

Sequencing a Microbial Genome

Maulik Shah

Computational Biosciences Program
Department of Mathematics and Statistics
Arizona State University

April 28th, 2005

Dr. Jeffrey Touchman
Dr. Rosemary Renaut
Dr. Phillip Stafford

Report Number: 05-10

Abstract

Although many genomes are available for download today, the underlying technologies should not be taken for granted. By using shotgun sequencing techniques and a gauntlet of informatics, we are able to produce high-quality DNA sequence. We will first look at some of the robotics and chemistries of preparing DNA as samples for the sequencing instruments. Then we will look at the series of applications used in taking raw data signals, converting them to sequence and then finally assembling the data into a single genome. Highlighted will be some of the techniques used to speed the informatics processes as well as some of the challenges that informatics faces in processing data and assembling the genome.

Introduction

Sequencing an organism's genome was something of a dream 20 years ago. Since then, we have made immense advances in microbiology and the dream has been realized hundreds of times over. However, the process is still not completely straight forward. The process is not turn-key and nor will current technologies allow the process to become much simpler. Despite the limitations, there are still many areas in which the process has become high-throughput. We will examine the process of sequencing a microbial genome starting from the DNA, through the robotics, informatics, and finishing phases.

Goals

1. Describe the process of shotgun sequencing – a stochastic process results in a high-throughput process.
2. Examine the use of robotics in preparation of DNA and the operation of DNA Sequencing instruments.
3. Review the workings of a quality control mechanism called the “data curation pipeline” that uses parallel functionality when appropriate to take advantage of computing resources.
4. Assess the challenges of re-assembling a genome after the shotgun process.
5. Review the challenges of the non-high-throughput parts of the process.

Shotgun Sequencing

Sequencing a genome is no small feat. Today's technologies make this effort one that requires systematic procedures and high-throughput processes that must be well-coordinated. We explore why we need these procedures and what they are.

Given a piece of DNA, today's sequencing instruments can sequence approximately 1000 base pairs. By sequence, I mean discern the sequence of that piece of DNA. The cost of reading these 1000 base pairs is approximately \$5; or half of a cent per base pair. A quick calculation shows that this would mean that a four million base pair genome would cost only \$20,000. However, the true cost of such an effort is closer to \$300,000 for reasons we will now explore.

The first step in the shotgun sequencing process is to literally "shotgun" the genome. Shotgunning the genome is a colorful metaphor for fragmenting the genome into many manageable chunks. These smaller fragments can then be strategically sequenced to ensure that the entire genome sequence can be discerned. The process of fragmenting is completely random. We are able to control the size of each fragment, however, at the end of the shotgunning process, we have no way to differentiate the fragments or know from which segment of the genome it came. We also know that these fragments overlap each other. Presumably, we used many copies of the genome to create this collection of fragments. This collection, or shotgun library, now contains all of the information we need in a more manageable form of DNA.

The next step is to ligate these fragments into a vector and label them. A vector is a circular piece of DNA that can be opened and have a foreign piece of DNA inserted. This vector can then be placed into a living cell of *E. coli* for amplification. Because *E.*

coli possess all the biological machinery for amplifying DNA and because they are easy to control, they are an excellent choice for the next step of the process. We need to amplify the quantity of each fragment. E. coli can be injected with a vector and then encouraged and multiply. When E. coli grow and divide, they copy not only their own circular genome, but the vector and its inserted fragment as well. Since we track each colony using robotics and libraries of cell lines, we can identify individual DNA fragments.

At this point, we remove the DNA from the E. coli and prepare for it to be sequenced. Remember, that we can only read 1000 base pairs of this fragment. We can also read this fragment from two directions. We can read the first thousand base pairs, or we can read the last thousand. During the process of sequencing a microbial genome, typically, three different fragment sizes are used; all are greater than 1000 in size. The various sizes are an important part of the overall sequencing strategy. In the next stage we use sequencing instruments, i.e. sequencers, to determine the sequence of these fragments [14].

As mentioned, the fragments were generated in a random fashion. To determine the sequence of the genome, we need to ensure a fragment has been taken from each part of the genome. However, due to statistical distributions for a project of this size, certain regions of the genome would not be represented by any fragment. We can create enough fragments to cover the genome ten times over and still only have the sequence representing 99.9% of the genome. Although sequencing the genome ten-fold redundancy is expensive, it is remarkably faster than sequencing each part of the genome once. The process is extremely high throughput because each fragment is treated identically and a

combination of robotics and informatics make this process very straight forward [15]. If one were to try use only \$20,000 to sequence each section of the genome once, one would spend an immense amount of time trying to determine the correct primer sequences to sequence the particular 1000 base pairs of the sequence. The time required to do such a process would be an order of magnitude longer and thus require many more resources. The shotgun sequencing methodology successfully, albeit redundantly, sequences the genome in a very time and resource efficient manner.

Having the sequences of each end of each fragment is still not enough. We will need to determine the entire sequence. This process of reconstituting the original genome from the fragments is called “assembling the genome”. This topic will be covered in the appropriate section. Finally, as mentioned before, there will still remain .01% genome to be sequenced. This portion of the genome may not have been sequenced for both biological and statistical reasons [15]. We will explore this issue of “finishing” in a later section as well.

The goals of shotgun sequencing are simple. It was designed to allow today’s sequencing technologies to quickly determine the sequence of a genome. To do this efficiently, we employ a variety of robotics and informatics. Let us now look at the robotics involved in making this process efficient and feasible.

Data Organization

Let us briefly explore how data is organized. The goal is to be able to track each of the fragments that were generated. During the last stage of the DNA journey through the laboratory, it is run on the DNA Sequencers where it is assigned a filename. Looking at the filename will help guide us through the rest of the laboratory processes

aaa0001_H12_x1_P24_096.ab1

- aaa – this “library code” represents all the fragments from a single shotgunning .sequence. A typical genome will have three such libraries.
- 0001 – This is the “plate number” – these fragments are grouped into batches of 96. 30,000 fragments would result in more than 300 such batches.
- H12 – this represents the coordinates within a 96-well plate in which the fragment resides. The plates are organized in 8 rows (A-H) and 12 columns (1-12).
- x1 – x represents the direction in which this fragment was read, this will be explained later. 1 represents the version of this filename; often the same fragment of DNA may need to be read multiple times.
- P24 – this is its coordinate in the 384-well system. Each group of 96 is read twice resulting in 192 files. Two such groups would result in 384 files. Since the instruments can handle 384-well plates, a pair of these groups is spread into different quadrants of the plate.
- 096 – There are 96 capillaries on the ABI 3730XL sequencing instrument, this part of the filename records through which capillary this fragment of DNA traveled.

- .ab1 – designates that this file is a 300Kb analog representation of the fragment of DNA.

Robotics and Lab Procedures

As mentioned previously, shotgun sequencing is effective because it can become a high-throughput process by using robotics and automation. In our lab, we use robots that are able to operate on 96-wells plates. The 'BioMech' and the ABI 3730XL sequencer are two robots that perform the crucial parts of the high-throughput process.

The BioMech manipulates plates by placing them in locations that allow precise liquid handling for the culture of DNA via PC software running Windows NT. The robot is controlled by a program that is fed to it using a standard PC. We use this robot in conjunction with an array of strong magnets. During the processing of the DNA, we insert microscopically sized metallic beads that bind to DNA through DNA conjugate sites. DNA is then positioned in a magnetic field, on an array of 96 magnets, to allow washing, dilution and extraction. By using a proprietary solution of "RNAase water", DNA is then released from the beads and it returns to solution. Again the plate is placed on the magnetic array and the beads move to the bottom. Now when the BioMech pipettes up the liquid, it takes the DNA with it [13]. The DNA is then placed in a new plate and is ready for the sequencer.

The sequencer is a very precise and accurate robot. It uses a very delicate set of 96-capillaries to separate the DNA fragments by length. These capillaries are razor thin and, for our purposes, extend to 50cm in length. As the DNA travels through these capillaries the shortest fragments move fastest while the longer fragments move more slowly. At the end of the capillary is a laser that uses various frequencies of light to fluoresce sample. Since we are interested in the fluorescence of the tags at the end of each fragment of DNA, a CCD light sensor, records total fluorescence for each fluor,

creating a continuous histogram of readings corresponding to each nucleotide in sequential order [9]. At this point, the software will have recorded an analog signal in a file on the attached PC.

These files are now ready to be checked for quality and their information recorded in a database. Manual quality control can be performed by looking at the entire image recorded by the CCD camera inside. This is, however, a very qualitative measure of success and is best for detecting holistic patterns. A more quantitative measure of success is described next.

Data Curation Pipeline: Overview

The Data Curation Pipeline (DCP) is software that is used for quality control for all samples that are passed through the sequencing instruments. Its goals include ensuring that quality remains high throughout the process by analyzing the quality and content of the DNA sequence that is read by the instruments. It is also responsible for organizing and archiving data to allow future retrieval. We will focus on the quality and content analysis goals as well the technologies and tools that are used to achieve them.

The DCP is comprised of five main stages. Chained together into a single pipeline, they analyze each piece of sequence that is produced by the instruments. The stages are: database init, basecalling, vector-screening, contamination-screening, report generation. We will examine the last four stages as the first is mainly a housekeeping task and is specific to the database.

Each stage incorporates a different bioinformatics tool to accomplish its task. Phred is a tool used to convert analog data to digital sequence, Crossmatch is used for vector-screening and Blast is used for contamination checking. The reporting stage is accomplished by querying the statistics recorded in a database. These critical stages are the middle three as they generate the relevant statistics and use the greatest amount of computer resources. To streamline these stages, we employ the construction of a threadpool. This construct is able to parallelize processing to speed the calculations. We will examine each of the stages and then analyze the workings and advantages of using the threadpool.

This piece of software was written in Java and is platform neutral. Although written and compiled on a 32-bit Linux platform running RedHat Server 2.1, it has been

deployed on AIX and SuSe Linux on a 64-bit architecture. The software is written in a modular fashion to allow for flexibility. Stages of the pipeline can be added or removed and a configuration file is used to change parameters for the environment in which it is run. Database access has been abstracted such that both the database and the structure of the database could be changed without the need to modify the code that belongs to the DCP. The software has successfully been used and maintained for two years and continues to serve the facility's needs.

Basecalling and Phred

The process of basecalling is the conversion of analog fluorescence data into a discrete sequence of base pairs. The term is derived from the process of watching analog data stream from the instrument and literally calling a peak in fluorescence as the indication of a particular base pair. Since the entire goal of such a project is to determine the base pair sequence of a genome, basecalling is the most crucial step in the process. However, because we are dealing with analog data and because this data is far from perfect, basecalling is not a solved problem.

The current standard for basecalling is a heuristic that has been coded in the C language and distributed as a program called 'Phred'. Written by Dr. Phil Green, this program has defined the methodology of basecalling for the entire world of sequencing. The algorithm, which we will look at next, is a heuristic for identifying peaks in a 'chromatogram'. As seen in the figure below, a chromatogram is the plotting of fluorescent signal vs. time for four different frequencies of light. Each of the different frequencies represents a different base pair. However, reading these peaks is not always easy. Often at the beginning, the sequencing instruments sensors might be saturated by what is called the "primer blob". This initial blob in the chromatogram represents the unused primer coming through the capillaries first. Towards the end of the chromatogram, we find that we run out of fragments. The resulting empty space is noise. Other aberrations of signal may come from seemingly random effects. However, Phred still attempts to determine which base pair a peak represents and report the confidence per call.

The algorithm behind Phred is straightforward. In the first pass through the data, Phred attempts to guess where peaks may be. It bases these guesses on the knowledge of how fast DNA fragments can move through the capillary. Over time, it can expect to find peaks at certain regular intervals; and Phred is able to determine many of the base pairs from this first pass. However, it makes a second pass looking for aberrant peaks. These peaks may, for some unknown reason, move too slowly or too quickly. These peaks are given a less confident score [5].

The scoring is based on an exponential error rate. For example, a “Phred score” of 20 represents an expected error rate of 1/10,000. Increasing the score by 10 indicates an error rate of 1/100,000. Like the human genome project, we use a Phred score of twenty as our minimum score [6].

Since the invention of Phred, a variety of base callers have attempted to improve upon Phred’s score. There is little literature that suggests these other base callers are significantly better. However, in high-throughput scenarios, seemingly marginal improvements can develop into significant gains and reductions of cost. For example, recently, ABI introduced a new basecaller that has been demonstrated to be marginally better. With one of our projects we found it to give much better coverage overall, even through the per-read increase in base pairs was not significant. One must always take these improvements with a grain of salt, as there is of course only so much data that can be gathered from a chromatogram.

Crossmatch and Smith Waterman

Crossmatch is a program that utilizes of the Smith Waterman alignment algorithm. The SW alignment algorithm is an exhaustive, optimal alignment algorithm that runs in $O(N^2)$ running time when implemented with all optimizations. A running time of $O(N^2)$ indicates that as the size of the problem doubles, the running time is squared; for the SW algorithm, the size of the problem is the sequence size. The algorithm makes use of dynamic programming to find the best alignment from a given pair of sequences.

Dynamic programming is the idea of finding all possible solutions for some optimization problem by creating a matrix where each cell's value is determined from the values of the cells above, and of those to the left. This methodology allows one to calculate each value quickly and methodically. The trade-off is that you will be required to keep all of these solutions in memory simultaneously [4]. In the case of sequencing alignment, each row can represent the base pairs of the first sequence, while the columns represent the base pairs of the second sequence. If you were to start at the upper right corner of this matrix and moves to the bottom you would pass through different alignments of these two sequences. If one moves down, or to the left, then the alignment is broken or a gap is introduced. Depending on the scoring rubric, this can either negate the entire alignment or simply cause a penalty score. If one moves diagonally down, the alignment continues unbroken. When we reach the bottom, we simply find the path through the matrix that gives us the best score. This path represents the base pairs that should align that will give us the optimal alignment [7].

Since the algorithm is exhaustive and optimal, it makes Crossmatch a very precise alignment tool. However, it does make it computationally expensive to use. Each time the sequence size doubles, the running time quadruples. However, for our purposes, the SW algorithm is perfect. Crossmatch is designed to look at 1000 base pair sequences (the actual reads) and ensure that none of them match the sequence of the vector. By doing so, we are ensuring that we successfully inserted a fragment of the genome into the vector. Failure to do so would be serious problem as it would mean we were sequencing the known sequence of the vector; an expensive, fruitless exercise.

Crossmatch is also a well-suited tool because it has added features of vector masking. When it finds a match between the vector and the sequenced DNA, it identifies vectors sequence with an 'X'. In this manner if one were to scan sequence manually, it becomes readily apparent how much of the sequence was vector [11].

Blast and Contamination Screening

The Blast algorithm uses a method of indexing that allows the quick scan of large sequences [1]. It is a heuristic algorithm and therefore suffers from the pitfalls of false positives and false negatives; however, for our purposes, it works well. In this stage of the pipeline we ensure that the DNA we are sequencing has not been contaminated by foreign sources. First let us examine the Blast algorithm.

The Blast algorithm's main technique for speeding up searching is the use of indexing. Suppose we plan to be looking for matches in the E. coli genome. We would first prepare an index for the E. coli genome. By breaking up this genome into small "words" we can make a map of the sequence. Suppose we have the following sequence:

seqA: AAAATTTTGGGGTTTTAAAAGGGGCCCC

And suppose we want to check if the following sequence matches some where within the previous sequence:

seqB: GGGGC

If we were using the Blast algorithm we would have broken up seqA into four letter words and kept track of where in seqA they occur. Then when we need to ask where in seqA does seqB occur, we can see that the word GGGG occurs in two places in. To determine which of the two locations is the correct one, the Blast algorithm then starts to extend the match beyond the word. In this example, we quickly realize that only one of the locations in seqA contains a C to the right of the word: GGGG . In this manner, indexing makes searching large portions of sequence computationally inexpensive [7].

For our purposes, we want to make sure that the DNA that we are sequencing has not been contaminated by foreign sources such E. coli. Remember that the vector and the

fragment were amplified in an *E. coli* cell. Thus, it is possible for the *E. coli*'s DNA to have become mixed in with the true sample. Other sources for contamination may also include mitochondrial DNA. However, when sequencing a prokaryotic microbial genome from a similar species, we may find false positives when searching *E. coli* because their sequence may actually be similar. These cases warrant special attention should they occur.

Reporting

After the data has been processed through the preceding stages, the results must be reported. The results have been stored in a database that maintains records of relevant statistics per read. Among them are the following:

- QLen – longest stretch of base pairs that are considered to be high quality
- Q20 – number of base pairs that have a quality level of ‘20’
- LSV – lost to sequencing vector – if a read is only sequencing vector
- LCV – lost to cloning vector – if a vector incorporated its host clone’s DNA
- E. coli – if the sequence is similar to some form of bacteria
- Success – if the read had at least 100 base pairs of quality sequence

Each of these factors is relevant in determining if the process is running efficiently and to help troubleshoot any problems. QLen and Q20 scores show the overall quality. These numbers fluctuate with the type of DNA that is being sequenced. When sequencing a microbial genome, it is expected for these scores to approach 900bp on average. It is possible to generate sequence with high quality sequence of un-important vector sequence. The LSV and LCV percentages help show at which stage of DNA preparation the problem is occurring. The E. coli percentage represents sequence that has been contaminated by E. coli. The Success percentage is a threshold value used to measure a minimum level of success that all DNA reads should follow. However, for high-throughput sequencing of a microbial genome, the QLen and Q20 scores are the most important factors for assessing overall quality [6].

The DCP compiles these statistics for each group of 96 reads that were run on the instrument, for each instrument, and for each different library code. These reports are then combined and sent out as an HTML-encoded report.

Threadpool

When processing large amounts of data, patterns in computation become evident. In our case, it is plainly obvious that we are repeating the same exact series of processes on each sequence read. Because of this parallelism, we can easily take advantage of the multiple processors in our computer by using a threadpool.

A threadpool is a collection of threads that will scale with the number of processors available to the pool. The size of the pool should be much smaller than the actual number of computations that need to be performed. In other words, if the number of threads and the number of computations are equivalent, then either the pool is too large or the construction of a threadpool is unnecessary. In our case a pool size of twice the number of processors in the computer was used. This ratio was empirically the fastest. Data of these experiments is, unfortunately, unavailable. Qualitatively, an improvement of approximately 40% was achieved by increasing the number of threads.

The goal of a threadpool is simple: to ensure that no processor sits idle and that no processor is overwhelmed by a multitude of processes. If successful, the threadpool will be able to continually feed a processor a single calculation and upon its completion, feed the next calculation. In this manner, each processor can focus its cycles on a single computation, yet never go unused.

Each thread in the pool is responsible for three jobs. The first is to get the next calculation in the queue. Next, the thread must start the calculation. Finally the thread must monitor the progress; when finished the thread will then begin at step one. In our case, the different calculations might be to run a read through Phred and obtain the sequence; or it might be to run that sequence through Crossmatch to check it for vector

sequence. We are processing thousands of reads per batch; therefore the use of a threadpool is appropriate.

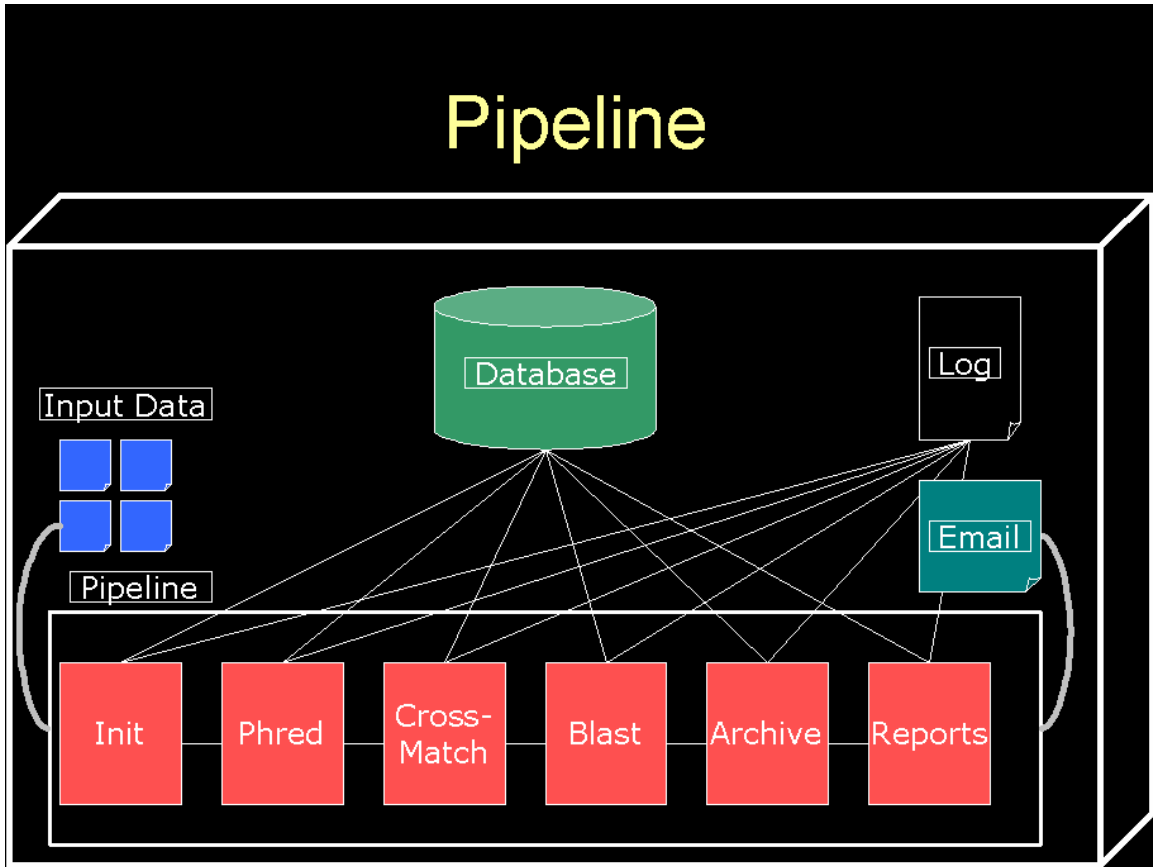
The DCP was developed on a 4-processor 32-bit computer running Linux. The pool size was optimized for this machine at eight. When monitoring processes, one can see that no processor sits idle through the run. However, we have recently moved the DCP to an 8-processor 64-bit computer; here processors are not fully utilized. This indicates that the pool size can grow and that we can thereby process the data faster.

Connectionpool

Much like the threadpool, we use a pool to manage the connection to the database. On the first iteration of the DCP, we opened a new connection to the database each time an update was performed. This was an inefficient practice but it did not significantly interfere with our processing times. However, moving to a different environment, our new database does not allow 100's of connections to be left open. We implemented a connectionpool to allow a connection to be used. This is completely transparent to the DCP as all code regarding database connections resides in a different module. This is a feature of object-oriented programming.

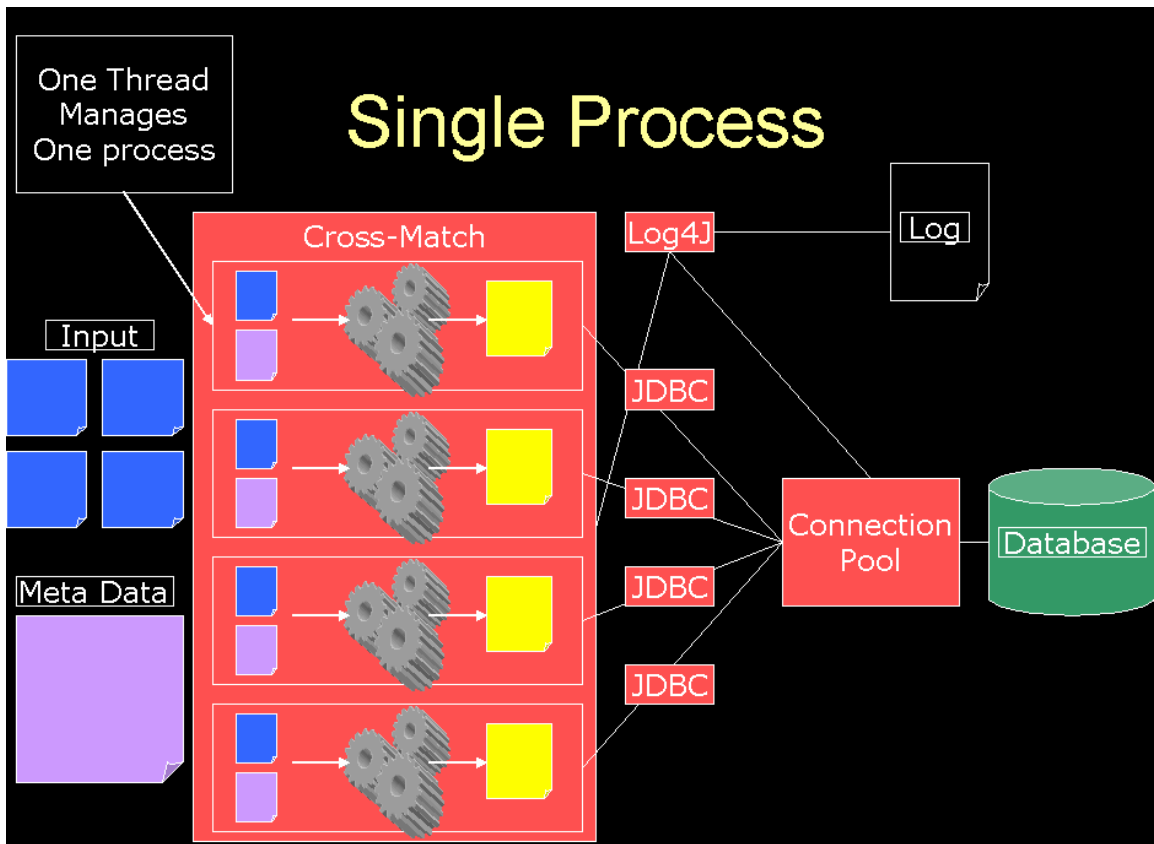
DCP Schematics

Below are visual schematics of the DCP. The first shows the entire DCP and how each stage is fed data, from left to right and how each stage communicates its results to the databases and updates the log. The Email report is generated in the final stage and shown as an output of the pipeline.



Below is the more detailed view of a single stage. In the schematics, we can see that input comes in the form of both actual input data (DNA sequence) and meta-data (contextual information such as where in the database to store the information). The graphics of gears represent the actual computation carried out by the Crossmatch program. It can be observed that each instance of the program is encapsulated within a thread. Finally, we can see that although each thread requests its own connection, via

JDBC, to the database, a connection pool is responsible for managing a limited amount of connections to ensure that the database is not overloaded.



Genomic Assembly

As described in the shotgun sequencing setup, these reads, or genome fragments, must be reconstituted into a single string of genomic sequence. To do so is akin to a jigsaw puzzle. Each piece belongs in the final puzzle, and the clues to its location can be inferred from its edges. When two fragments seem to have similar sequence, most likely they were from overlapping segments of the original genome.

Although the problem can be stated simply, it is not a simple problem. For one, the puzzle is quite large. Even keeping all the pieces of the puzzle in memory is a challenge. Depending on the assembler that is chosen, different kinds of computers can be used. The simplest to use is called Phrap, part of the Phred/Phrap/Consed package [8] [11]. Because a publication is not available we are not sure exactly why Phrap is limited by 32-bit computing resources. My theory, is that a 32-bit computer can, at most, allocate a single block of 2^{32} bytes or 4 GB of RAM. This means that even though our original server has 8 GB of RAM, only 4 GB could be used by Phrap. Having access to 64-bit computing environments was necessary for our microbial genomes. In a 64-bit environment, the computer can address 2^{64} bytes of RAM. Phrap allocates somewhere between 6-10 GB of RAM for assemblies. Although other assemblers use as much or maybe greater amounts of memory, they are able to operate within the confines of a 32-bit environment because they do not allocate a single, massive, contiguous block of memory. Rather their data structures break up the memory into smaller chunks [2].

The next issue in dealing with such a large problem is that finding neighbors is complicated. Comparing two sequences is not an insignificant computation. Although a single alignment may take less than a fraction of a second, when performing tens of

thousands of alignments, processing times will increase. One technique for dealing with this is rather than doing alignments in the Smith-Waterman style, is to use the Blast technique. By indexing words from each segment it is easier to find matches.

Another useful method of finding where pieces fit in the puzzle stems from the knowledge of the shotgun sequencing process. We know that each read has a mate-pair. In other words, because each clone was read from both the forward and reverse direction, we know that the distance between them is some known size. If we find two neighbors, we know that each of their mate-pairs might also be neighbors. In this manner, not only can we find neighbors more quickly, we can also test the validity of reads determined to be neighbors. Suppose we know that the reads in a single mate pair have been placed 10,000 base pairs from each other. If they are the only ones in that region, this may be a simple misplacement of that read pair. However, if there are a group of mate-pairs that have been separated by unreasonably large sizes, we know that there is bigger problem. Such a situation maybe indicative of the existence of a repetitive region of the genome. Fixing these types of errors will be covered in the 'Finishing' section.

Although Phrap does not use this mate-pair information in putting the puzzle together, more advanced assemblers, such as Arachne, are able to do so [2]. This feature is critical for larger genomes where repetition can be ubiquitous. Using mate-pair information in building the puzzle means that using clones-inserts of different sizes can be particularly advantageous. Varying sizes of inserts means that an advanced assembler can place reads more accurately. It can use larger inserts as 'scaffolding' on which to build the rest of the assembly. It is akin to building a fence with guideposts marking the path. Without the guideposts, it becomes more difficult to build a straight fence.

Finishing

Finishing is the last stage in the process of sequencing a microbial genome. Until now, we have been reading fragments of the genome in a random fashion. By reading fragments with a random distribution across the genome, we are able to very quickly read the vast majority of the genome. However, due to the size of the problem, and extent to which we sequence, there is statistical likelihood that we will miss certain regions [15]. These regions are known as gaps and must be “finished”.

Not only have the statistics of the algorithm inferred that we may have gaps. It is very likely that parts of the genome may contain a greater proportion of certain base pairs. For example, often a G/C rich region occurs in long stretches. This means in these regions, sequencing reactions may yield lower quality reads because of chemical reasons. Normally, there are an equal number of fluorescently labeled base pairs in a sequencing reaction's solution. However, for G/C rich regions, the G's and C's will be consumed more quickly and the reaction will essentially cease. To remedy such situations, one must include additional G's and C's in the solutions. This tends to compensate for the highly concentrated G/C region of the genome. Learning where these regions may exist is part of the finishing process.

Repetitive regions of the genome can also cause great confusion during the finishing process. A first round assembly may have been linked together disparate regions of the genome because it thought that they matched due to sequence similarity. Such situation can hopefully be remedied by clone-inserts that are greater than the size of the repeat. If such insert sizes are used, then they would be able to span across the gap and include regions of sequence that are unique to that location.

Solving for the simplest gaps is straight forward. However, it is by no means high-throughput. Simple gaps are those that can be closed by sequencing across the middle region of a fragment. When a fragment is inserted, we know that sequence is read from both of its ends. The middle region is left unread. Often, a mate-pair will land on the end of neighboring contigs. From this we can infer that these contigs are indeed neighboring segments of the genome if we knew the sequence of the fragments. Primers can be designed, based on the known sequence, to allow sequencing through the unknown portions of the fragment. When complete, reassembling the genome should show that these gaps have indeed filled and the contigs should merge.

The application of choice for finishing is called Consed; which is, again, part of the Phred/Phrap/Consed package. Consed is a graphical tool for viewing assemblies and aids in finding mis-assemblies, measuring progress and manually editing basecalls to allow more accurate assemblies. Consed's assemblyview feature is extremely useful in prioritizing finishing efforts. Often, it is desirable to work on simple gaps first or to work to bring together the largest portions of the genome. Assemblyview helps quickly determine which segments of the genome to prioritize as well as presenting an overall picture of the status of a genome. Assemblyview also aids in finding mis-assemblies by highlighting mate-pairs that have stretched too far across the genome.

Arguably the most powerful feature of Consed is called Autofinish. Autofinish is a component that designs and organizes finishing experiments. This includes designing primers and measuring the success of previous experiments. Because Autofinish provides a systematic approach to finishing, it aids in making this process less error-prone than if done by hand [8].

Another tool used in finishing is called Orchid. Orchid is primarily a visualization tool to allow one to better understand the state of an assembly. It features a circular view of the project which is particularly useful for microbial genomes. In general, having the entire project represented in a single screen gives a better understanding of the state of the project. Compared to Consed's Assemblyview, Orchid provides a more holistic view. Unfortunately, it is more difficult to configure. Whereas the Phred/Phrap/Consed can be reconfigured by editing a perl script, Orchid requires one to modify and recompile C++ code. It should be noted that these are one-time changes. However, clearly, it would be desirable for this type of configuration to be part of a configuration file of some sort instead of hard coded into the applications' source code [10].

As mentioned before, finishing is not a high-throughput process. Often numerous rounds of finishing are required to reach the end goal of a complete genome. Often, it takes the most time and incurs the greatest cost. The effort and cost of this process, alone, provides incentive to improve technologies upstream. Improvements in assemblers and basecaller would provide more accurate sequence and assemblies and reduce the resources spent on finishing.

Resequencing and Resequencer

Appendix I is a manuscript that will be submitted to the Bioinformatics journal once the appropriate data is collected. It details the features and usage of a graphical application designed to efficiently design primers for resequencing projects.

Resequencing is the idea of sequencing a particular portion of the genome multiple times. Resequencing is necessary for a variety of reasons. For microbial genomes, this is important for verification purposes. In analyzing a completed genome, scientists are interested in the differences between homologous genes. Since these differences are often on the per-nucleotide level, it is important to confirm the DNA sequence. This process is much like the finishing process in that special primers need to be designed in order to sequence the appropriate portions of the genome. Resequencer helps to automate and organize this process by automatically retrieving data and exporting primers in a systematic fashion. Please view the manuscript for details.

Conclusion

The process of sequencing a microbial genome is well-understood process but not without its challenges. Without a turnkey solution, an efficiently organized and executed pipeline is the best way to reach the goal. By using robotics for high-throughput preparation of the lab, combined with comprehensive quality control tools in software, the process is greatly quickened. The systematic use of naming conventions helps keep the low-throughput portions of the project. I have had the privilege to help build parts of this pipeline and appreciate the amount of work that is required to produce a complete sequence.

After a complete sequence is created, the next steps that are taken are very bioinformatics intensive. The first task would be to annotate as much of the genome as possible. Beyond this, one can assess the differences between functionality, look for SNPs, or measure evolutionary patterns. These tasks, again, have been done before, but are by no means completely automated. The future of DNA Sequencing is very bright. With new technologies on the horizon, this type of capillary sequencing instruments may become obsolete, however the processes involved and the principles will remain the same.

References

1. Altschul, S.F., W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. 1990. Basic local alignment search tool. *J. Mol. Biol.* **215**: 403-410
2. Batzoglou S. ARACHNE: Whole-genome shotgun assembler. *Genome*. 2002.
3. Bouck J, Miller W, Gorrell JH, Muzny D, Gibbs RA. Analysis of the quality and utility of random shotgun sequencing at low redundancies. *Genome Res.* 1998 Oct;8(10):1074-84.
4. Corman T. H., Leiserson C. E., Rivest R. L. Introduction to Algorithms. The MIT Press/McGraw-Hill, 1990.
5. Ewing, B., L. Hillier, M.C. Wendl, and P. Green. 1998. Base-calling of automated sequencer traces using phred. I. Accuracy assessment. *Genome Res.* **8**: 175-185
6. Ewing, B. and P. Green. 1998. Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Res.* **8**: 186-194
7. Mount D. Bioinformatics: Sequence and Genome Analysis. 2001.
8. Gordon D, Abajian C, Green P. Consed: a graphical tool for sequence finishing. *Genome Res.* 1998 Mar;8(3):195-202.
9. Huang XC, Quesada MA, Mathies RA. DNA sequencing using capillary array electrophoresis. *Anal Chem.* 1992 Sep 15;64(18):2149-54.
10. Orchid - <http://www-shgc.stanford.edu/informatics/orchid.html>
11. Phrap – <http://phrap.org> (no publication available)
12. Smith T.F. and Waterman M.S. Identification of common molecular subsequences. *J. Mol. Bio* 1981. 147, 195-197.

13. Wahlberg J, Holmberg A, Bergh S, Hultman T, Uhlen M. Automated magnetic preparation of DNA templates for solid phase sequencing. *Electrophoresis*. 1992 Aug;13(8):547-51.
14. Weber JL, Myers EW. Human whole-genome shotgun sequencing. *Genome Res*. 1997 May;7(5):401-9.
15. Wendl MC, Yang SP. Gap statistics for whole genome shotgun DNA sequencing projects. *Bioinformatics*. 2004 Jul 10;20(10):1527-34. Epub 2004 Feb 12.

Appendix I

-- Please See Attached --