until 2005, most DNA sequencing reads were generated by Sanger technology, published by Dr. Fred Sanger in 1977 (Perkel, 2009). Sanger sequencing, also called dideoxy sequencing, utilizes dideoxynucleotides (ddNTP's) that have a hydrogen molecule on the 3' carbon instead of a hydroxyl group (Figure 1). The absence of a 3' hydroxyl group prevents the addition of more nucleotides on a DNA strand (Wilton, 2002). Adding fluorescently or radioactively labeled ddNTP's to a DNA synthesis reaction using the target DNA as template DNA will produce a pool of DNA molecules in which some molecules terminate at every position along the target DNA (Perkel, 2009). The target DNA sequence is determined by separating the pool of DNA molecules by size and reading the labeled ddNTP at each position. Originally, this process was accomplished using slab polyacrylamide gel electrophoresis; however, modern machines use capillary gel electrophoresis (Kim *et al.,* 2008). Sanger sequencing is very accurate but expensive and requires a significant amount of human oversight, as DNA must be cloned and maintained in cell libraries (Perkel, 2009). The high cost and slow pace of Sanger sequencing made it unsuitable for sequencing entire genomes.

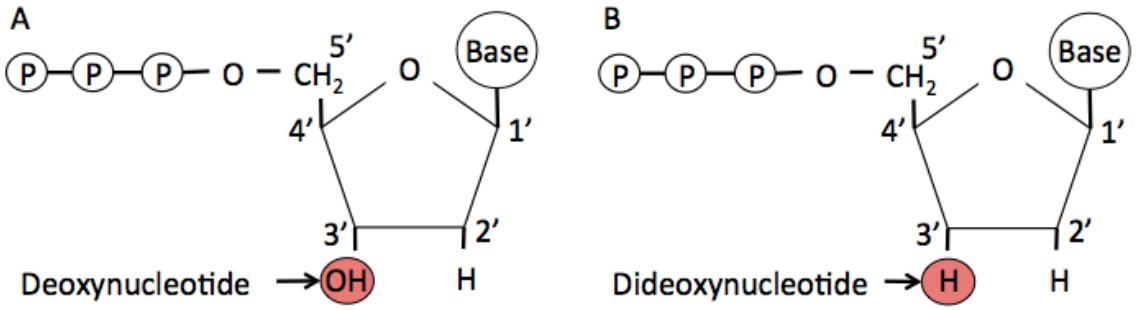

Figure 1. Nucleotide molecules. 1A: A regular deoxynucelotide (dNTP) molecule. DNA can elongate. 1B: A dideoxynucleotide (ddNTP) molecule. DNA elongation halts.

Over the last few years, interdisciplinary teams of investigators have developed new sequencing technologies, dubbed "next generation sequencing technologies", or next-gen technologies. These next-gen technologies are much faster and less expensive than Sanger sequencing through massive parallel processing, by which millions of DNA fragments are immobilized and sequenced simultaneously (Pop, 2009). However, next-gen technologies produce shorter read lengths, are less accurate for an individual read, and have other complications specific to the sequencer that are not present in Sanger sequencing. Fortunately, these problems can be solved. The high speed and low cost of these technologies make it possible to sequence each nucleotide many times, which improves the overall accuracy through redundancy. In addition, scientists have developed assembly algorithms optimized for use with short read lengths. Ultimately, the increased capacity and low cost of second-generation methods make them ideal for whole genome sequencing projects.

As of 2012, the DNA sequencer market is filled with various platforms, each sequencing DNA in a different manner. Sequencers such as 454 (DNA sequencing - the 454 method, 2011), Illumina (DNA sequencing - the Illumina method, 2011), and Helicos (tSMS How It Works, 2008), use a DNA polymerase-based approach, where DNA polymerase

activity is systematically halted and measured (Mardis, 2008). The Helicos platform can almost be described as "Sanger sequencing 2.0," since it uses fluorescently labeled, reversible terminators to pause, record, and re-initiate base by base sequencing (Perkel, 2009). The significant difference is that rather than analyzing populations of molecules, Helicos evaluates single molecules. Life Technologies' Ion Torrent (Ion Torrent technology, 2012) platform measures the release of hydrogen molecules as DNA polymerase incorporates nucleotides, and Life Technologies' SOLiD (Sequencing by Oligo Ligation and Detection; 5500 Series SOLiD Sequencers, 2011) platform employs DNA ligase to sequence through the hybridization and ligation of known, fluorescently labeled 8-mer oligonucleotides to template DNA (Mardis, 2008). All of these technologies generate reads that range from 50 base pairs up to 1,000 base pairs in length. The goal in all technologies, though, is to produce longer reads.

Other companies have taken a different approach by sacrificing massive parallel processing for more accurate, ultra long reads (thousands of base pairs long). Pacific Biosciences' sequencer (Introduction to SMRT Sequencing, 2010) measures DNA polymerase activity and records fluorescently labeled bases as they are incorporated in real time. ZS Genetics' platform (Reading and Analysis, 2012) uses an entirely different approach by directly reading a DNA sequence through transmission electron microscopy (Perkel, 2009). Finally, Oxford Nanopore Technologies uses nanopores to directly sequence DNA through electronic analysis of one base at a time as DNA polymers are forced through a tiny channel (Nanopore Sensing, 2012). In short, DNA sequencing technology is rapidly improving and expanding. Some scientists are refining older technologies, while others are developing completely novel technologies.

The next-gen technology most relevant to PHAST is the pyrosequencing technology used by Roche's 454 sequencers, because 454 sequencers produced the majority of Phage Hunter phage genomes. Pyrosequencing (Principle of Pyrosequencing Technology, 2012) gets its name from the pyrophosphate molecule (Figure 2). Pyrophosphate is a natural byproduct produced when DNA polymerase adds a nucleotide to a growing DNA strand. By enzymatically converting pyrophosphate to ATP, which powers light production through luciferase, pyrosequencing generates bursts of light for each base added (Margulies *et al.,* 2005).

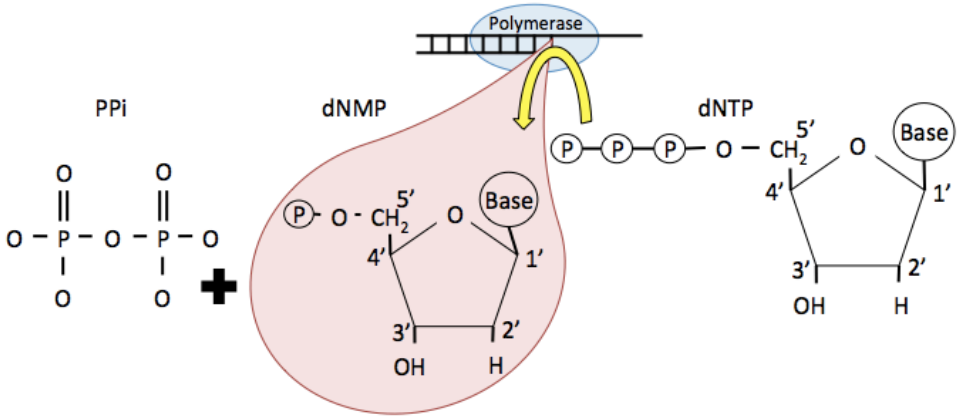

Figure 2. DNA elongation and the pyrophosphate molecule. When a deoxynucelotide triphosphate (dNTP) is incorporated, its molecular structure changes to a deoxynucelotide monophosphate (dNMP) and releases a pyrophosphate (PPi) molecule.

Pyrosequencing begins by breaking the template DNA into fragments and attaching common DNA adaptors to each end of all of the fragments (Figure 3A). These adaptors correspond to millions of oligomers attached to the surface of hundreds of thousands of agarose beads (Mardis, 2008). The DNA fragments are ligated to agarose beads such that most beads house a single DNA fragment. Next in emulsion PCR (emPCR), single beads are captured in water droplets (Figure 3B) that are suspended in synthetic oil and act as PCR

micro reactors. After emPCR amplification, each bead contains millions of identical DNA fragments (Figure 3C).

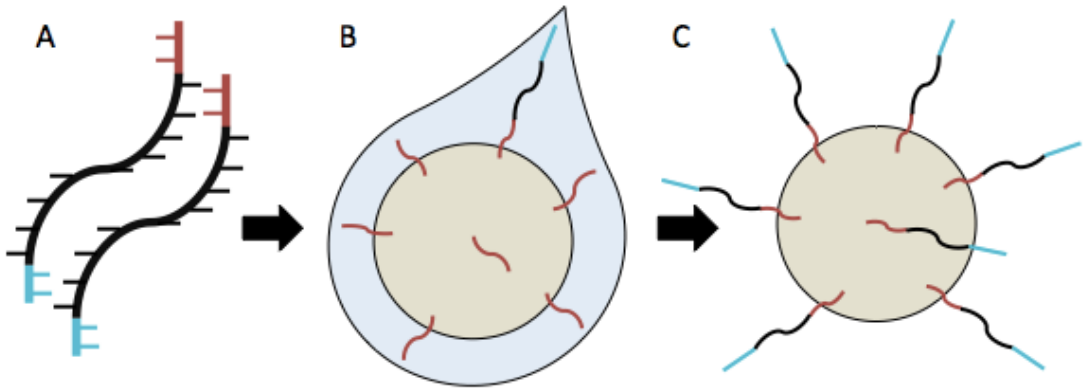

Figure 3. The first steps of 454 pyrosequencing. 3A: Adaptors (red and blue) are attached to DNA fragments. 3B: A DNA fragment is ligated to an agarose bead with oligomers corresponding to the adaptor sequence. These fragments are captured in water droplets. 3C: The DNA is amplified, the emulsion is broken, and the clonal DNA fragments are denatured.

The DNA-coated beads are loaded into the wells of a PicoTiterPlate (PTP) that are attached to fiber optic strands (Perkel, 2009). The PTP is placed in the sequencer and acts as a flow cell for the pyrosequencing light reaction. The pyrosequencing method sequences the template DNA by measuring the light intensity produced by a series of reactions whenever a nucleotide is added to the complementary DNA strand. Each nucleotide is washed over the PTP in a closely regulated series of steps. When a nucleotide is incorporated into the DNA strand attached to a particular bead, a camera reads the flash of light. This light flash is proportional to the number of bases added, so that the light intensity from two bases added in a row is twice as intense as the light from a single base added (Perkel, 2009). The light information is captured in recorded into a flowgram, which is a data structure that stores the light intensities of each base as it flows over a well of the PTP plate (Figure 4; Margulies *et al.,* 2005). Using this system, the most recent 454 GS FLX+

platform is capable of producing quality reads up to 1,000 base pairs in length. Because 454 uses adaptors, cleaning reads of any vestiges of adaptor sequences is a crucial step in the sequencing process using 454 data.

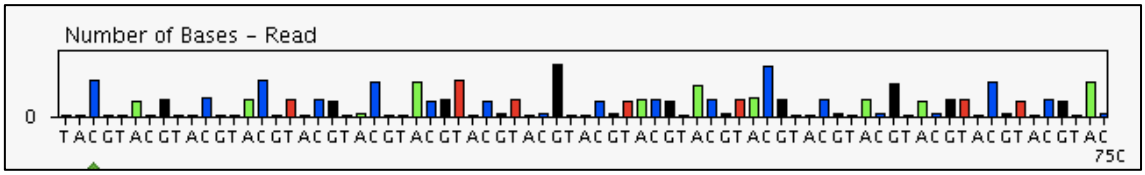

Figure 4. A 454 flowgram of a read from the timshel phage. *Screenshot taken from the GS De Novo Assembler (Newbler).*

4 FUNDAMENTALS OF GENOME SEQUENCING

There are three strategies used to sequence entire genomes: (1) whole genome shotgun, (2) hierarchal genome shotgun, and (3) pooled genome indexing (Kim *et al.*, 2008). Hierarchal genome shotgun is very time consuming and expensive because it involves mapping the genome using bacterial or yeast vector libraries (Campbell and Heyer, 2007). Pooled genome indexing is a hybrid approach that combines hierarchal genome shotgun and whole genome shotgun. Of the three, the most common and cost effective method is whole genome shotgun (WGS) sequencing. In WGS, multiple copies of genomic DNA are broken into millions of random fragments. These fragments are sequenced individually and their DNA sequences are electronically "stitched" back together into the genome through a process called assembly. Assembly is extremely complicated because there are missing DNA segments (gaps), errors in nucleotide identification, and long segments of repeated nucleotides, making it difficult to assemble the final genome.

As an analogy for WGS sequencing, imagine three copies of the same novel, printed in a foreign language. Because most novels are at least 40,000 words, which corresponds nicely to the number of base pairs in a small phage genome (Nebula Awards, 2009), each word in the novel represents a nucleotide in the genome. Every page of each of the three novels has been randomly cut into thousands of horizontal strips, and some of the strips are missing. Furthermore, suppose each of the three copies of the novel has random typos throughout, in different places in each copy. The assembly task is to arrange the thousands of strips of paper from all three novels to assemble a single copy of the original book (Figure 5).

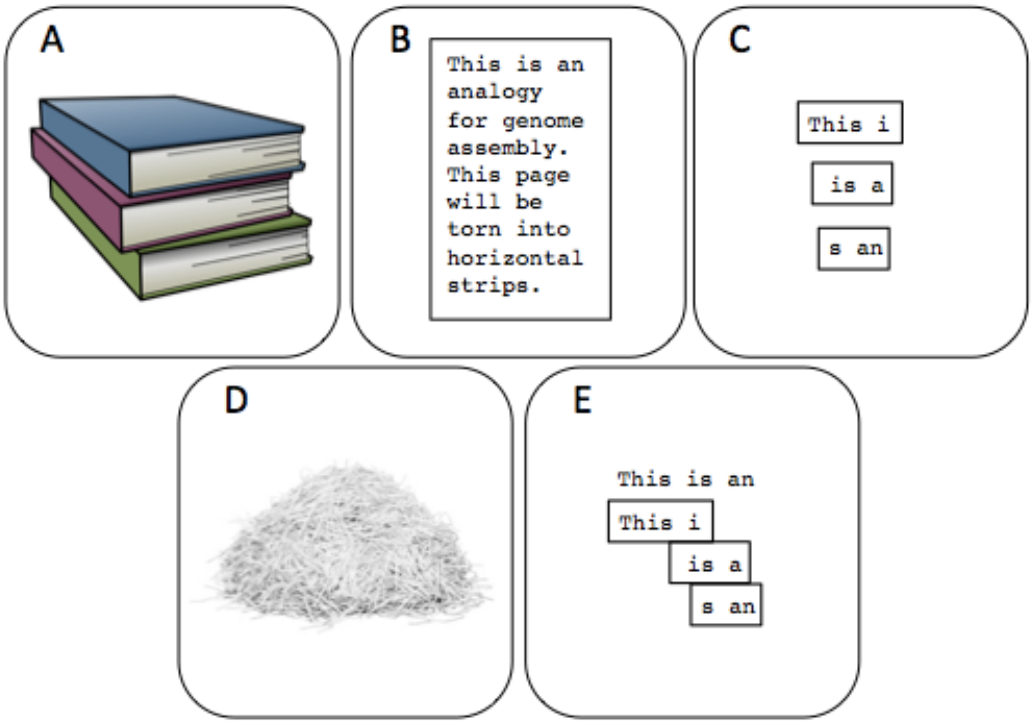

Figure 5. Genome assembly novel analogy. 5A: Three copies of the same novel. 5B: An example of one page out of the novel. All pages will be randomly cut into strips of characters. Note that there are random typos and errors throughout each novel. 5C: A few strips of characters from one page. 5D: All of the strips of characters from the 3 novels. 5E: Every single strip from 5D must be assembled as shown here to create a single copy of the novel. Note that some of the strips are also missing, further complicating this process.

Critical to the genome assembly process is genome coverage. Genome coverage refers to the ratio between the cumulative size, in nucleotides, of a set of reads and the size of the genome. For example, a dataset with 1 million reads of 100 nucleotides each that perfectly samples a 4 million base pair genome has 25-fold coverage (100 million nt sequenced/4 million bp in the genome = 25x coverage).

No sequencing technology is perfect and each sequencing technology has different error rates. Technologies can misread a nucleotide or skip a nucleotide altogether. However, with high coverage, a computer or person can more accurately deduce the correct genome sequence based on the consensus of the majority of the smaller reads (Figure 6). Because each nucleotide of each read in an overlap effectively casts a single vote for the final sequence, the final, assembled genome is often referred to as the "consensus sequence."

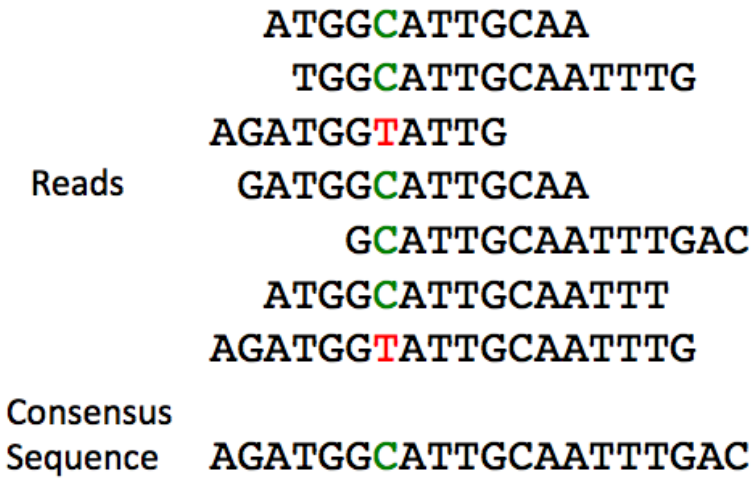

Figure 6: The consensus sequence is determined by the majority of reads. The green C appears in the majority of reads, suggesting the red T is an error. Therefore, the consensus sequence contains a C instead of a T.

Greater genome coverage produces a better genome assembly (Figure 7) just as having more books torn into strips would make it easier to assemble the original novel. High coverage gives the assembly algorithm the ability to identify raw reads that overlap with each other. Using the book analogy again, 25-fold coverage would be like having 25 novels cut into strips instead of only 3 novels. With more novels, the likelihood of identifying strips of paper that overlap, fill in gaps, correct for typos, and cover the entire

text increases. In other words, with greater genome coverage, assembly algorithms are able to do a better job of reconstructing the genome.

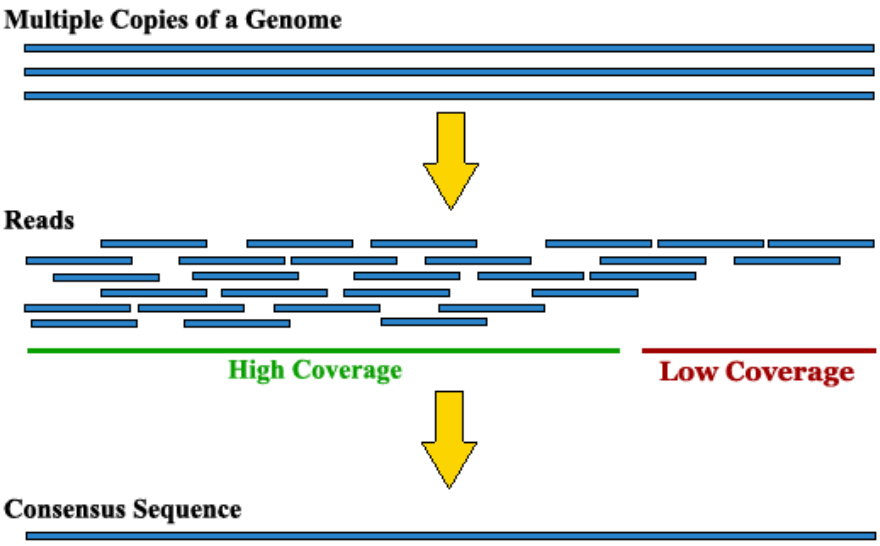

Figure 7. Genome coverage and the assembly process. Multiple copies of a genome are randomly broken into small fragments. Pieces of these DNA fragments are sequenced, generating reads. Reads are combined in areas they overlap. Portions of the genome in which many reads overlap are said to have high coverage (green bar). Portions in which few reads overlap are said to have low coverage (red bar). The majority of the reads form the final consensus sequence (see Figure 6). The higher the coverage of a consensus sequence segment, the more confidant you can be in the accuracy of that segment.

Most genome assembly software packages combine several steps in the assembly process surrounding the actual assembly step. In general, this workflow follows (1) trimming vector sequences, (2) fragment assembly, (3) assembly validation, and (4) scaffold generation (Kim *et al.*, 2008).

It is extremely important to trim any vestiges of vector sequences that may have been introduced during the sequencing process. For example, during the 454 sequencing process, the DNA fragments carry short DNA sequence adaptors. These adaptor sequences are often included within the raw read output and must be trimmed off electronically

(Chevreux, 2010). Additionally, labs sequencing multiple genomes use DNA tags, called Multiplex Identifier (MID) tags, to track each individual genome by identifying a specific genome in the 454 workflow (Chevreux, 2010). Each project uses a unique MID tag and these sequence tags are part of the raw read output. In both cases, these DNA tag sequences complicate the assembly because they produce false overlapping segments of DNA that can lead to erroneous contig formation.

Fragment assembly involves the actual assembly of the raw shotgun fragment data into contigs. Contigs are contiguous sequences of DNA based on overlapping sections of DNA (Figure 8A). In the book analogy, a contig would be a page stitched together based on areas where the random paper strips overlapped (Figure 8B). Longer reads make it easier to assemble the consensus contig. Growth of the contig continues as long as quality overlaps exist between raw reads.

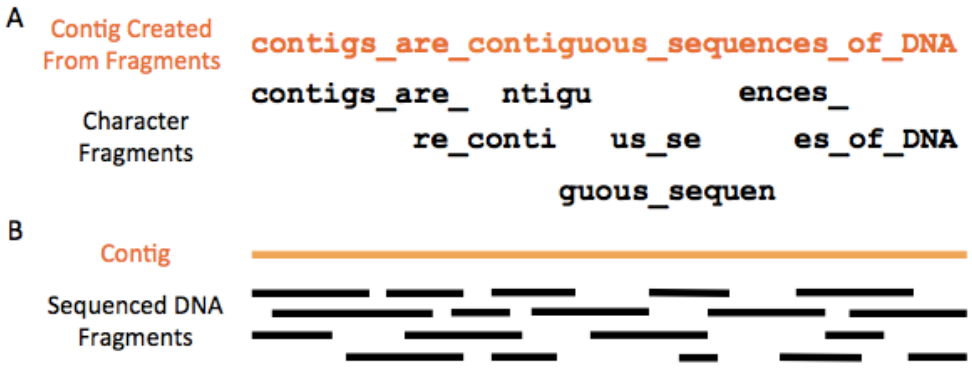

Figure 8. Contig formation. 8A: Creating a contig (orange) from line fragments (black). 8B: Creating a contig (orange) from genome reads (black). Bars represent DNA sequences.

Before proceeding to the scaffolding process, contigs should be verified. Contigs may be incorrect due to the repeat structure of the target genome. Repeats may occur many times within a genome, possibly as a result of transposable elements, genetic duplications,

or prophages (Wetzel *et al.*, 2011). Repeat length varies from short tandem repeats, such as AAAAAAAA or ACACACACA, to long DNA stretches thousands of base pairs (Wetzel *et al.*, 2011). The best way to verify contigs is through wet-lab experiments and various mapping techniques; however, the cost and time of such experiments make them impractical for many sequencing projects. Instead, statistical analyses are often used to identify misassembled regions. For example, the Celera Whole Genome Assembler uses an A-statistic to compute the probability of the distribution of read fragment start points in a contig. If the A-statistic is greater than 10, the contig is deemed "correct" (Kim *et al.*, 2008).

Small, simple genomes that lack large repetitive DNA sections can often be assembled using only the contig formation process. For larger, more complex genomes, an additional step called scaffolding is needed. Scaffolds are generated by assembling contigs, ordered (first to last) and oriented (facing left or right) with respect to one another and the physical genome. In order to generate scaffolds, additional information on the target genome is needed. Most crucial is mate-pair information[1], where segments of DNA of a known size are sequenced on both 5' and 3' ends (Mate Pair Sequencing, 2012). Because the physical genomic distance between these sequences tags is known, these data can be used to distance contigs that contain each tag. For example, if a 2 kb fragment of a genome were sequenced 100 bp on each end, then these reads and the contigs containing them

[1] Mate-pair information is sometimes called paired-end information; however, in reality the two refer to different protocols. They produce similar information – the distance between two sequences. Mate-pair insert size ranges from 2 – 5 kb (Mate Pair Sequencing, 2012), and is generally used for scaffolding. Illumina makes mate-pairs by taking a large DNA fragment, cutting the middle out, ligating the ends together, and sequencing the ends of the large fragment with the paired-end protocol. Paired-end insert size is much smaller, generally 200 – 500 bp in size (Paired-End Sequencing, 2012). Illumina refers to paired-end as the protocol for sequencing both ends of an actual DNA fragment (without cutting out the middle).

should be roughly 2 kb apart (the actual mate pair protocol is slightly more complex than described). Some scaffolding algorithms, such as GigAssembler, make use of other information such as physical genome maps, genetic genome maps, and expressed sequence tags (EST) in addition to mate-pairs (Kim *et al.*, 2008). All this additional information is used to order and orient contigs in an attempt to piece together scaffolds. In many assembly projects, a finished genome is the result of filling in gaps between scaffolds.

5 GENOME ASSEMBLY METHODS

There are two basic fragment assembly approaches: a reference-based approach, and a *de novo* approach. In a reference-based, or comparative, assembly, the raw reads of the genome being assembled are compared to an established reference genome sequence as an assembly guide. Reference-based assembly is fast and uses less computational power than *de novo* assembly. However, reference genome assembly may introduce a bias and is only appropriate if a good reference genome is available. A reference genome should be the genome of a closely related species, such as a different strain of the same bacteria (Pop, 2009). In many cases a reference genome is not known at the time of assembly.

When an appropriate reference genome is not available, scientists use the *de novo* (*i.e.,* done from scratch) assembly process. *De novo* assembly takes longer and uses more computational resources, but is the only option for genomes with no suitable reference genome. Within the category of *de novo* assembly, investigators use one of three possible methods: (1) the greedy method, (2) the overlap layout consensus (OLC) method, and (3) de Bruijn graph method. The greedy method joins a sequence read with another read that has the best overlap score until no more reads can be joined. The OLC method generates a directed graph using reads and overlaps. The nodes (circles) of the graph are reads, and the edges (arrows) represent an overlap of variable length between the suffix of the source node (arrow tail) and the prefix of the destination node (arrow head). In this way, the assembly process becomes synonymous with finding a path through the graph that visits every node exactly once (Figure 9B; edges not labeled, overlap of at least 2 nt). Finally, the de Bruijn method constructs a graph similar to the OLC graph. Roughly speaking, in a de

Bruijn assembly graph the edges are unique subsequences of length *k* within the reads, and the nodes represent common *k* – 1 subsequences (section 5.3). Thus, the assembly algorithm becomes finding a path in the graph that visits every edge at least once (each node may be visited more than once; Figure 9C).

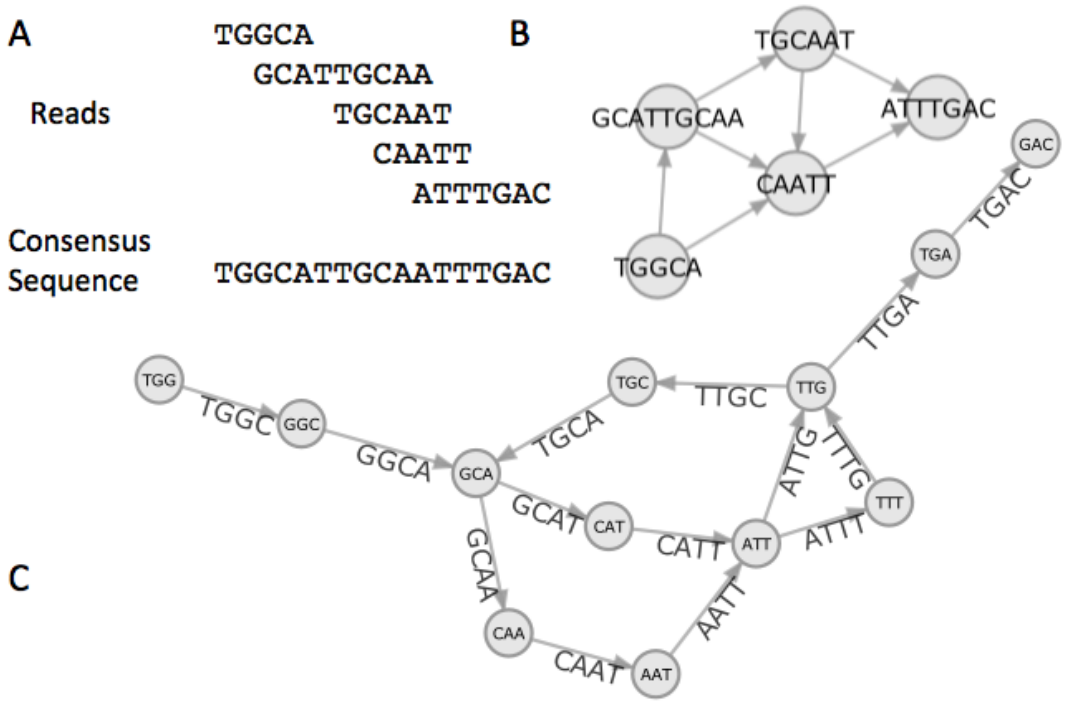

Figure 9. Reads and two possible assembly graphs. 9A: Hypothetical reads aligned to the consensus sequence. 9B: An OLC assembly graph created from these reads. Edges represent overlaps of 2 or more nt. The assembly process is to visit every node. 9C: A simplified de Bruijn assembly graph created from these reads. Edges represent 4 nt segments with 2 nt overlap between the nodes. The assembly process is to visit every edge.

In both the OLC and de Bruijn assembly methods, it might be possible to traverse the graph in more than one way, representing different genome arrangements. Although *de novo* assemblies use one of the three methods described above, different investigators have implemented their method of choice with slight variations. These variations can produce

different outcomes, so that even though two assembly programs may both use an OLC approach, for example, their output could be quite different.

5.1 Greedy Assembly Algorithms

The first *de novo* assembly algorithms were greedy assembly algorithms. Greedy assembly algorithms begin by calculating pairwise alignments of all read fragments. These alignments are scored, usually based on the length of the overlap and the percentage of matching bases (Pop, 2009). The two reads with the highest scoring overlap are merged together, and the resulting "contig" is added to the pool of sequences (Narzisi and Mishra, 2011). The operation of extending a contig continues until no more quality overlaps exist (Miller *et al.*, 2010).

Greedy assembly algorithms are called "greedy" because they optimize a local objective function - the quality of overlaps between reads (Pop, 2009). Inherent to this approach is a tendency to get stuck at local maxima, instead of identifying a globally optimal solution (Miller *et al.*, 2010). For example, because greedy assemblers process reads based on the highest scoring overlap, they may incorporate a read into one contig that should have been incorporated into another, especially when dealing with repeats (Figure 10; Pop, 2009).

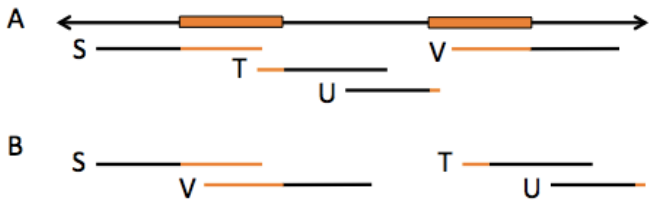

Figure 10. An incorrect greedy fragment assembly of reads. 10A: Correct assembly of reads. Orange segments are repeats. 10B: Assembly of reads using greedy fragment assembly.

Early greedy assemblers, such as TIGR and CAP3, were optimized for Sanger data and followed the algorithm previously described. More recent greedy assemblers, such as SSAKE, SHARCGS, and VCAKE, are optimized for short-reads and vary slightly from the earlier greedy algorithms. These short-read greedy assemblers begin by selecting an unassembled read as a seed for contig formation. This contig is extended from the 3' end and then from the 5' end of the reverse complement contig sequence (Pop, 2009). The extension process continues as long as the extension reads, that overlap the contig sequence, lack many sequence differences (Figure 11). In general, these stringent extension requirements avoid misassembles, but produce small contigs, only a few kilobases in length (Pop and Salzberg, 2008).

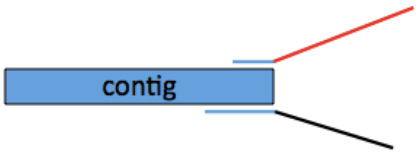

Figure 11. A repeat boundary. The black/blue and red/blue lines represent two reads. These reads overlap with the contig (blue sections), but do not overlap beyond the contig (red and black sections). The contig cannot be extended because the reads that would extend the contig do not agree.

5.2 Overlap Layout Consensus (OLC) Assembly Algorithms

Overlap layout consensus (OLC) is a popular method used for *de novo* genome assemblies. OLC requires three steps: (1) overlap, (2) layout, and (3) consensus. The overlap stage computes and builds the basic assembly graph. The layout stage compresses the graph, and the consensus stage determines the genome sequence based on the graph generated in the previous two steps.

5.2.1 Overlap

A basic, simplistic overlap algorithm compares the sequence of each read to that of every other read, in both the forward and reverse complement orientations. Using such a method makes overlap computation a very time intensive step – especially if the set of reads is very large. For example, the whole genome shotgun assembly of Drosophila had about 3 x 10^6 reads of 500 bases, requiring roughly 10^{13} pairwise comparisons (Deonier *et al.*, 2010). Even on today's computers, running that many comparisons is impractical, so scientists developed ways to decrease the number of comparisons by using seeded algorithms, like those behind NCBI's BLAST (Basic Local Alignment Search Tool). In seeded algorithms, short, unique, fixed-length sequences of DNA, called *k*-mers, are used to identify reads that share potential overlap. For example, ATGC would be a 4-mer (k = 4). If two reads share a *k*-mer, these reads likely share an overlap, so the algorithm takes the time to calculate the overlap between the reads (Schatz *et al.*, 2010). Even with seeded algorithms, the overlap computation step remains expensive in both time and computational resources. With these faster methods and a computer that could make 32 million read comparisons per second, the Drosophila project required multiple processors and parallel computing (Myers *et al*., 2000). Furthermore, with modern sequencing technology, many more reads are generated, requiring more comparisons, making the overlap step difficult even for super computers, simply due to the large quantities of data generated through sequencing.

Different OLC algorithms have different criteria for OLC-quality overlaps. For example, the Celera Assembler, one of the first OLC programs, defines read overlaps of at least 40 nucleotides with least 94% similarity as quality overlaps and adds these overlaps

as edges in the assembly graph (Myers *et al.*, 2000). The OLC overlap criteria result in two types of overlaps: true overlaps (Figure 12A) and repeat overlaps (Figure 12B). For example, in figure 12B, an overlap occurs between reads S and T, due to the orange repeat section, not because reads S and T truly overlap one locus in the genome, as in figure 12A. Ideally, repeat overlaps would be excluded from the assembly graph; however, assembly algorithms construct the assembly graph using both types of overlaps, as true and repeat overlaps cannot be distinguished individually. In the assembly graph, the nodes represent actual reads, and edges represent OLC-quality overlaps between these reads (Figure 13). Thus, the genome assembly becomes equivalent to finding a path through the graph that visits each node exactly once (*i.e.*, a Hamiltonian path).

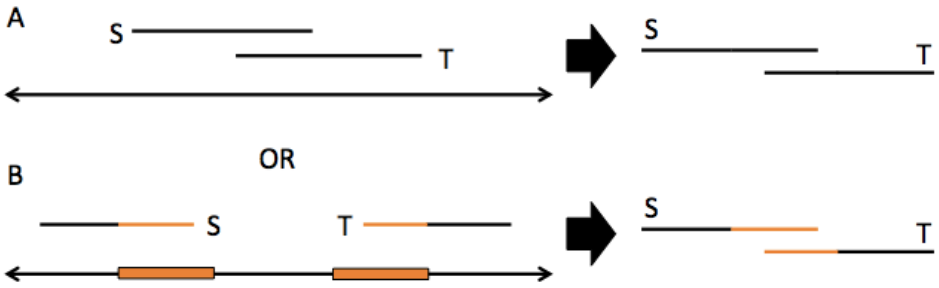

Figure 12. True and false overlaps. The line with arrows represents a genome. 12A: True overlap between reads S and T. 12B: Repeat induced overlap between reads S and T. The orange segments represent repeats within the genome. *Figure derived from Myers et al. (2000).*

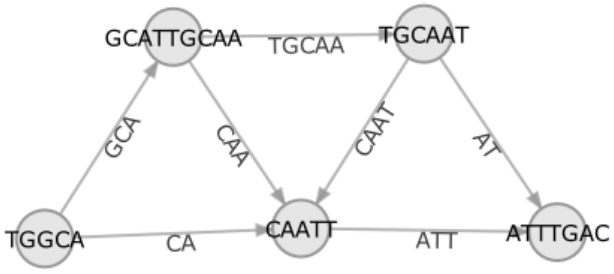

Figure 13. An OLC assembly graph. Nodes are complete reads and edges connect reads that overlap. Note that in an actual OLC assembly graph, reads and overlaps would be much larger. Here, theoretical reads and overlaps are shortened for clarity.

5.2.2 Layout

Finding a path through the OLC graph is not a trivial task. Imagine a graph of millions of nodes and edges. Identifying a Hamiltonian path – a path that visits every node exactly once – is extremely difficult, even for a powerful computer. In order to find such a path, an algorithm must to start at some node and proceed to other, connected nodes. If the algorithm visits a node more than once, it must backtrack, adjust the path, and test this new path. So as the graph size increases, the number of options there are to test grows exponentially. This process of identifying a Hamiltonian path falls into a class of problems called NP-complete, for which the time required to solve a problem increases exponentially with the problem size.

In order to decrease the size of the graph, the OLC assembly graph is simplified in the layout stage, where segments of the assembly graph are compressed into contigs. Recall that contigs are collections of reads that clearly overlap each other and refer to the same overall sequence. Thus in the overlap graph, a contig would be a subgraph, or a group of nodes, with many connections among each other, as they all overlap with each other and refer to the same sequence (Figure 14A and 14B). Once a subgraph is identified, these nodes and edges are compressed into one node, or a contig, thereby simplifying the graph (Figure 14C). Note that within the contig subgraph, only the outer “beginning” and “end” nodes connect to nodes outside of the subgraph. This is because nodes are compressed into contigs until a fork is reached.

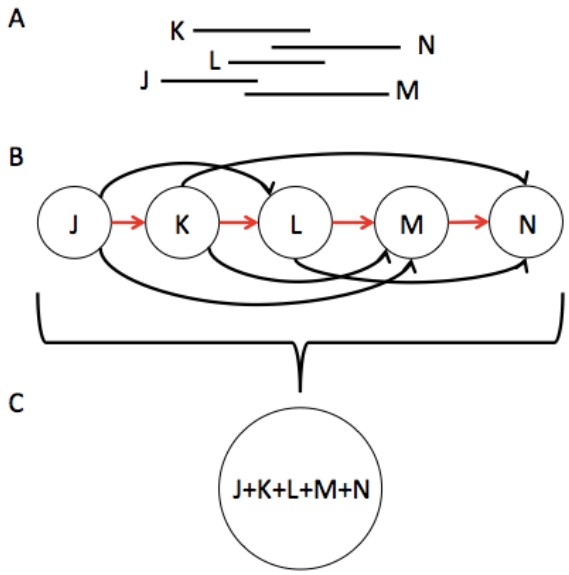

Figure 14. Graphical representation of a unique contig. 14A: The reads J, K, L, M, and N. 14B: An OLC graph of the reads. There is one path that visits every read, highlighted in red. 14C: Reads J, K, L, M, and N are compressed into a single node, J+K+L+M+N. Node J+K+L+M+N represents a unique contig.

Forks are nodes connected to two or more other nodes that do not share any overlap. Contig compression stops at a fork, because forks typically signify the boundary between repeats and unrepeated segments of the genome (Pop, 2009). If contig formation were to continue after a fork, it would diminish the certainty that the contig reads truly cover the same genomic sequence. For example, in figure 15A and 15B, the $R_1 + R_2$ repeat contig is created from a subgraph of overlapping reads. Extension of this contig halts after the repeat section because of a fork. This fork comes from reads whose prefix overlaps with the suffix of the $R_1 + R_2$ repeat contig and whose suffix either overlaps with the prefix of unique contig X or the prefix of unique contig Y (the reads that begin black but turn green or red). Because the suffixes of these reads do not overlap each other, a fork is formed (Figure 15B).

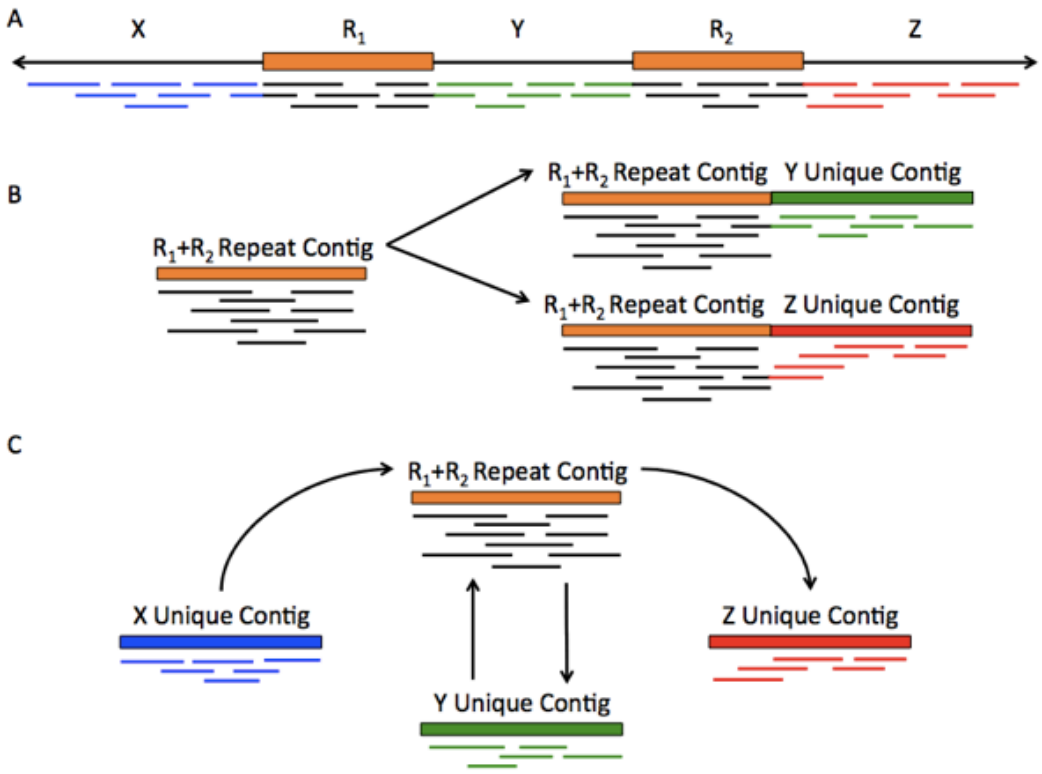

Figure 15. Contig classification. 15A: A genome that has three unique sections – X, Y, and Z – and two repeats – R_1 and R_2. 15B: A fork is formed because the reads that link the $R_1 + R_2$ contig to Y and Z do not overlap on the suffix end. 15C: A contig graph representation of this assembly. Both repeat sections, R_1 and R_2, are compressed into a single contig. *Figure derived from Myers et al. (2000).*

The Celera Whole Genome Assembler introduced the concept of two classes of contigs: unique contigs (unitigs) and repeat contigs. Unitigs are composed of reads that can be unambiguously assembled. Generally, these reads only overlap in one way and do not contain repeats within the genome. Essentially, unitigs are contigs not flagged as repeat contigs.

Repeat contigs have abnormally high read coverage or connect to an abnormally large number of other contigs. For example, in figure 15C, the $R_1 + R_2$ repeat contig is connected to more contigs and has more read coverage than the unique contigs X, Y, and Z. Both the X and Y unique contigs are connected to the prefix of the $R_1 + R_2$ repeat contig as

they both have overlaps to this sequence. Similarly, unique contigs Y and Z have overlaps with the suffix of the $R_1 + R_2$ repeat contig. Additionally, this $R_1 + R_2$ repeat contig is different from other contigs because it has such high read coverage. Abnormally high coverage allows assembly programs to identify repeat contigs during assembly validation through statistics that compare the coverage of each contig (such as the A-statistic mentioned earlier). Contigs with too much coverage are most likely due to over-collapsing of repeats and are flagged as repeat contigs, to be used later in the layout stage (Myers *et al*., 2000).

In the final stages of layout, unique contigs are joined into larger scaffolds. As discussed earlier, the most common way to piece contigs into scaffolds is through mate-pair information. With mate-pair information, assemblers can identify how far reads and unique contigs should be apart from each other. After all unique contigs that can be combined with mate pair information are joined, most OLC algorithms attempt to fill gaps within scaffolds by using repetitive contigs (Batzoglou, 2004).

5.2.3 Consensus

After contig generation and scaffolding, the consensus sequence is derived. At this point, the assembly graph has been reduced to large scaffolds – ideally a single scaffold. A single scaffold would be represented by one node that resulted from collapsing all previous nodes. Starting from the left most read of each scaffold, the OLC algorithm computes the consensus of all of the reads composing each scaffold. Gaps in the genome may still be present if the consensus step had insufficient mate-pair or repeat contig information. If an assembly had gaps, it would result in a fragmented genome composed of multiple scaffolds, because the gaps between the scaffolds could not be joined.

5.2.4 OLC vs Greedy Assembly

Both greedy algorithms and OLC algorithms begin with overlap generation. However, the steps of OLC enable a global analysis of the assembly problem, instead of the local analysis of greedy algorithms (Pop, 2009). This difference is especially apparent in how both algorithms handle repeats. For example in figure 15, the proper genome reconstruction – X, R_1 + R_2, Y, R_1 + R_2, Z – is easily inferred using the OLC method. A greedy-extension assembly would either produce a fragmented assembly – perhaps joining X and R_1 + R_2, leaving Y and Z unassembled – or produce a misassembly by combining X, Y, Z and leaving R_1 + R_2 unassembled (Pop, 2009).

5.2.4 Newbler Assembler

Because the majority of raw reads in the PHAST database are sequenced using a 454 sequencer, it is important to discuss a particular OLC implementation called Newbler. Newbler is a proprietary assembler, distributed by Life Sciences with all 454 sequencers. The program is very effective at assembling 454 data, in part due to its unique way of calculating read overlaps. Instead of exclusively calculating overlaps in "base-space," or centered on the actual base calls of each read, Newbler also calculates pairwise overlaps in "flow-space," or based on the flowgram of each read (Quinn *et al.,* 2008). Inherent to the pyrosequencing technology of 454 sequencers is a degree of uncertainty about the number of bases in homopolymer repeats (repeats of identical bases, such as AAAAA). Recall that flowgrams record the light intensity of each base as it flows over a well of the PTP plate (Figure 4). By normalizing these data, the signal recorded in flowgrams becomes proportional to the number of contiguous bases in each repeat. Such information allows

Newbler to be able to align reads with a higher precision than possible in base-space through comparison of the normalized signals (Miller *et al.,* 2010).

As published in 2005, the basic Newbler algorithm runs two rounds of OLC. In the first round, unitigs are generated, and in the second round these unitigs are used to generate larger contigs by calculating pairwise overlaps between unitigs. These larger contigs are split in areas of low coverage (less than 4 spanning reads) in a quality control step (Margulies *et al.,* 2005). A paper published in 2008 discusses an updated version of Newbler that follows the same general process, but updates the algorithms used at the specific stages to better fit the longer reads generated by the newer GS FLX sequencer, instead of the previously GS 20 model sequencer (Quinn *et al.*, 2008). In the updated Newbler, "detangling algorithms" are used to simplify the assembly graph after the generation of larger contigs in the second round of OLC. After simplifying the graph, each node (or contig) is classified a large contig (greater than 500 bp) or a regular contig (greater than 100 bp). If information on paired end reads is available, Newbler performs an additional scaffolding step. However, since these two publications, Newbler has been updated several times to account for improvements in the 454 sequencing technology, so it is difficult to identify the exact algorithms the current release uses (Miller *et al.,* 2010).

5.3 de Bruijn Assembly Algorithms

A third, general assembly algorithm uses de Bruijn graphs instead of overlap graphs. The de Bruijn assembly offers advantages over the traditional OLC algorithm: (1) no expensive calculations of pairwise overlaps and (2) efficient algorithms exist for computing Eulerian paths, that do not exist for Hamiltonian paths (Pop, 2009).

Instead of calculating overlaps, a de Bruijn assembly begins by calculating *k*-mers. These *k*-mers are used to build the assembly graph, and decrease graph construction time. Each *k*-mer is stored in memory at most once, regardless of the number times it occurs in a genome, which means graph construction can proceed in constant time through a hash table implementation (Miller *et al.,* 2010). Since overlaps are never explicitly computed, as in OLC assemblies, a substantial amount of computing time is saved.

In a de Bruijn assembly graph, the edges of the graph are unique subsequences of length *k* within reads, and the nodes of the graph represent common subsequences of length $k - 1$. Thus, an edge connects two nodes if the suffix of the source node shares an exact match of length $k - 2$ with the prefix of the destination node (Pop, 2009)[2]. For example, in figure 16, the edge CCAA connects the nodes CCA and CAA because there is an overlap of CA. These characteristics are shown in a simplistic de Bruijn graph (Figure 16).

[2] De Bruijn assembly graphs are also used to describe other *k*-mer graphs (Schatz *et al*., 2010; Flicek and Birney, 2009; Zerbino, 2009). These alternative *k*-mer graphs use *k*-mers as nodes, connected by edges when an exact $k - 1$ overlap occurs between nodes (Miller *et al.,* 2010). Edges of such an alternative *k*-mer graph are $k + 1$ sequences. If an assembler ensures the $k + 1$ sequences exist in the genome, the genome assembly process still corresponds to finding an Eulerian path (Pevzner, Tang, and Waterman, 2001; Butler *et al.,* 2008). Such an assembler essentially uses the *k*-mer graph described in this thesis where the true value of $k = k + 1$. If the assembler does not verify these $k + 1$ sequences exist in the genome, the basic assembly process corresponds to identifying a Hamiltonian path, a fundamentally different problem (Compeau *et al.,* 2011).

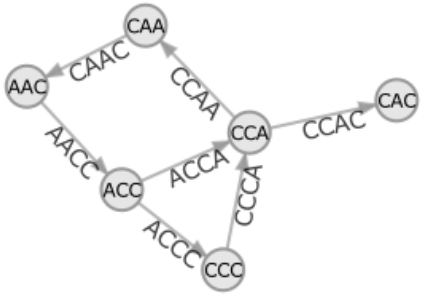

Figure 16. A simple de Bruijn graph with k = 4. The graph corresponds to a series of short reads for the consensus sequence ACCCAACCAC. For simplicity, reverse complement edges and nodes are ignored.

However, figure 16 overlooks a key feature of DNA – DNA is double stranded. In order to properly reflect DNA, de Bruijn graphs must represent the forward and reverse complements as *k*-mers in the assembly graph (Miller *et al.,* 2010). Different de Bruijn assemblers represent reverse complements differently. For example, the Velvet algorithm stores forward and reverse *k*-mers as half nodes, while ABySS stores forward and reverse *k*-mers as a single node with two sides (note these algorithms use the alternative *k*-mer graph described in footnote 2; Miller *et al.,* 2010). Idury and Waterman (1995) simply include both forward and reverse edges and nodes in the assembly graph. In all cases, reverse complements complicate the assembly graph. For the sake of simplicity, reverse complements are not considered in figures in this document.

Following graph construction, error correction is an essential step in de Bruijn assemblies. Unlike OLC implementations, de Bruijn graphs are extremely sensitive to sequencing errors as they can introduce new *k*-mer. Various methods have been proposed to identify errors. One method, spectral alignment, involves preprocessing reads before graph formation (Pevzner *et al*. 2001; Chaisson and Pevzner, 2008). The method defines the set of *k*-mers in genome *G* as G_k. In a *de novo* assembly, since G_k is not actually known, it is approximated through a user-defined multiplicity threshold, *m*. *K*-mers that appear more

often than *m* are quality and part of G_k. *K*-mers that do not meet the threshold are considered "errors." Reads containing these erroneous *k*-mers are altered with the minimum number of substitutions, insertions, and deletions, such that they no longer contain erroneous *k*-mers. This produces a set of raw reads whose entire *k*-mer spectrum falls in G_k, based on the *m* threshold. Although this method is efficient when compared to other preprocessing methods that involve read alignments, it begins to lose its efficiency in repetitive regions or in reads with high error rates (Chaisson and Pevzner, 2008).

Other methods identify and correct errors by analyzing the graph structure. These methods rely on identifying "tips" and "bubbles" (or bulges). Errors less than *k* base pairs from the end of reads tend to create unique *k*-mers that form dead-end "tips" in the assembly graph (Figure 17B; Zerbino, 2009). Errors in the middle of long reads form "bubbles" or "bulges" in the assembly graph, where two paths start and terminate at the same node (Figure 17B). For example, in figure 17B, the path ACG-CGC-GCA-CAT-ATT starts and ends at the same nodes as the path ACG-CGT-GTA-TAT-ATT. After identifying these graph structures, erroneous nodes and edges or low coverage nodes and edges are removed (Schatz *et al.*, 2010). Again, tips and bubbles are handled slightly differently in different de Bruijn assemblers, in part due to the different graph formations in representing forward and reverse complements.

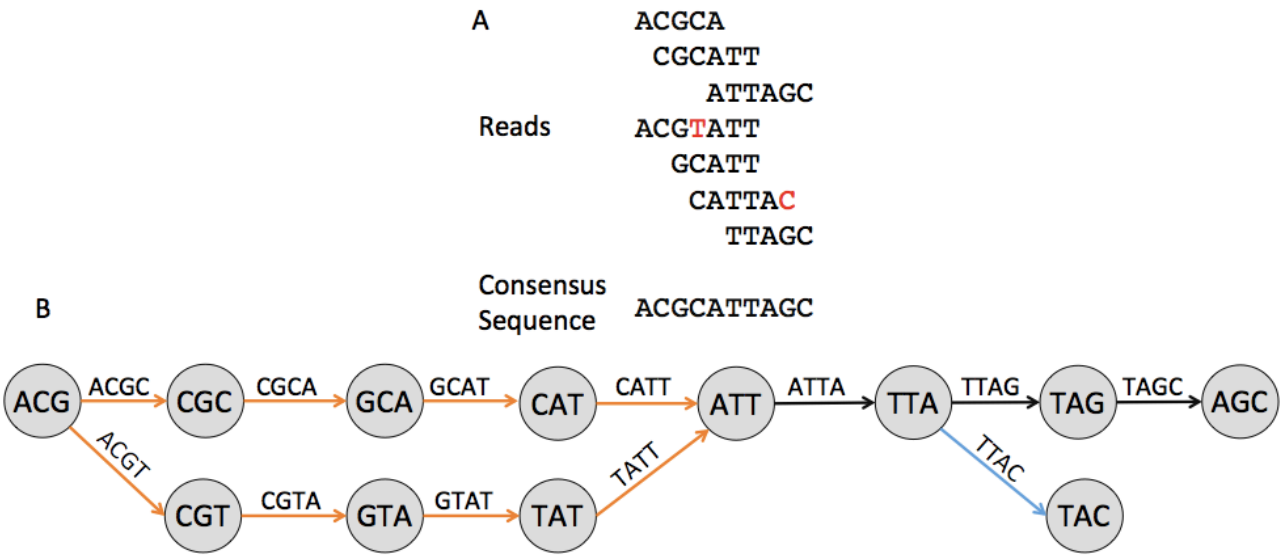

Figure 17. Bubble and tip in a de Bruijn assembly graph. 17A: Reads and consensus sequence. Red bases represent sequencing errors. 17B: de Bruijn graph where k = 4. The orange edges form a bubble. The blue edge is a tip.

After error correction, most de Bruijn assembly algorithms compress the assembly graph into contigs. As with OLC assembly graphs, these contigs are non-branching subgraphs of the de Bruijn graph (Schatz *et al.,* 2010). Contigs are extended until branches, which are essentially forks (pg 27). Repeats cause branches and create ambiguity in the graph, or multiple ways to traverse the assembly graph. Assemblers must identify an Eulerian path through this graph – a path that visits each edge exactly once. Unlike the Hamiltonian path (a path that visits every node exactly once) required to assemble OLC graphs, efficient algorithms exist for calculating Eulerian paths. Unfortunately, the ambiguity created by repeats means multiple Eulerian paths exist (Figure 18B, 18C). The job of a good de Bruijn assembler is to identify the correct Eulerian path, representing the correct genome assembly (Figure 18C).

Figure 18. Multiple Eulerian paths in de Bruijn assembly graph. Numbers on graphs represent the order of a traversal through a graph. 18A: Reads in relationship to the original sequence. 18B: A wrong Eulerian path for k = 5 (video available on PHAST website and CD copy). 18C: The correct Eulerian path for k = 5 (video available on PHAST website and CD copy).

Information critical to solving this problem is lost in the standard de Bruijn formulation, as reads are broken into multiple k-mers. In a perfect world, WGS would produce reads with equal length of k bases that perfectly sample the genome (Figure 18A). In such a world, each edge in the de Bruijn graph would be a complete read of length k, and no information would be lost. Unfortunately, sequencing technologies do not produce reads

of equal length or sample the genome perfectly. Instead, assemblers must use a *k* value that is less than the length of the majority of the reads (Figure 19A). For example, in figure 19 a value of 4 must be selected, because at $k \geq 5$ the assembly graph loses a key read of length 4, TAGC, which is critical for connecting this graph, as shown in figure 19B. When $k \geq 5$, the graph does not contain an Eulerian path, resulting in a fragmented assembly of multiple contigs (Figure 19B). Once a value for *k* is chosen, reads longer than *k* become fragmented into *k*-mers, so that the resulting de Bruijn graph no longer represents full reads as edges (Figure 19C). For example in figure 19C, the read CGCATT is broken into the edges CGCA, GCAT, and CATT because $k = 4$. If $k = 6$, the length of CGCATT, then this read would be preserved as a single edge.

The classical Eulerian path assembly approach loses edge linkage information, as in figure 19C, when reads are broken into multiple edges in a de Bruijn graph. Losing this information is a problem because read linkage information can reveal the correct Eulerian path when multiple Eulerian paths exist due to repeats. For example, without read linkage information, you could not tell which path depicted in figure 19D and 19E corresponds to the actual genome sequence, as you could progress from node CAT to either node ATT or ATC. Pevzner *et al.* (2001) proposed the Eulerian super-path in attempt to resolve this graph ambiguity. An Eulerian super-path is an Eulerian path constructed from sub-paths that correspond to reads (Figure 19D). In this way, the actual read data is preserved and fewer alternative Eulerian paths exist. For example, the Eulerian super-path approach identifies the correct assembly, figure 19D, from the incorrect assembly, figure 19E. Read linkage information in figure 19C tells the assembler to visit ATT after the first time it visits CAT.

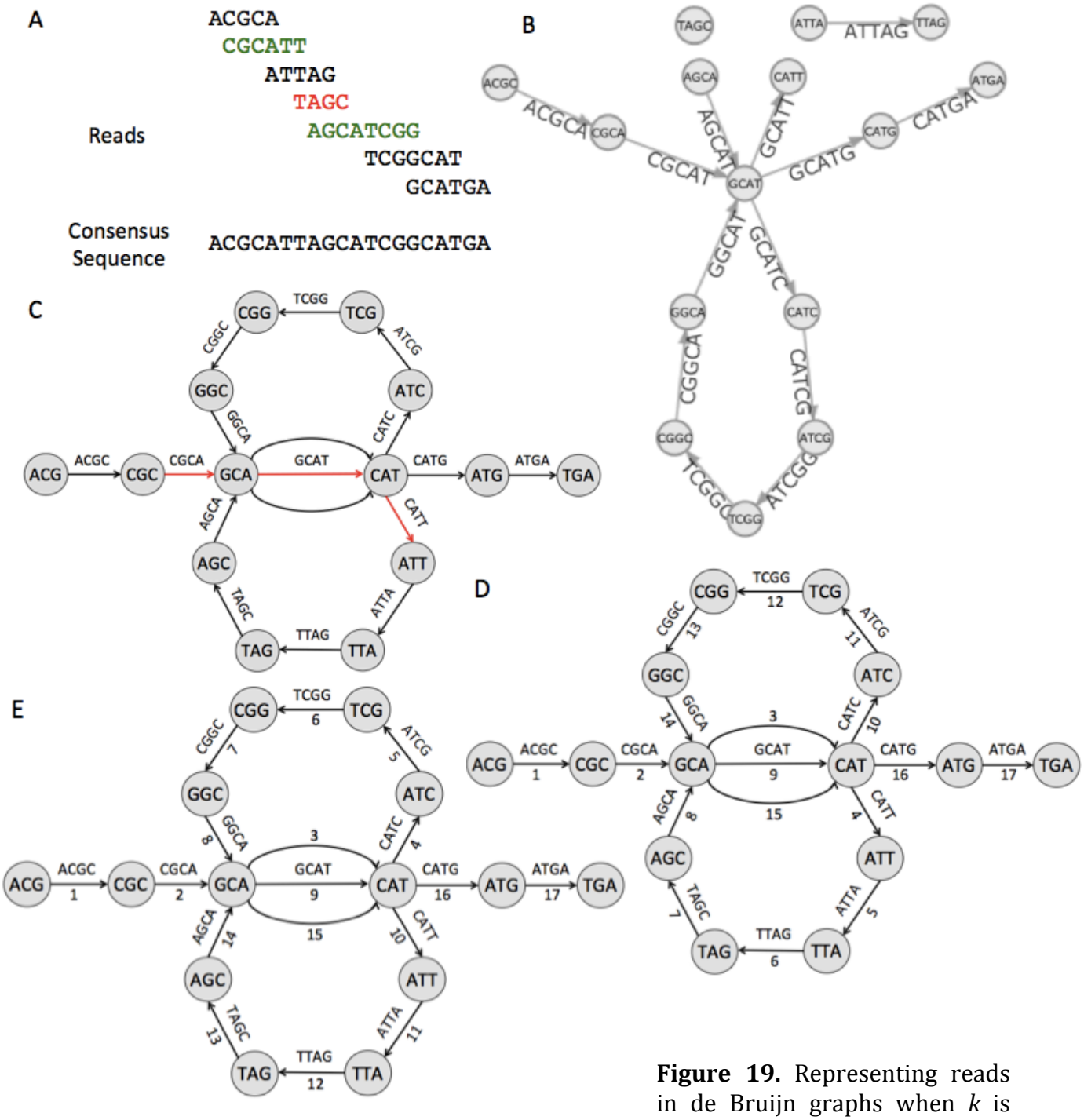

Figure 19. Representing reads in de Bruijn graphs when k is less than read length. 19A: The reads are of varying lengths. The green reads identify the correct path in the Eulerian super-path problem. The red read causes no solution when $k \geq 5$. 19B: Choosing $k = 5$ results in a graph that has no solution. The node TAGC is unattached and would be thrown out. Also, the edge CATTA is missing, because that edge does not exist in within a single raw read, and could not be identified as a k-mer. 19C: The highlighted edges are edges that came from the read CGCATT. 19D: The Eulerian super-path of this graph is the correct path. In the video edges are not highlighted based on linkage information, but are highlighted in the sequence that linkage information would tell the assembler is the correct path (video available on PHAST website and CD copy). 19E: A wrong Eulerian path (video available on PHAST website and CD copy).

In 2007, Medvedev *et al.* proved the Eulerian super-path problem to be NP-Hard, just as difficult as the Hamiltonian path problem of OLC assemblers. Nevertheless, most de Bruijn assemblers use the idea of mapping reads to the de Bruijn assembly graph in order to resolve repeat branches. As a final step, most de Bruijn assemblers use mate pair information, when available, to further simplify the graph in a scaffolding process.

5.4 MIMICKING INTELLIGENT READ ASSEMBLER (MIRA)

PHAST uses an assembly program called MIRA (Mimicking Intelligent Read Assembly) that has both OLC and greedy components (Chevreux, 2005). As with most assemblers, MIRA begins by cleaning reads of low quality bases and vector vestiges. These "cleaned" sections of reads are called high confidence regions (HCR) and are used for the general assembly. Low confidence regions (LCR) of a read are stored for later use to further verify HCR contigs. Using the HCR data, MIRA performs a fast read comparison – where each read is compared to every other read and its reverse complement – in order to identify potential overlaps. Originally, two algorithms, called DNA-SAND and ZEBRA, were used to scan for potential overlaps (Chevreux, 2005). However, in the most recent MIRA release, an unpublished scanner, SKIM, has replaced these algorithms (FreeLists, 2011). According to Chevreux, the author of MIRA, SKIM is very similar to a published algorithm called SlideSort (FreeLists, 2011; Shimizu and Tsuda, 2011). Both SKIM and SlideSort identify reads with potential overlap based on common *k*-mers. In addition to returning reads with possible overlaps, these algorithms return an approximate offset, or distance from the end of a read, for the sequence alignment (Figure 20A).

MIRA scores and filters overlap regions using a banded Smith-Waterman alignment algorithm. The banded Smith-Waterman alignment algorithm is a simplification of the

standard Smith-Waterman alignment algorithm. In a normal Smith-Waterman alignment, a path corresponding to the local alignment is generated in an n_1 x n_2 alignment matrix, where n_1 and n_2 are the lengths of two different reads. This path starts and ends at the offset positions of the alignment between reads. Instead of calculating the entire n_1 x n_2 alignment matrix, the banded Smith-Waterman alignment calculates a subset (or band) of that matrix based on the offset approximations from SKIM (Figure 10B). The band area is determined by multiplying the minimum distance between the overlap offsets (which is equivalent to estimating the length of the overlap) by k, where k is a parameter representing the band width (usually, $30 \leq k \leq 70$). The banded alignment requires k because the parameter acts as an error buffer to account for indels in the alignment. This simple modification decreases the complexity of the alignment problem from $O(n_1 * n_2)$ to $O(k * \min[n_1,n_2])$, as only small bands, roughly the size of the overlap need to be calculated (Chevreux, 2005).

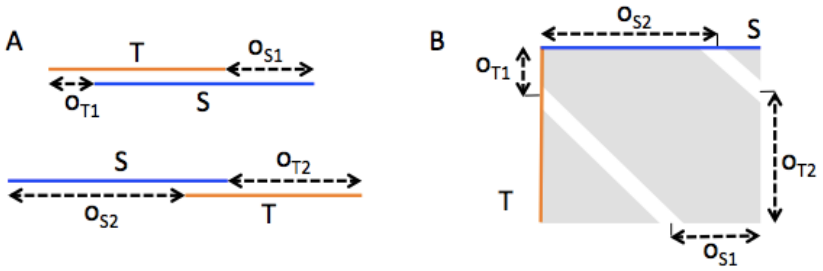

Figure 20. Simplifying a Smith-Waterman alignment using overlap offsets. 1A: Reads S and T overlap in two different positions. Each overlap has a different offset, labeled by o_n. 1B: The bands of a banded Smith-Waterman alignment matrix (shown in white) inside the area of a normal Smith-Waterman alignment matrix (shown in grey). The offset length is used to approximate the band locations. These bands are slightly larger than the predicted offset to account for possible errors. *Figure derived from Chevreux (2005).*

The resulting Smith-Waterman alignments of reads are scored based on similarity in the alignment. In order to fit next-gen shotgun sequencing data, the alignment-scoring

method is slightly different from the standard Smith-Waterman scoring scheme. When a known base, perhaps "A", aligns to an unknown base, "N", instead of receiving a misalignment penalty, zero is added to the score. Unknown bases occur when a sequencer sees a base in the sequencing signal but cannot confidently determine the specific base. An alignment of a known base to an unknown base is neither a misalignment nor a perfect alignment, so the score remains unchanged. In order to account for the low error rate of next-gen sequencers, the alignment score is incrementally scaled down according to the number of gaps or long stretches of mismatches in an alignment. With next-gen sequencers, errors are unlikely, so if an alignment differs at more than a few consecutive bases, it is unlikely these reads share a true overlap. Overlaps with scores close to a perfect overlap score are considered quality and become edges in the overlap graph.

MIRA uses quality overlaps to build a weighted overlap graph, in which an edge's weight is based on the alignment score and the length of the overlap. The weighted overlap graph is used to construct contigs. MIRA divides the process of contig construction into two modules: a pathfinder module and a contig module. The pathfinder module identifies the next node in the overlap graph for contig extension and tries to add it to a growing contig. The contig module either accepts and incorporates the read (node) or rejects the read (Figure 21).

The pathfinder module begins by identifying a node in the assembly graph with the most highly weighted edges. This node is submitted to the contig module and serves as an "anchor" for contig formation since it has the longest and highest quality overlaps with the greatest number of reads in the overlap graph. Next, the pathfinder traverses the graph and identifies the next node to submit to the contig module for incorporation. As of 2005, the

pathfinder module traverses the graph using an improved greedy algorithm, which helps avoid misassemblies or highly fragmented assemblies. The basic algorithm iteratively expands out by *n* nodes for *m* levels. The module begins by identifying *n* nodes with the largest edge weights that are also adjacent to the current contig nodes. The pathfinder's goal is to determine which of these *n* nodes is most likely to lead to the best possible future overlaps, in addition to the current overlap. Each of the *n* nodes serves as the beginning "seed" of a path that looks ahead at future overlaps, called a look-ahead path. Using a recursive algorithm, the pathfinder module traverses each partial path for a maximum of *m* recursions (usually 4 to 5 recursions). At each recursive step, the algorithm selects the next *n* largest edges to continue the look-ahead path. After generating every look-ahead path, MIRA computes the maximum score of each seed by calculating the sum of the largest edges in the seed's look-ahead path. The pathfinder module selects the seed with the largest maximum score and submits it to the contig module for evaluation.

The contig module analyzes the impact of this new read on the consensus of the contig. If the nucleotides of the new read are largely consistent with the current consensus sequence, then the contig module accepts this read and adds it to the contig. Otherwise, if too many base differences occur, the contig module rejects this read. If a read is rejected, the pathfinder will search for the next best partial path. The algorithm will submit the seed read of the next best path to the contig module, so the pathfinder may submit the same, rejected read at a different position within the contig. When the pathfinder module has exhausted all possibilities for contig growth, the algorithm searches for a new anchor point

to start a new contig. The entire contig formation process is complete once every read is part of a contig or identified as a single-read contig[3] (Figure 21; Chevreux, 2005).

```
while overlap graph contains reads not in a contig:
      pathfinder selects anchor read
      while un-submitted contig overlaps exist:
            pathfinder submits seed read of best look-ahead path
            if contig module accepts read:
                  read added to growing contig
```

Figure 21. Pseudo-code for MIRA contig construction process.

The first assembly draft consists of contigs generated from the high confidence parts of reads. As a final step, MIRA attempts to link contigs and fix misassemblies by re-analyzing low confidence parts of reads. In this process, LCR's are unmasked and aligned to the existing consensus sequence. In most cases, the HCR's overrule discrepancies in LCR's. However sometimes, in areas with particularly low coverage, LCR's of several reads may agree with each other and overrule an incorrect call made by a small number of HCR's covering this section of the contig. In addition, several highly similar LCR's may extend past a contig, joining it to another contig. In this way, MIRA makes maximal use of all data gathered from the reads.

The MIRA parameters used in PHAST are general parameters that should produce a reasonable assembly of most small phage genomes. PHAST is not optimized to perfectly assemble one particular phage, so it might be possible to improve each assembly produced by PHAST by slightly adjusting the MIRA assembly parameters. However, most of the MIRA assembly parameters are fixed in PHAST, because the website is intended to be a general,

[3] Technically, a single read is not a contig. However, MIRA will put such a read in the same contig data structure.

genome assembly teaching tool. Fixing these parameters simplifies the assembly process and helps a user focus on the more relevant, essential parameters in a genome assembly.

5.5 Conclusion

There are several classes of genome assembly methods, with different advantages and disadvantages. There is no single best assembly algorithm; the proper assembly algorithm depends on the target genome complexity, the size of the target genome, and most importantly the type of reads. In general, OLC implementations perform very well with longer reads, such as Sanger reads or the newer and longer 454 reads, while de Bruijn implementations excel in assembling shorter reads, such as Illumina reads (Schatz *et al.,* 2010). As sequencing technologies change and improve, assembly techniques must also adapt to fit the data.

6 RESULTS

I developed PHAST (Phage Assembly Suite and Tutorial) to be a web-based, interactive genome assembly educational tool. PHAST is freely available online at http://compbio.davidson.edu, and the code will work on most Unix-based servers. The homepage is a general outline of the genome assembly process. Users who want to immediately begin assembly and quickly learn the features of PHAST should go to the "Quick Walkthrough" page. The majority of this Results section is available online through PHAST. Below is a brief highlight of PHAST's major features.

PHAST includes tutorials that provide an overview of current genome assembly algorithms, described in detail in Chapter 5 of this thesis. In addition to learning about genome assemblies, users can perform their own assemblies and discover the effects concepts introduced in the tutorials have on the quality of a genome assembly. For example, a user can decrease genome coverage by using a percentage of raw reads. A user can choose not to clean 454 reads of vector sequences, which inevitably complicates the assembly by generating false overlaps. A user can also compare an assembly with a different assembly that used two sets of reads. In some cases, a sequencing facility sequences the genome of a phage more than once in order to piece together contigs or resolve genomic areas of low coverage. The multi-read assembly feature allows a user to assemble a phage genome using all available reads, divided among files from different sequencing runs. An advanced user can select reads from two separate genomes to find out if the software can assemble each genome separately. In a perfect world with a perfect assembler, a two-genome assembly would produce two large contigs, corresponding to the

two different genomes. In some cases, like with the acadian and timshel phages, MIRA will produce an assembly with two large contigs; however, in other cases, like with the bpbiebs31 and euphoria phages, the assembly is extremely fragmented.

After assembling several genomes, a user can compare assemblies using an integrated dotplot tool called gepard (GEnome PAir - Rapid Dotter; Krumsiek *et al.,* 2007). With gepard, users can visualize the relationship of one genome to another and assess the effect of the parameters used in an assembly. For example, users can compare two assemblies of the same set of reads, one using 100% of the reads and one using 10% of the reads. In the resulting dotplot, a user can quickly identify fragmented areas of the 10% assembly caused by a decrease in genome coverage. A series of assemblies using different amounts of the available reads will allow users to determine a minimum threshold necessary for complete assembly. Finally with gepard, users can also identify similar genomic regions between two different phages, just as PHIRE researchers do to classify unknown phage genomes (Pope *et al.*, 2011).

Future versions of PHAST could include features designed to further enrich the user experience. A web-based, multiple genome comparison tool would be useful not only to PHAST but to the scientific community in general. Currently, to compare several genomes at once, a user must download each genome's fasta file and compare them using a desktop application such as MAUVE (Darling *et al.,* 2004). A web-based interface, perhaps coded in JavaScript and HMTL5, that allows a user to visualize and interact with a multiple genome alignment would streamline the comparison process in PHAST and have many other applications across the web. Also, PHAST could be expanded to support multiple genome assembly methods. This feature would allow the user to run an assembly using a standard

OLC assembler, such as the Celera Whole Genome Assembler, or a standard de Bruijn assembler, such as the Euler Assembler. A user could then determine which assembly method is best suited for their specific sequence data. Such features would further enhance the ability of PHAST to teach genome assemblies in an interactive, self-contained environment.

7 CONCLUSIONS

The future holds many opportunities and challenges for improvement and innovation in the emerging field of genome assembly. Assembly algorithms must adjust to accommodate constantly changing genomic input data. In 2000, most sequencing projects used long Sanger reads. Roughly seven years later, the standard whole genome sequencing method had shifted to cheaper next-gen sequencers (454, Illumina, etc.) that generate short reads. Now, companies promise sequencers that generate extremely long reads for a fraction of the current cost. With further advancement in sequencing technology, there will be opportunities to improve and develop assembly methods tailored to fit the characteristics of new sequencing data.

The methodology used to validate and compare genome assemblies must also keep pace with the developing sequencing technology. As whole genome sequencing becomes commonplace, sequencing facilities will have an even greater need to automate genome assembly validation tools to replace the current expensive and tedious manual approaches. In addition, tools designed to simultaneously compare multiple entire genomes will become particularly important to make use of the rapidly increasing number of whole genomes sequenced. Scientists will want to know how genomes differ on a large, whole genome structure scale in addition to a small, localized base pair scale. Efficient tools designed to visualize complex information will play a crucial role understanding genomic differences between both individuals and species.

Finally, researchers must continue to develop tools designed to assemble, analyze, and validate genomic data from entire communities (metagenomics), rather than specific

individuals. The study of metagenomics explores the interactions between species in entire communities through sequencing complex samples recovered from an environment. Development in metagenomic tools will give researchers the ability to better understand the functional differences at various levels in communities and populations, which is currently too difficult using today's technology. As sequencing technology and assembly algorithms advance, the scientific community will have the opportunity to gain a greater understanding of organisms and their relationship to one another than was ever previously imaginable.

8 REFERENCES

1. 5500 Series SOLiD Sequencers [Internet]. Life Technologies: c2011 [cited 2012 Jan 24]. Available at: http://media.invitrogen.com.edgesuite.net/ab/applications-technologies/solid/solid-5500.html
2. Batzoglou S. Encyclopedia of Genetics, Genomics, Proteomics and Bioinformatics. (Jorde LB, Little PFR, Dunn MJ, Subramaniam S, editors.). Chichester, UK: John Wiley & Sons, Ltd; 2004.
3. Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES, Nusbaum C, Jaffe DB. ALLPATHS: De novo assembly of whole-genome shotgun microreads. Genome Research. 2008 February 21;18(5):810–820.
4. Campbell AM, Heyer LJ. How are Genomes Sequenced. In: Discovering Genomics, Proteomics, & Bioinformatics. 2nd ed. San Francisco: Benjamin Cummings; 2007. p 34-59.
5. Chaisson MJ, Pevzner PA. Short read fragment assembly of bacterial genomes. Genome Research. 2008 February 1;18(2):324–330.
6. Chevreux B. MIRA: an automated genome and EST assembler. Ruprecht-Karls University, Heidelberg, Germany. 2005.
7. Chevreux B. Sequence assembly with MIRA3: The Definitive Guide [Internet]. 2010 [cited 2012 Feb 15]. Available from: http://mira-assembler.sourceforge.net/docs/DefinitiveGuideToMIRA.pdf
8. Compeau PEC, Pevzner PA, Tesler G. How to apply de Bruijn graphs to genome assembly. Nature Biotechnology. 2011 November 1;29(11):987–991.

9. Darling ACE, Mau B, Blattner FR, Perna NT. Mauve: multiple alignment of conserved genomic sequence with rearrangements. Genome Research. 2004 July;14(7):1394–1403.
10. Deonier RC, Tavare S, Waterman MS. Computational Genome Analysis. New York: Springer; 2005.
11. DNA sequencing - the 454 method [Internet]. London: Wellcome Trust: 2011 [cited 2012 Jan 31]. Available from: http://www.wellcome.ac.uk/Education-resources/Teaching-and-education/Animations/DNA/WTX056046.htm
12. DNA sequencing - the Illumina method [Internet]. London: Wellcome Trust: 2011 [cited 2012 Jan 31]. Available from: http://www.wellcome.ac.uk/Education-resources/Teaching-and-education/Animations/DNA/WTX056051.htm
13. Flicek P, Birney E. Sense from sequence reads: methods for alignment and assembly. Nature Methods. 2009 November;6(11s):S6–S12.
14. FreeLists: Mailing List Archive for mira_talk [Internet]. Avenir Technologies: c2000-2011 [cited 2012 Feb 24]. Available from: http://www.freelists.org/archive/mira_talk
15. Hatfull GF, Pedulla ML, Jacobs-Sera D, Cichon PM, Foley A, Ford ME, Gonda RM, Houtz JM, Hryckowian AJ, Kelchner VA, et al. Exploring the Mycobacteriophage Metaproteome: Phage Genomics as an Educational Platform. PLoS Genetics. 2006;2(6):e92.
16. Huerta M, Downing G, Haseltine F, Seto B, Liu Y. NIH working definition of bioinformatics and computational biology. The Biomedical Information Science and

Technology Initiative Consortium (BISTIC) Definition Committee of National Institutes of Health (NIH). 2000;17.

17. Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. Journal of computational biology: a journal of computational molecular cell biology. 1995;2(2):291–306.
18. Introduction to SMRT Sequencing [Internet]. Pacific Biosciences of California Inc: c2010 – 2011 [cited 2012 Jan 24]. Available from: http://www.pacificbiosciences.com/video-gallery/index.html
19. Ion Torrent technology how does it work? [Internet]. Life Technologies: 2012 [cited 2012 Jan 24]. Available from: http://www.invitrogen.com/site/us/en/home/Products-and-Services/Applications/Sequencing/Semiconductor-Sequencing/Semiconductor-Sequencing-Technology/Ion-Torrent-Technology-How-Does-It-Work.html
20. Kim S, Tang H, Mardis E. Genome sequencing technology and algorithms. Norwood: Artech House Publishers; 2008.
21. Krumsiek J, Arnold R, Rattei T. Gepard: a rapid and sensitive tool for creating dotplots on genome scale. Bioinformatics. 2007 April 30;23(8):1026–1028.
22. Mardis ER. The impact of next-generation sequencing technology on genetics. Trends in Genetics. 2008 March;24(3):133–141.
23. Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen Y-J, Chen Z, et al. Genome sequencing in microfabricated high-density picolitre reactors. Nature. 2005 September 15;437(7057):376–380.

24. Mate Pair Sequencing [Internet]. Illumina Inc: 2012 [cited 2012 Feb 1]. Available from: http://www.illumina.com/technology/mate_pair_sequencing_assay.ilmn
25. Medvedev P, Georgiou K, Myers G, Brudno M. Computability of models for sequence assembly. Algorithms in Bioinformatics. 2007:289–301.
26. Miller JR, Delcher AL, Koren S, Venter E, Walenz BP, Brownley A, Johnson J, Li K, Mobarry C, Sutton G. Aggressive assembly of pyrosequencing reads with mates. Bioinformatics. 2008 October 24;24(24):2818–2824.
27. Miller JR, Koren S, Sutton G. Assembly algorithms for next-generation sequencing data. Genomics. 2010 June 1;95(6):315–327.
28. Mycobacteriophage Database [Internet]. 2012 [cited 2012 Feb 21]. Available from: http://phagesdb.org
29. Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, Flanigan MJ, Kravitz SA, Mobarry CM, Reinert KH, Remington KA, *et al*. A whole-genome assembly of Drosophila. Science. 2000 March 24;287(5461):2196–2204.
30. Nakabachi A, Yamashita A, Toh H, Ishikawa H, Dunbar HE, Moran NA, Hattori M. The 160-Kilobase Genome of the Bacterial Endosymbiont Carsonella. Science. 2006 October 13;314(5797):267–267.
31. Nanopore Sensing [Internet]. Oxford: Oxford Nanopore Technologies: c2008–2011[cited 2012 Jan 24]. Available from: http://www.nanoporetech.com/sections/index/55.
32. Narzisi G, Mishra B. Comparing De Novo Genome Assembly: The Long and Short of It Aerts S, editor. PLoS ONE. 2011 April 29;6(4):e19175.

33. Nebula Awards [Internet]. Science Fiction & Fantasy Writers of America: 2009 [cited 2011 July 7]. Available from: http://www.sfwa.org/nebula-awards/rules/.
34. Paired-End Sequencing [Internet]. Illumina Inc: 2012 [cited 2012 Feb 1]. Available from: http://www.illumina.com/technology/paired_end_sequencing_assay.ilmn
35. Perkel JM. Sanger Who? Sequencing the next generation. Science. 2009 April 6;324(5924):275–279.
36. Pevzner PA, Tang H, Waterman MS. An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences of the United States of America. 2001 August 14;98(17):9748–9753.
37. Phanning the Phlames: Expansion of the Phage Hunters Integrating Research and Education (PHIRE) Program [Internet]. Chevy Chase (MD): Howard Hughes Medical Institute: c2012 [cited 2012 Feb 21]. Available from: http://www.hhmi.org/grants/professors/hatfull.html
38. Pop M. Genome assembly reborn: recent computational challenges. Briefings in Bioinformatics. 2009 June 7;10(4):354–366.
39. Pop M, Salzberg SL. Bioinformatics challenges of new sequencing technology. Trends in genetics : TIG. 2008 March;24(3):142–149.
40. Pope WH, Jacobs-Sera D, Russell DA, Peebles CL, Al-Atrache Z, Alcoser TA, Alexander LM, Alfano MB, Alford ST, Amy NE, et al. Expanding the Diversity of Mycobacteriophages: Insights into Genome Architecture and Evolution Aziz R, editor. PLoS ONE. 2011 January 27;6(1):e16329.
41. Principle of Pyrosequencing Technology [Internet]. QIAGEN: 2012 [cited 2012 Jan 31]. Available from: http://www.pyrosequencing.com/DynPage.aspx?id=7454.

42. Quinn NL, Levenkova N, Chow W, Bouffard P, Boroevich KA, Knight JR, Jarvie TP, Lubieniecki KP, Desany BA, Koop BF, et al. Assessing the feasibility of GS FLX Pyrosequencing for sequencing the Atlantic salmon genome. BMC Genomics. 2008;9(1):404.
43. Reading and Analysis [Internet]. ZS Genetics Inc: c2004 – 2012 [cited 2012 Feb 26]. Available from: http://www.zsgenetics.com/reading%20&%20analysis.htm
44. Russell PJ. iGenetics: A Molecular Approach (3rd Edition). 3rd ed. Benjamin Cummings; 2010 p. 21.
45. Schatz MC, Delcher AL, Salzberg SL. Assembly of large genomes using second-generation sequencing. Genome Research. 2010 September 1;20(9):1165–1173.
46. Shimizu K, Tsuda K. SlideSort: all pairs similarity search for short reads. Bioinformatics. 2011 February 8;27(4):464–470.
47. tSMS How It Works [Internet]. Cambridge (MA): Helicos BioSciences Corporation: c2008 [cited Feb 26]. Available from: http://helicosbio.com/Technology/TrueSingleMoleculeSequencing/tSMStradeHowItWorks/tabid/162/Default.aspx
48. Wetzel J, Kingsford C, Pop M. Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies. BMC Bioinformatics. 2011 April 13;12(1):95.
49. Wilton S. Dideoxy Sequencing of DNA. In: eLS [Internet]. Chichester, UK: John Wiley & Sons, Ltd; 2002 [cited 2012 Jan 23]. Available from: http://dx.doi.org/10.1038/npg.els.0003768.

50. Zerbino DR. Genome assembly and comparison using de Bruijn graphs. Hinxton, Cambridge, United Kingdom. 2009;164.

9 APPENDIX 1: CODE

The latest release of PHAST is available for download at
http://compbio.davidson.edu/phast_release.zip