CSC 2427 Project: Genome Assembly Algorithms for New Sequencing Technologies

Alexander Hertel & Philipp Hertel

May 4, 2006

Contents

1 Background

A sample of DNA is said to be sequenced if we can produce a string of symbols A, C, G, T which accurately reflects the order of the nucleotides in the sample. For short snippets of DNA (≤ 500 nucleotides) this problem is easy since there exist 'DNA Sequencers' which can identify the nucleotides in the sequence based on their chemical properties. The problem is much harder when the DNA sample is significantly longer, because we then face the problem of having to assemble the DNA fragments. For example an entire genome can be *billions* of base pairs (b.p.) in length, and represents a colossal assembly problem.

Since genomes can potentially tell us a great deal about the organisms from which they are taken, it is desirable to have a reliable method for sequencing them. As one might expect, there has been much work done in the area of whole genome sequencing and different approaches exist. Since it is currently infeasible to directly read sequences of more than about 500 nucleotides, these methods universally include a step which decomposes the genome into thousands of short pieces (called reads) followed by subsequent steps which reassemble the pieces once they have been directly sequenced. Whole genome sequencing is therefore also referred to as 'whole genome assembly'. Methods differ both in how they decompose the genome and how they reassemble the sequenced reads.

The two main approaches to sequencing the human genome were pioneered by the publicly funded international Human Genome Project (HGP) [Con01] as well as by Celera Genomics [VAM01], a private company based in the United States. The public project's strategy is called the 'BAC-by-BAC Technique', and Celera's technique is called 'Whole Genome Shotgun Sequencing'.

1.1 The BAC-by-BAC Technique

Also called the 'Hierarchical Shotgun Approach', the BAC-by-BAC method gets its name from BACs, or Bacterial Artificial Chromosomes. BACs are segments of human DNA which have been artificially replicated by inserting them into the circular genome of e. coli bacteria, and allowing it to divide, thereby also creating copies of the human DNA, which are later extracted.

The BAC-by-BAC technique works as follows:

1. Make many copies of the human genome to be sequenced.

2. Take a number of copies of the human genome, and split them up into much smaller strands of DNA, each consisting of about 150 000 b.p.. Each of these strands is called a BAC, since the BAC technique will be used to duplicate them. The high-level idea is that we will sequence the BACs, and then use overlaps between the BACs as well as the locations of known genes to put all of those sequences together, and thereby obtain a sequencing of the entire human genome. We therefore must make sure that all of the BACs together not only contain the entire human genome, but that the overlaps are sufficiently large for us to be able to reliably put them together again.

3. Before we are able to assemble the BACs we must first sequence all of them. Sequencing the BACs is done analogously to sequencing the genome: we take each BAC, make copies of it using the e. coli bacteria technique, and then randomly cut them up into small pieces which are approximately 4000 b.p. long. We take these and use our automated sequencing technology to sequence approximately 500 b.p. at each end. Each of these 500 b.p. sequences is called a 'read', and it tells us not only what is on one strand of the DNA, but by complementary base pairing, it immediately gives us the other strand as well. Since the reads are only 500 b.p. long at each end, we are left with a gap which is about 3000 b.p. long for each fragment. It is also normal to use longer 40 000 b.p. pieces with a much larger gap. The gap information will be useful later when putting the reads back together. One important note is that each read is done from the $5'$ to the $3'$ end, so the two reads from each fragment are from opposite strands, and opposite ends of the fragment. A fundamental difficulty is that we do not know which strand each read is from, just that certain reads are from opposite strands. With enough overlapping reads, it becomes possible to put them together in order to

sequence each BAC. This is done by building a bidirected graph in which the nodes contain the read information, and then finding a Hamiltonian Path (or something very close) in it. For more information on this step, please refer to [Her06].

4. Once we have sequenced the BACs, we have to find out where they came from in the genome, and then choose the smallest set of BACs that completely covers the entire genome. It is easy to see that if we put the BACs together correctly, then the fact that we have sequenced them will give us a correct sequencing of the entire human genome. The process of 'tiling' the genome in this way is very expensive in human time, and in practice involves physically mapping the BACs to a copy of the complete genome by using the locations of known genes.

1.2 Whole Genome Shotgun Assembly

In [WM97], Weber and Myers developed a technique which can be automated to a greater degree than the BAC-by-BAC method. They teamed up with Celera Genomics and its founder Craig Venter to implement their new technique. Celera's idea was to cut out the intermediate step of having to sequence the BACs; instead, they split the original DNA sequence into 40 000 and 4000 b.p. segments directly (these are called cosmids and plasmids, respectively), and from there to read them and then assemble the reads for the whole genome. Since the splitting of the genome into the shorter pieces is relatively random, ensuring that the entire genome gets covered requires the sequencing of base pairs approximately ten times the size of the genome, or $3 \cdot 10^{10}$ b.p.. This translates to approximately $6 \cdot 10^7$ reads.

This entire process is called 'whole genome shotgun sequencing', since the process of fragmenting DNA strands is reminiscent of the tiny shot pellets coming out of a shotgun. With a sufficient number of reads, it is not hard to see that the entire problem of sequencing the human genome is just a (much) larger version of the problem of sequencing a BAC.

2 Project Motivation

Although Celera's methodology is the current state-of-the art, there are certainly improvements that can be made in the sequencing of the human genome. In particular, there are two obvious areas where it can be improved. These areas form the motivation of our project.

2.1 Lowering Cost

The first area where genome sequencing can be improved is by lowering the cost, both in terms of time and in terms of dollars. Both Celera's technique and the HGP's technique are enormously expensive, and incredibly slow. The goal within the foreseeable future is to bring the cost of sequencing a genome down to under $10,000, and to bring the time required to under a day. The current technology is not anywhere even close to this goal, suggesting that enormous improvements can be made.

2.2 Improving Confidence

Although both results were widely lauded by the media as well as the general scientific community, there have also been many legitimate criticisms of their methodology.

The literature contains two academic (rather than financial) criticisms of the two sequencing projects. The first criticism is that Celera's technique is over hyped, much of their data was 'borrowed' from the HGP, that Whole Genome Shotgun Assembly is not cheaper or faster than the BAC-by-BAC technique, and that a number of myths have evolved concerning Celera's results. Many such criticisms are leveled by Green in [Gre02]. In fairness to Celera, many of these criticisms involve secondary issues which would be considered superficial by theoretical computer scientists who are only interested in how accurate the results are rather than whether they benefitted from the HGP's work, etc..

The second type of criticism is more serious [Gre97]. One of the major flaws with both the HGP as well as Celera's techniques is that their algorithms and methodology are reliant on ad hoc heuristics and are not solidly grounded in theoretical computer science, and that there are no guarantees or theoretical bounds known which allow us to have confidence in the results. This criticism is perhaps best expressed in [WLS02]: nobody is saying that there definitely are serious flaws with mankind's attempts to sequence the human genome, but rather that there are no guarantees that the results are correct.

If runtime and confidence bounds could be rigorously proven for a sequencing algorithm, that would certainly be an improvement.

2.3 Future Technology

The two goals of lowering the cost of sequencing in terms of dollars and time, as well as improving the confidence we have in assembly algorithms are the motivation for our project. Given the current sequencing technology, it would be difficult to do much better than Celera or the HGP. However, as this technology improves, it will be possible to implement assembly algorithms which meet both of these goals. However, we do not need to wait for this sequencing technology to be invented before designing appropriate algorithms which utilizes it; as is so often the case in computer science, we can design the algorithms so that they are ready and waiting well before they can be implemented.

It is impossible to predict what kinds of sequencing technologies will be invented in the future, but the most promising one on the horizon seems to be 'nanopore technology'. The two human genome sequencing projects relied on sequencing technology that is very accurate, but only for DNA segments that are approximately 500 b.p. long. Recent research into nanopores at Harvard [CGB+04] gives researchers hope that within a few years, we will have sequencing technology capable of reading DNA segments containing approximately $100\ 000 - 150\ 000$ b.p., albeit with an error rate of about 30%. More information on nanopores can be found below in Section 3.

The ultimate goal for sequencing technology is to build sequencers that can accurately read billions of base pairs in an entire genome just like an entire piece of cloth being fed through a sewing machine. Such technology would clearly make assembly algorithms unnecessary, but this seems very far off. In the foreseeable future, it is reasonable to assume that sequencers will continue to be constrained by the limitation of only being able to read DNA fragments which are considerably smaller than the size of a entire genome, and that assembly algorithms will be necessary for a number of years.

Genome assembly algorithms (or whether they are even necessary) invariably reflect the sequencing technology from which they get their inputs. Just as Celera and the HGP's algorithms reflect the current state of the art in sequencing technology, the aim of this project is to design algorithms which reflect near future sequencing technology which is capable of obtaining much longer reads with a high error rate.

3 Project Overview

In this section we give a high-level overview of our project. We start by providing a more detailed account of nanopore technology, and explain how it can be used in a two-stage assembly algorithm. The first stage is the building of what we call a 'scaffold'. The second stage is the correction of the errors in the scaffold.

3.1 Nanopore Technology

Nanopores are a new technology being developed at Harvard for sequencing DNA. A nanopore is a tiny 2-4 nm hole in a silicon nitride membrane, as shown below in Figure 1.

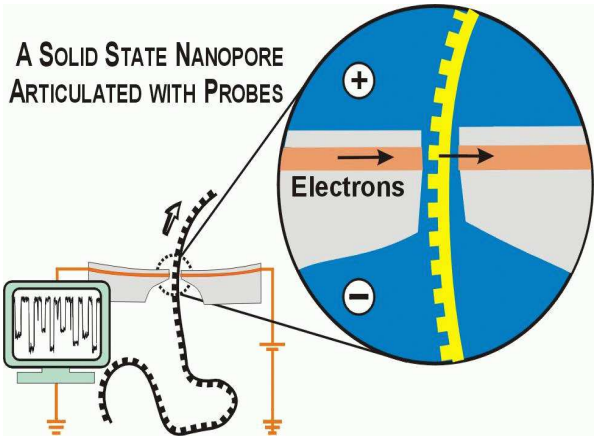

Figure 1: A nanopore probe reading a string of DNA. *http://www.mcb.harvard.edu/branton/*

The idea is to take fragments of DNA and place them in a solution such that the strands are on one side of the membrane. The opposite side of the membrane is positively charged, and therefore attracts the negatively charged DNA strands via electrophoresis, which then travel through the nanopore. A second voltage across the nanopore allows us to read the bases as they pass through. This process allows us to sequence DNA fragments which are much longer than the current state of the art allows, but it also has a much higher error rate. As already stated in the previous section, researchers hope that within a few years we will have sequencing technology capable of reading DNA fragments containing approximately $100\ 000 - 150\ 000$ b.p., with an error rate of approximately 30%, where errors include substitutions, omissions, and insertions.

For more information on nanopores, please refer to [Har]and [CGB+04].

3.2 Scaffolds

The key idea behind this project is the use of a 'scaffold'. A scaffold is an entire sequenced genome, but with up to 30% error. Once we have such a scaffold, we can then use it to get an accurate sequencing of the entire genome by removing its errors. In effect, the scaffold-based assembly algorithms that we propose consist of two parts: the first part consists of building the scaffold, and the second consists of removing the scaffold errors.

3.3 Scaffold Building Algorithm

Under the assumption that nanopore technology is mature enough so that we are able to efficiently and cheaply read DNA segments containing $100\ 000 - 150\ 000$ b.p. with a 30% error rate, building a scaffold can be done considerably faster and more cheaply than the current costs in terms of time and money of sequencing genomes. Simply use the standard graph-theoretic approach that is used in shotgun sequencing: obtain a sufficient number of error-prone reads from the nanopores in order to completely cover the genome, create nodes containing these reads, and build the standard bidirected graph as described in Section 1. Finally, find a Hamiltonian Path (or something very close to it) in this graph using the same techniques that were used by the HGP and by Celera. Of course, when building the bidirected graph, instead of putting in edges for reads which overlap almost perfectly, we must now use a more liberal definition of overlap which includes the potential 30% error on each end of the read.

Although the techniques being used are almost identical to those used in the HGP and by Celera, building the scaffold would be considerably cheaper. This is for two reasons. Firstly, the reads given to us by nanopores are twenty to thirty times larger than those given to us by the current state of the

art sequencing hardware. This means that the bidirected graph that we build will have less than one twentieth as many nodes. But secondly, since the reads are so much longer, we can demand a much larger overlap, and thereby reduce the number of 'false edges' that we include in our graph. In effect, we will need fewer reads to build a much smaller graph that contains fewer erroneous edges, which in turn means that the bidirected graph will be much easier to solve. Assuming that nanopore reads are about as expensive as the current reads being used, this translates into an assembly problem which can be solved much more cheaply and quickly, and with less computer power than before.

After this process is complete, we are left with the entire scaffold genome.

3.4 Consensus Algorithms

Once we have a scaffold, we enter the second phase of our algorithm, the removal of errors. In order to remove these errors, we will describe two different algorithms called 'consensus algorithms'. These are described in below Section 4.

3.5 Error Model

For the purposes of this project we have used an error model which assumes that errors are uniformly distributed and independent within our scaffold. It may be the case that nanopore errors are not actually independent of one another. Errors may come as quite large sequences of mistakes, rather than isolated substitutions. We feel justified in overlooking this problem since we are assuming that our scaffolds are constructed of numerous nanopore reads. Even if the errors within a single such read are not totally independent, the position of and lengths of the sequences of errors across all the reads will be roughly independent and uniformly distributed. Essentially, the errors may be clustered, but they will be locally clustered and these local clusters will be spread uniformly across the reconstructed scaffold. This seems like a reasonable conclusion if we assume that we will be using shotgun assembly to build the scaffold from the nanopore reads in the first place.

We have also not included insertions or deletions in our error model. There are numerous examples in computational biology of simple models which can be extended relatively easily to include insertions and deletions, and we feel that this is true of our model. Our model can be easily generalized to include deletions and insertions by using Needleman-Wunsch instead of simple discrepancy counting to score how well a read maps to a scaffold. This could be a potential avenue for future development of this project.

4 Consensus Algorithm Details

Our consensus algorithms are so called because they work to remove the errors in our scaffold by mapping reads to the scaffold until each index has a sufficiently high number of reads. We then take the majority, or consensus, of the reads at each index. In this way, we use the power of probability to wash out the error.

Below, we describe and analyze two such probabilistic algorithms, the 'Long' and 'Short' consensus algorithms. In order to analyze the algorithms we require formal definitions of the main objects involved.

Definition 1. *A **genome** G of length n is a random String of n symbols taken from $\{A, C, G, T\}$.*

Definition 2. *Let G be a genome of length n. A δ-**scaffold** S for G is a string of n symbols taken from $\{A, T, C, G\}$, in which each position i in S has a probability of δ of differing from the i^{th} position of G.*

Definition 3. *Let G be a genome of length n. A k-**read** (or simply read) from G is a length k substring of G chosen uniformly at random.*

These definitions assume that genomes act like random strings. Though this is clearly a simplification, it is not an unreasonable model to use given the probabilistic nature of our algorithm. Our experimental results for the short consensus algorithm in Section 6 correspond very closely to its analysis using this model. More complicated models for genomes could most likely be readily incorporated into the analysis.

4.1 Long Consensus Algorithm

The Long Consensus algorithm is the most simple. If we can build scaffolds very quickly and very cheaply, then the obvious thing to do is to build many independent scaffolds, align them, and for each index in the genome take the consensus (majority) of the base in that location to be the correct answer. This is where the Long Consensus algorithm gets its name: the alignment is done on strings which are long with respect to the size of the genome.

This algorithm can be easily implemented by keeping four parallel arrays in memory (one for each base) that are as long as the scaffolds. Initialize them all to zero, and for every scaffold read, increment the appropriate parallel array at that index. Finally, once all of the scaffolds have been processed, simply take the consensus at each index, and output that as the genome. In the case of ties, simply output one of the arbitrary bases which tied.

4.1.1 Long Consensus Algorithm Analysis

The Long Consensus algorithm can be analyzed to give us very good guarantees of accuracy. In fact, the analysis shows that one does not need very many scaffolds in order to obtain very accurate results.

Theorem 4. *The long consensus algorithm requires* $m = \left\lceil \frac{2\log\epsilon}{\log\delta} \right\rceil$ *δ-scaffolds to generate a consensus genome with expected accuracy* $\geq 1-\epsilon$.

Proof: We prove an upper bound on the coverage required for the long consensus algorithm to produce a genome that is $(1-\epsilon)\times 100\%$ correct. We do so by bounding the coverage required at a single position of the consensus to ensure that the position has a probability of $1-\epsilon$ of being correct. Assuming that the error at each position is independent we immediately expect that $(1-\epsilon)n$ positions of the genome will be correct.

The consensus genome is comprised of m δ-scaffolds over the alphabet $\{A, C, G, T\}$ each of which has length n. If we stack these scaffolds, we produce an $n \times m$ matrix, X where position $x_{i,j}$ denotes the i^{th} symbol of the j^{th} scaffold. Each column of this matrix produces its most frequent entry as its consensus symbol.

We are interested in fixing a column i of X and calculating how many entries m the column must have in order to ensure that its consensus symbol has probability at most ϵ of being incorrect.

Without loss of generality, suppose that the i^{th} symbol of the true genome is A. For each entry $x_{i,j}$, we define four Bernoulli Random Variables, $x_{i,j} = A$, $x_{i,j} = T$, $x_{i,j} = C$, and $x_{i,j} = G$. Assuming that A can mutate to any of the other three bases with equal likelihood, the variables have the following distributions:

Variable	$Pr(True)$	$Pr(False)$
$x_{i,j} = A$	$1-\delta$	δ
$x_{i,j} = T$	$\delta/3$	$1-\delta/3$
$x_{i,j} = C$	$\delta/3$	$1-\delta/3$
$x_{i,j} = G$	$\delta/3$	$1-\delta/3$

We now define the variables B^i_A, B^i_T, B^i_C, and B^i_G, which will each represent the sum of the occurrences of each symbol in column i of the matrix. Formally, we define B^i_Y for $Y \in \{A, C, G, T\}$ as:

$$B^i_Y = \sum_{j=1}^{m} (x_{i,j} = X)$$

Clearly, the expected value of B^i_A, $Exp(S^i_A)$, is $(1-\delta)m$, while the expected value of each of the other variables is $(\delta/3)m$.

We are now interested in $Pr(B^i_A < B^i_T \vee B^i_A < B^i_C \vee B^i_A < B^i_G)$. We can bound this probability by $Pr(B^i_A \leq 0.5m)$, essentially bounding the probability that the plurality of the symbols in column i are

*A*s by the probability that the majority of the symbols in column i are *A*s. Clearly $Pr(B_A^i \leq 0.5) = Pr(B_T^i + B_C^i + B_G^i \geq 0.5m)$. We can calculate this probability quite easily.

$$Pr(B_T^i + B_C^i + B_G^i \geq 0.5m) \leq \delta^{0.5m}$$

We can then calculate the coverage required for $Pr(B_T^i + B_C^i + B_G^i \geq 0.5m) \leq \epsilon$.

$$\begin{aligned} \epsilon &\geq \delta^{0.5m} \\ \log \epsilon &\geq 0.5m \log \delta \\ m &\geq \left\lceil \frac{2 \log \epsilon}{\log \delta} \right\rceil \end{aligned}$$

□

Corollary 5. *Let* $\delta = 0.3$.

1. *If run on* 4 *independent scaffolds, the Long Consensus algorithm produces a genome which is* $> 99.9\%$ *accurate.*
2. *If run on* 8 *independent scaffolds, the Long Consensus algorithm produces a genome which is* $> 99.99\%$ *accurate.*
3. *If run on* 12 *independent scaffolds, the Long Consensus algorithm produces a genome which is* $> 99.999\%$ *accurate.*

It is therefore easy to see that if scaffold building is fast and cheap, then the Long Consensus algorithm is feasible and gives us very good results. Given m scaffolds of length n, its running time is trivially $O(mn)$.

4.2 Short Consensus Algorithm

If scaffold building is not fast or cheap, then it may take considerable resources to build even one. We know that if nanopore reads are approximately as expensive as the current technology being used, then scaffold building will be considerably faster and cheaper than the current techniques used by Celera and the HGP for genome assembly. Nevertheless, the cost may be too high to be able to build the many copies required by the Long Consensus algorithm. In this case, we can use the 'Short Consensus' algorithm.

This algorithm works as follows: given one scaffold, we correct its errors by mapping random reads to it and using consensus as before at each index. The Short Consensus algorithm is so called because the reads it is mapping are short compared to the scaffold. We can choose to use traditional reads, which are highly accurate but only about 500 b.p. long, or we can use the longer 150 000 b.p. nanopore reads. As with the Long Consensus algorithm, we keep four parallel counter arrays, one for each base, and initialize them to zero. Every time a read is mapped to the scaffold, increment the appropriate indices of the appropriate arrays. Once the scaffold has been sufficiently covered, take the consensus at each index. If there are any indices which have received no coverage, simply output the scaffold's bases at those indices.

Mapping reads to the scaffold can be done very efficiently. When looking at a read and determining where to map it, there is no need to search the entire genome. Instead, we can use hashing like in the BLAST algorithm. The idea is to build a hash table as a preprocessing step, where the hash table keys are all of the possible base sequences of length c in lexicographic order starting with $AAA, ..., AAA$ and ending with $TTT, ..., TTT$ In other words, our hash table contains 4^c indices. For each hash table key, enter all of the scaffold indices for where that key occurs in the scaffold. We sequentially check the hash key starting at each index of the read and its complement against the scaffold until a hit occurs. For each hit we check if the read actually maps to the corresponding scaffold location. In so doing we tolerate an appropriate number of discrepancies between the read and the scaffold location. An upper bound on the number of discrepancies tolerated is given by Lemma 6. If we do not use the BLAST-like hashing, but instead try to match each read to every position in the scaffold, then we call that algorithm the 'brute force short consensus algorithm'. Though this is horribly inefficient in practice, we use it later on in our analysis for clarity.

We could also use the same algorithm with nanopore generated reads. This would yield an expected combined error rate given by the expression in Lemma 9. If we substitute this expression in for δ in Lemma 6, then we get an upper bound on the number of discrepancies between a read and the scaffold.

4.2.1 Short Consensus Algorithm Analysis

The Short Consensus algorithm can also be analyzed to give us very good guarantees of accuracy:

Lemma 6. *Let G be a genome of length n, and let S be a δ-scaffold for G. Consider S_i^k, a length k substring of S beginning at some randomly chosen index i of S. Then the probability that S_i^k contains more than $\delta k + \sqrt{4\delta k \ln n}$ errors, is less than $1/n$.*

Proof: Let $x_i, x_{i+1}, \cdots, x_{i+k-1}$ be Bernoulli random variables, where x_{i+j} is true iff position j of S_i^k is different than $G[i+j]$. By the definition of S, each x_{i+j} has probability δ of being true. We now define the variable B as follows:

$$B = \sum_{j=0}^{k-1} (x_{i+j} = true)$$

Since each x_{i+j} is a Bernoulli Random Variable with probability δ of being true $Exp(B) = \delta k$ and we can use Chernoff bounds to determine how many errors can be introduced into S_i^k before the probability of such a segment occurring in S is less than $1/n$ and therefore effectively 0. The Chernoff Bound for the upper tail of our distribution is:

$$Pr(B \geq (1+t)\delta k) \leq e^{-t^2 \delta k/3}$$

We can then set this probability to $\frac{1}{n^{\frac{\alpha}{3}}}$, for some $\alpha > 3$ and solve for t.

$$\begin{aligned} \frac{1}{n^{\frac{\alpha}{3}}} &\leq e^{-t^2\delta k/3} \\ -\frac{\alpha}{3}\ln n &\leq \frac{-t^2\delta k}{3} \\ t^2 &\geq \frac{\alpha \ln n}{\delta k} \\ t &\geq \sqrt{\frac{\alpha \ln n}{\delta k}} \end{aligned}$$

We now know the following, where $n^{\alpha/3} = n^{1+\xi}$:

$$Pr(B \geq \left(1 + \sqrt{\frac{\alpha \ln n}{\delta k}}\right) \delta k) \leq \frac{1}{n^{1+\xi}}$$

Simplifying we obtain the following, as required.

$$Pr(B \geq \delta k + \sqrt{4\delta k \ln n}) < \frac{1}{n}$$

Theorem 7. *Let G be a genome of length n and let S be a δ-scaffold for G. Let R be a random k-read taken from G starting at randomly chosen position i. If $k - 2(\delta k + \sqrt{4\delta k \ln n}) > 2\log_4 n$, then the short consensus algorithm will map R only to position i of S.*

Proof: To prove this theorem, we need to show that the probability that the short consensus algorithm will map R to any position of S other than i is less than $1/n$. Since there are only n positions in S, this will mean that we can expect the read to be mapped uniquely. Suppose R was originally taken from interval $G[i, i+k-1]$ of G, and should therefore be mapped to $S[i, i+k-1]$.

Let $S[j, j+k-1]$ be an arbitrary length k interval of S such that $j \neq i$. We will show that there is a low probability that the short consensus algorithm maps R to interval $S[j, j+k-1]$. For convenience, let $\chi = \delta k + \sqrt{4\delta k \ln n}$.

By Lemma 6 we know that any interval of S with length k will only differ from its corresponding interval in G in at most χ positions. The short consensus algorithm must therefore tolerate at most χ errors between R and the interval of S that it is trying to map R to before it rejects the interval. This means that R and $S[j, j+k-1]$ can viewed as having χ adversarially chosen positions in common. We also know that $S[j, j+k-1]$ itself can contain another χ errors.

If these errors are chosen to be disjoint from the previous χ 'free spots' and if each error sets its position in $S[j, j+k-1]$ to equal the corresponding position in R, then R and $S[j, j+k-1]$ can be viewed as having 2χ common symbols. At this point any discrepancy between R and $S[j, j+k-1]$ will mean that $S[j, j+k-1]$ must be rejected and we have $k - 2\chi$ position left to compare. The probability of the short consensus algorithm mapping R to $S[j, j+k-1]$ is therefore the probability that these last $k - 2\chi$ are identical. This is the same as trying to find the probability that two strings of length $k - 2\chi$ randomly chosen from G are identical. In other words, the probability is equal to finding a single string of length $k - 2\chi$ twice in G. If $k - 2\chi > 2\log_4 n$, then the probability of doing so is less than $1/n^2$, which is much smaller than $1/n$. Since S is arbitrary, this holds for all length k intervals of S that differ from R's correct spot.

Therefore, if $k - 2(\delta k + \sqrt{4\delta k \ln n}) > 2\log_4 n$ we can expect that any k-read taken from R will map to no other interval in S.

□

Theorem 8. *Given a δ-scaffold S for a genome G of length n and m reads of length k from G, such that $k - 2(\delta k + \sqrt{4\delta k \ln n}) > 2\log_4 n$, the brute-force short consensus algorithm will produce a consensus genome for G that has accuracy $1 - \delta e^{mk/n}$.*

Proof: By Lemma 6 we know that any length k interval of G is going to have at most $\delta k + \sqrt{4\delta k \ln n}$ errors, and by Theorem 7 we know that the short consensus algorithm will not try to map a read to the incorrect position. Since the brute-force short consensus algorithm attempts to place each read into every position of G, the correct one will eventually be found. And since the algorithm tolerates $\delta k + \sqrt{4\delta k \ln n}$ errors, each read will be mapped to its rightful place in the scaffold. Therefore, the only positions in the consensus genome which may end up in error are the position which have no reads mapped to them. By the result of [PSMW95], we know that the number of covered positions is at least $1 - e^{mk/n}$. For any position which remains uncovered, we can still use the scaffold's value which has a $(1 - \delta)$ probability of being correct. Therefore the overall accuracy of the consensus will be $1 - \delta e^{mk/n}$.

□

Note that the BLAST-like short consensus algorithm will perform nearly identically, since for short hash keys it is very unlikely that a read will fail to be mapped to some position of its correct interval. As can be seen from our empirical results this is born out in practice.

Lemma 9. *Let G be a genome of length n and let S be a δ-scaffold for G. Let R be a random nanopore k-read taken from G which then has each of its bases mutated with probability γ. The expected number of differences between R and the interval of S corresponding to R's location in G is $1 - \gamma - \delta + (4/3)\gamma\delta$.*

Proof: Let R_i be a random position of R. This position of R should map to a certain position of S_j corresponding to R's source in G. Without loss of generality, assume that the original spot in G contained A. The with probability $(1 - \delta)$, $S_j = A$ and with probability $(1 - \gamma)$, $R_i = A$. So the probability that both S_j and R_i remain A is $(1 - \gamma)(1 - \delta)$ and the probability that both mutate to the same symbol is $3(\delta/3)(\gamma/3) = (\delta\gamma/3)$. The total expected probability that the symbols agree is therefore $(1 - \gamma)(1 - \delta) + (\delta\gamma/3)$ which simplifies to $1 - \delta - \gamma + (4/3)\delta\gamma$. So the probability that there is a discrepancy between the positions is $(4/3)\delta\gamma - \delta - \gamma$.

□

The brute-force short consensus algorithm takes time $O(kmn)$ where k is the read-length, n is the genome length, and m is the number of reads used. This is clearly an upperbound on the running time of the BLAST-like short consensus algorithm which, given a reasonable hash-key length, we can expect to complete in time $O(km)$, since each key should be very close to unique in the genome.

One final observation about the Short Consensus algorithm is worth mentioning: By its nature, it can be used to map a read to all of the positions where it matches. This is a robust quality that would help very much with real biological genomes, which, unlike random strings, contain a great deal of repeated sequences.

5 Implementation

As part of this project, we implemented the Short Consensus algorithm. The implementation executable is called 'CSC 2427 Project.exe', and was programmed in C++ with MFC for the graphical user interface. The system requirements of the program depend entirely on which genome it is to be run, since it stores a scafffold as well as the parallel arrays and hash table mentioned above in memory.

For our implementation, we assume that a separate scaffold building algorithm has given us a scaffold containing errors. The purpose of the implementation is to demonstrate how quickly and accurately the errors can be removed from the scaffold.

Please refer to the screenshots below while reading the following description of how to use the program, which has four steps:

5.1 Step 1:

The screenshot below in Figure 2 shows the program immediately after loading. In this screen, the user selects the parameters with which to execute the Short Consensus algorithm.

Figure 2: The Short Consensus algorithm with no genome loaded

The user has control over five algorithm parameters. The 'Error Rate' parameter determines what fraction of the scaffold's indices have been permuted. The 'Hash Table Key Size' parameter determines how many keys the hash table will have; it has 4^c keys, where c is the key size. The 'Target Coverage' parameter determines how many reads will be read; reads will continuously be read and mapped to the scaffold until this level of coverage is reached. Finally, the 'Read Size' parameter determines the size of the random reads that are to be mapped by the algorithm. Note that the reads being mapped do not contain errors.

When the 'Load Genome' button is pressed, a dialog box selecting which genome file in .fa format to read. It is capable of reading files containing both capital and lower case bases. Once a genome has been selected, the program reads it, builds a scaffold, builds a hash table, and goes to the next step.

5.2 Step 2:

The screenshot below in Figure 3 shows the program immediately after a .fa genome file has been opened. This screen provides some basic information about the genome such as its file name and genome size in base pairs. Note that the algorithm parameters have been locked and can no longer be changed at this point.

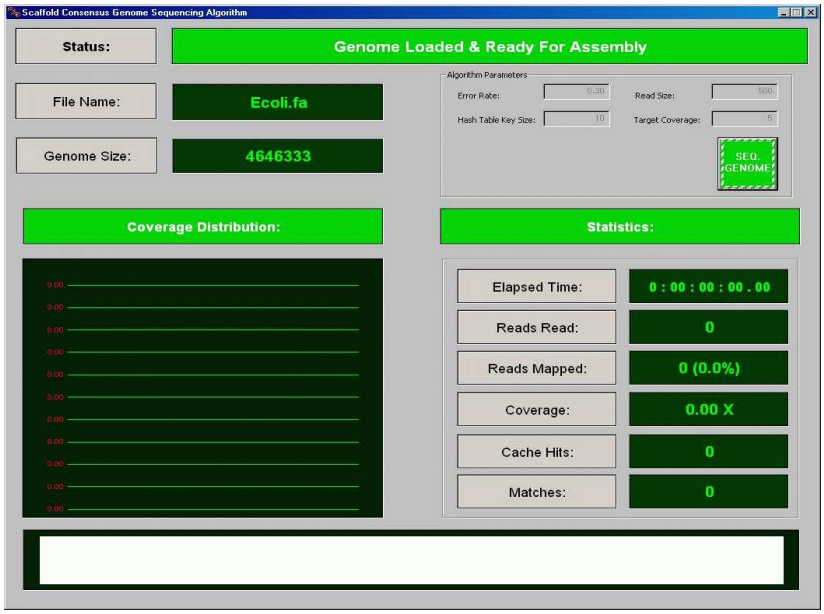

Figure 3: The Short Consensus algorithm with a genome loaded

5.3 Step 3:

The screenshot below in Figure 4 shows the execution of the Short Consensus algorithm. This screen displays a variety of statistics about the algorithm.

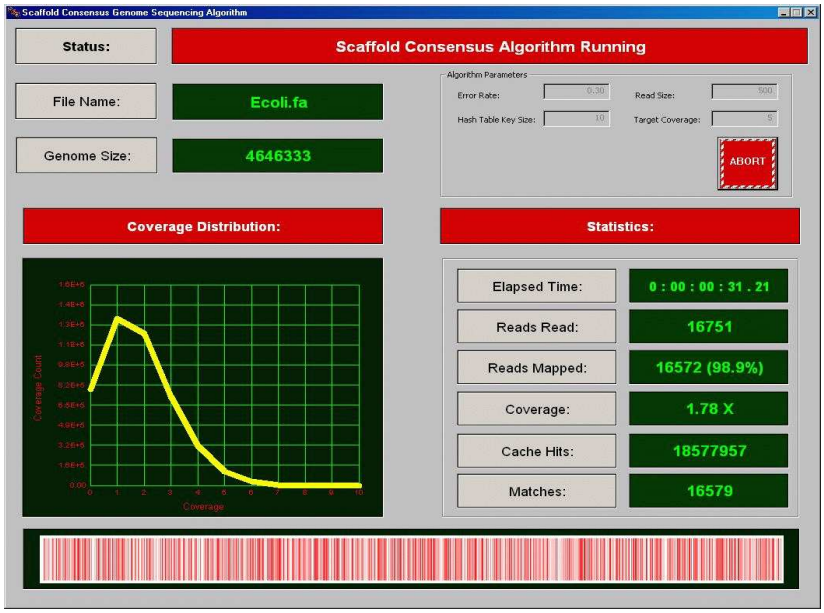

Figure 4: The Short Consensus algorithm while executing

'Elapsed Time' shows how long the algorithm has been running. 'Reads Read' shows how many random reads have been read. 'Reads Mapped' shows how many of those reads have been successfully mapped to the scaffold, as well as what percentage of the total reads that represents. 'Coverage' indicates how many times the total number of mapped reads would cover the entire genome in absolute terms. 'Cache Hits' shows how many cache hits the random reads have had. Finally, 'Matches' shows how many of the cache hits were exact matches.

The chart under the 'Coverage Distribution' heading gives an indication of what kind of coverage most of the indices in the scaffold have. For example, the chart in Figure 4 shows that very few indices in the scaffold have 10× coverage, whereas very many of them have 1× coverage.

The long picture at the bottom of the window shows the actual distribution of the mapped reads in graphical form. The darker the horizontal bands, the greater the coverage at that position.

5.4 Step 4:

The screenshot below in Figure 5 shows the results of the Short Consensus algorithm once the target coverage has been reached. The accuracy of the algorithm is shown in the status bar at the top. This accuracy is calculated by comparing the result of the consensus algorithm to the original genome from the input file, and counting the number of differences. The 'Reset' button clears the memory and returns the program to Step 1.

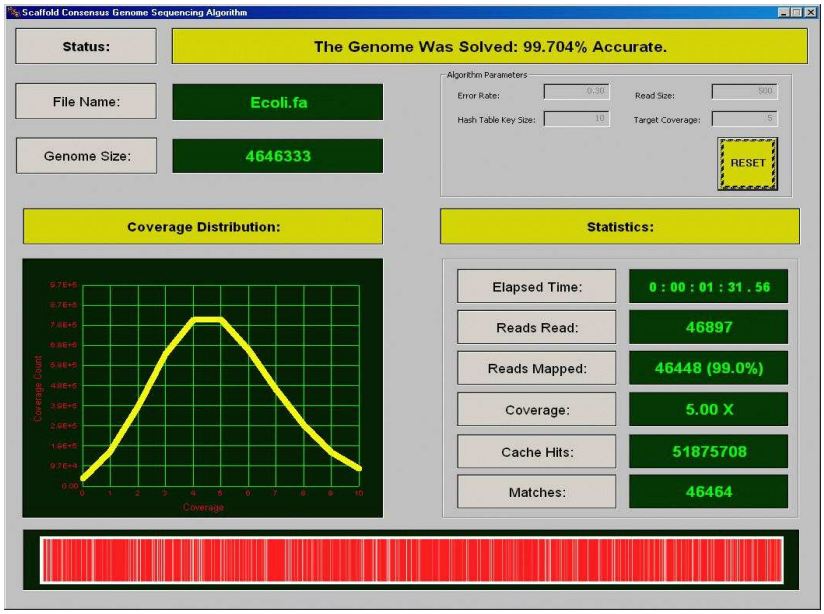

Figure 5: The Short Consensus algorithm after completion

6 Empirical Results

We were able to run our implementation on a number of different real genomes and genome fragments in the form of chromosomes. The Chicken DNA sequences were obtained from [Was]. The results of these tests are shown in the table below.

All tests were run on a 3.2 GHz Pentium 4 Processor with 2GB of RAM. Each test used random reads of length 500, a scaffold error-rate of 0.3, hash keys of length 10. We have results for both 5× coverage and 10× coverage.

Results for 5× coverage.					
	Genome Name	Size	Time (Seconds)	Accuracy (%)	Reads Mapped (%)
1	Chicken Chromosome 16	190218	1.00	99.496	98.7
2	Chicken Chromosome 32	990148	9.54	99.783	98.7
3	Chicken Chromosome 27	2501418	39.53	99.656	99.0
4	Salmonella Typhimurium	4294297	101.29	99.707	98.8
5	E.Coli	4646333	113.82	99.688	99
6	Chicken Chromosome 24	5779841	202.09	99.720	99
7	Chicken Chromosome 21	6044381	215.23	99.705	98.9
8	Chicken Chromosome 17	9892546	573.67	99.687	98.9
9	Chicken Chromosome 20	13294267	1067.75	99.735	98.9

Results for 10× coverage.					
	Genome Name	Size	Time (Seconds)	Accuracy (%)	Reads Mapped (%)
1	Chicken Chromosome 16	190218	1.98	99.935	99
2	Chicken Chromosome 32	990148	18.25	99.975	99
3	Chicken Chromosome 27	2501418	80.14	99.906	99.1
4	Salmonella Typhimurium	4294297	201.51	99.938	99
5	E.Coli	4646333	219.98	99.934	99
6	Chicken Chromosome 24	5779841	401.04	99.973	99
7	Chicken Chromosome 21	6044381	449	99.948	99
8	Chicken Chromosome 17	9892546	1127.4	99.926	99
9	Chicken Chromosome 20	13294267	2081.01	99.967	98.9

These results were promising for a number of reasons. Firstly, they are very good in terms of accuracy, with all of the results being at least 99.9% accurate. Secondly, the algorithm consistently wasted fewer than 1% of the reads. Since reads are relatively expensive when compared to computer power, this is important for keeping the cost of sequencing down. Thirdly, the algorithm performed well on both prokaryotic and Eukaryotic genomes. Finally, the results required very little time on a normal consumer-level computer with a single processor, which means that this algorithm would be very cheap to implement, and performs very quickly, two of the goals of this project. Implementing this algorithm in hardware is certainy feasible.

In short, given the read size and error rate, the accuracy results are consistent with Theorem 8 which suggests that at least with respect to our analysis, our test genomes behave very similarly to random strings.

One item worth mentioning is that the reads which went unmapped did so because they could not be hashed to the correct site. When running the algorithm, shortening the hash key results in more reads being mapped at the expense of time. Therefore, if reads are relatively expensive, it is worth-while to build a smaller hash table, or even to use brute-force mapping. On the other hand, if an acceptable number of reads are being mapped it may pay to increase the hash key length and therefore improve running times. A value near $\log_4 n$, where n is the length of the whole genome, seems optimal since that

key length would create the smallest hash table in which each entry is expected to map uniquely to the genome.

7 Concluding Remarks

In conclusion, given nanopore technology, both the Short and Long Consensus algorithms seem very promising. The Short Consensus algorithm's practical performance matches its predicted theoretical performance quite closely. Being both simple and effective, these algorithms could potentially live up to their goal of bringing costs down once nanopore technology is mature.

Though shotgun genome assembly should work well for scaffold building, it will be necessary to make minor modifications for it to work in this setting. Shotgun assembly should work relatively well for nanopore reads because they are many times longer than the reads currently being used. This may make it economical to build a small number of scaffolds and use them in the Long Consensus algorithm. If accurate shotgun assembly is more difficult than sequencing a large number of short reads, then one can use the Short Consensus algorithm.

There is of course more work that can be done on this project in the future. For example, incorporating a more realistic error model which includes insertions and deletions would bring it one step closer to being usable. It would also be interesting to run the Short Consensus algorithm on genomes which are much larger.

Since the Consensus algorithms can be implemented so cheaply, and since they work so quickly and accurately, they essentially reduce the problem of assembling a highly accurate genome to assembling a scaffold with about 70% accuracy. In other words, if one can build scaffolds, then one is very nearly done. Even without the nanopore technology, there may be other ways of constructing genomes with about 70% accuracy which could then take advantage of either Consensus algorithm.

References

[CGB+04] P. Chen, J. Gu, E. Brandin, Y. Kim, Q. Wang, and D. Branton. Probing Single DNA Molecule Transport Using Fabricated Nanopores. *Nano Letters*, Vol. 4, No. 11:2293 – 2298, 2004.

[Con01] Internation Human Genome Sequencing Consortium. Initial Sequencing and Analysis of the Human Genome. *Nature*, Vol. 409, No. 6822:860 – 921, 2001.

[Gre97] P. Green. Against a Whole-Genome Shotgun. *Genome Research*, 7:410 – 417, 1997.

[Gre02] P. Green. Whole-Genome Disassembly. *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 99, No. 7:4143 – 4144, 2002.

[Har] Harvard Nanopore Group. http://www.mcb.harvard.edu/branton/.

[Her06] A. Hertel. CS 2427 -Computational Biology Lecture 2 Scribe Notes. Lecturer: Michael Brudno, 2006.

[PSMW95] E. Port, F. Sun, D. Martin, and M. Waterman. Genomic mapping by end-characterized random clones: A mathematical analysis, 1995.

[VAM01] J.C. Venter, M.D. Adams, and E.W. Myers. The Sequence of the Human Genome. *Science*, 291:1304 – 1391, 2001.

[Was] Washington University Genome Sequencing Center. http://genomeold.wustl.edu/projects/chicken/index.ph

[WLS02] R.H. Waterston, E.S. Lander, and J.E. Sulston. On the Sequencing of the Human Genome. *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 99, No. 6:3712 – 3716, 2002.

[WM97] J.L. Weber and E.W. Myers. Human Whole-Genome Shotgun Sequencing. *Genome Research*, 7:401 – 409, 1997.