Reducing Genome Assembly Complexity with Optical Maps Final Report

Lee Mendelowitz
LMendelo@math.umd.edu

Advisor: Dr. Mihai Pop
Computer Science Department
Center for Bioinformatics and Computational Biology
mpop@umiacs.umd.edu

May 15, 2012

Abstract

The goal of genome assembly is to reconstruct contiguous portions of a genome (known as contigs) given short reads of DNA sequence obtained in a sequencing experiment. De Bruijn graphs are constructed by finding overlaps of length $k-2$ between all substrings of length $k-1$ from reads of at least k bases, resulting in a graph where the correct reconstruction of the genome is given by one of the many possible Eulerian tours. The assembly problem is complicated by genomic repeats, which allow for exponentially many possible Eulerian tours, thereby increasing the de Bruijn graph complexity. Optical maps provide an ordered listing of restriction fragment sizes for a given enzyme across an entire chromosome, and therefore give long range information that can be useful in resolving genomic repeats. The algorithms presented here align contigs to an optical map and then use the constraints of these alignments to find paths through the assembly graph that resolve genomic repeats, thereby reducing the assembly graph complexity. The goal of this project is to implement the Contig-Optical Map Alignment Tool and the Assembly Graph Simplification Tool and to use these tools to simplify the idealized de Bruijn graphs on a database of 351 prokaryotic genomes.

1 Introduction

Genome assembly is the computational task of determining the total DNA sequence of an organism given reads of DNA sequence obtained from a sequencing experiment. DNA is a double stranded helical molecule, where each strand comprises a sugar-phosphate backbone and a sequence of nucleobases. The four nucleobases are: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C), with complementary base pairing between (Guanine, Cytosine) and (Adenine, Thymine). For each nucleobase along the primary strand, the complementary base appears in the same position of the complementary strand. Inside the cell, DNA molecules are condensed into a coiled secondary structure known as a chromosome. Due to the compact structure of the chromosome, the DNA bases are relatively inaccessible. As a result, experimentalists do not have

the ability to read an organism's DNA sequence directly from the chromosome itself. Instead, it is necessary to perform a DNA sequencing experiment.

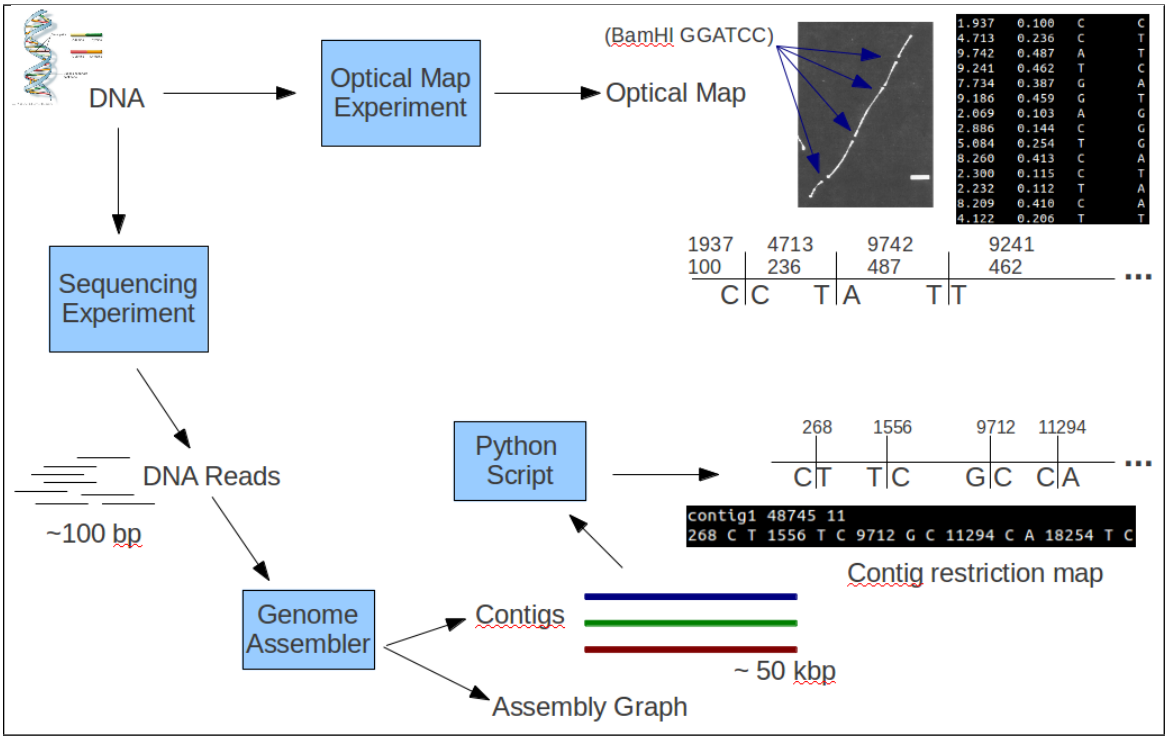


Figure 1: This figure provides the context for the algorithms to be developed in this project. A sequencing experiment produces short reads of DNA sequence from a set of chromosomes. Genome assembly software uses the reads to construct an assembly graph, and from the assembly graph outputs a set of contiguous DNA sequence known as contigs. In an optical mapping experiment, the same chromosomes are digested with a restriction enzyme, which cuts the DNA at sites with a specific recognition sequence, usually six base pairs in length. An optical map gives the restriction site pattern for each chromosome, providing the mean and standard deviation for each fragment length, as well as limited sequence information around each restriction site. An in-silico contig restriction map can be created for each assembled contig. The tools created as part of this project will align contigs to the optical map and search for unique paths through the assembly graph that are consistent with the optical map, thereby extending contig length and reducing the complexity of the assembly graph.

1.1 DNA Sequencing and Genome Assembly

In a DNA sequencing experiment, DNA is extracted from a sample and sheared into random fragments. The DNA sequence at the end of the fragments is determined experimentally, producing a set of reads. Today's sequencing technology is highly automated and parallelized, cheaply producing short reads of 35 to 400 base pairs long [6]. While the reads are error prone, sequencing machines deliver high throughput, providing many reads that cover the same portion of the genome. The task of genome assembly is an inverse problem which is often compared to the pro-

cess of assembling a jigsaw puzzle: given the DNA reads, an assembler must assemble the unique genome from which the reads originate. In practice, an assembler will not be able to perfectly reconstruct the entire genome, and instead will output a set of contiguous DNA sequences known as contigs.

Currently there are two different formulations of the genome assembly problem. In one formulation, known as overlap-layout-consensus (OLC), each read obtained in a sequencing experiment is compared to every other read in search of overlaps. This produces what is known as an overlap graph where reads are nodes and edges represent overlaps. A reconstruction of the genome is given as a Hamiltonian path through the graph that visits each node exactly once. Identifying Hamiltonian paths is NP-Hard, so algorithms that use OLC rely on various heuristics to produce contigs [4].

A second approach to genome assembly is to construct a de Bruijn graph from reads [3]. For a given read length of $k > 1$, a de Bruijn graph can be constructed by creating a node for each length $(k-1)$ substring of DNA bases that appears in a read. A directed edge is drawn from node A to node B if B follows A in a read (meaning B overlaps A by $k-2$ characters). Given perfect sequencing data such that each node in the graph is a length $k-1$ substring from the genome and each $k-2$ overlap is represented by exactly one edge, then the correct reconstruction of the genome is given by an Eulerian tour through the graph that uses each edge exactly once. From the construction of the de Bruijn graph, it should be clear that the in-degree of a node is equal to the out-degree, except for the starting and ending node corresponding to the beginning and end of a chromosome. For the case of a circular chromosome, each node will have equal in- and out-degrees. In addition, a perfect de Bruijn graph for a circular chromosome will consist of a single strongly connected component.

In contrast to Hamiltonian paths, Eulerian tours are easy to identify and can be found in linear time [3]. The de Bruijn graph formulation of genome assembly has the advantage that it avoids the computationally expensive pairwise comparison of all reads. However, it comes with the disadvantage that information of the sequence of each read is lost after nodes with sequence of length $k-1$ are formed. In addition, a de Bruijn graph structure is highly sensitive to sequencing errors, which yield length $k-1$ substrings which do not exist in the genome, resulting in extraneous nodes and edges in the graph.

The original de Bruijn graph with nodes of length $k-1$ can be simplified through a series of graph compression techniques which reduce the number of nodes and edges but leave the DNA sequence spelled by each possible Eulerian tour unchanged (see [1]). As a result of these lossless compression techniques, edges become labeled with DNA sequence. This simplified graph has the property that the sequence between each edge and the nodes which define it overlap by $k-2$ bases.

Even with graph compression techniques, the number of possible Eulerian tours for a given graph is exponential in the number of repeat nodes (i.e. nodes which must be traversed multiple times in an Eulerian tour). The number of potential genome reconstructions ending at node t of a de Bruijn Graph $G = (V, E)$ with node set V, and collection of directed edges E can be determined exactly [1]. Let $\boldsymbol{A}$ be the adjacency matrix of G, and let $d^-(u)$ and $d^+(u)$ be the in-degree and out-degree of node u. Define $\boldsymbol{r}$ such that $r_t = d^+(t) + 1$ for node t and $r_u = d^+(u)$ for all nodes $u \neq t$. The Laplacian matrix of G is given by $\boldsymbol{L} = \texttt{diag}(\boldsymbol{r}) - \boldsymbol{A}$. Then the number of possible linear genome reconstructions W corresponding to Eulerian tours through G ending at node t is given

by:

$$W(G,t) = \det L \left\{ \prod_{v \in V} (r_u - 1)! \right\} \left\{ \prod_{(u,v) \in E} a_{uv}! \right\}^{-1} \tag{1}$$

For the case of a circular chromosome, the number of genome reconstructions is given by $W(G,t)/d^+(t)$ since each circular genome has $d^+(t)$ linear representations that end at node t.

As an example of the application of (1), consider the simplified de Bruijn graph with $k = 100$ for the single circular chromosome of the bacterium *Mycoplasma genitalium*, which has the smallest known cellular genome at 580,076 base pairs. The de Bruijn graph has 84 nodes and 120 edges where 28 nodes are repeated twice and 4 nodes are repeated three times in each Eulerian tour. The number of unique genome reconstructions from this de Bruijn graph for the smallest known cellular genome is a staggering 21,897,216.

Since the number of possible genome reconstructions for the typical de Bruijn graph is so large, it is useful to define a different metric that measures the repeat structure of the graph. For a given repeat node v with $d^+(v) = d^-(v) = a$, the complexity of the node, $C(v)$, is defined to be equal to the number of targeted experiments necessary to match each incoming edge with each outgoing edge [7]:

$$C(v) = \sum_{i=2}^{a} i = \frac{a(a+1)}{2} - 1 \tag{2}$$

The sum is from $i = 2$ to $i = a$ since once these incoming edges are matched to an outgoing edge, the last remaining incoming edge is automatically matched to the last remaining outgoing edge. The total finishing complexity of a de Bruijn graph $G = (V, E)$ is the sum of the finishing complexity of the repeat nodes:

$$C(G) = \sum_{v \in V} C(v) \tag{3}$$

Since the number of Eulerian paths is exponential in the number of repeat nodes, the genome assembly task is to identify the one Eulerian tour of many possible tours that gives the correct reconstruction of the genome. Additional experimentally obtained information must be used to provide constraints on the allowable tours. Experimentally obtained information frequently used for this purpose are pairs of reads, known as mate pairs, that are separated by an approximately known distance in the genome [4]. Instead, in this project, we consider using information provided by an optical map.

1.2 Optical Mapping Technology

An optical map is produced by immobilizing a DNA molecule tagged with fluorescent marker on a slide [5]. A restriction enzyme is washed over the slide, and the enzyme cuts the DNA at loci with a particular 4 to 8 base pair recognition sequence known as a restriction site, producing a set of restriction fragments whose lengths are measured. As an example, the restriction enzyme PvuII has recognition sequence `CAGCTG`, so PvuII will cut the DNA molecule wherever this sequence of bases occurs. An optical map provides an ordered list of restriction fragment sizes along the length of the DNA molecule. Future optical mapping techniques may also be able to provide limited DNA sequence data around the restriction sites. Experimental errors from the optical mapping process include missing restriction sites due to partial enzyme digestion of the molecule, false

restriction sites due to random breakages of the DNA molecule, and missing small restriction fragments. Despite these errors, optical maps provide useful long range information over the entire length of a DNA molecule - information that can be used to determine segments of the correct Eulerian tour through the de Bruijn graph.

2 Project Objectives

The goal of this project is to build software that uses optical maps with limited sequence information around each restriction site to resolve repeats in an idealized de Bruijn assembly graph, thereby extending the length of contigs and reducing the number of edges and nodes in the graph. Two separate pieces of software will be developed. The Contig-Optical Map Alignment Tool will align contigs to an optical map based on the ordered listing of restriction fragment lengths. The Assembly Graph Simplification Tool will provide functionality to simplify a de Bruijn graph using a unique shortest path heuristic. Lastly, a pipeline will be developed to integrate these tools.

These algorithms will be individually validated on user generated data sets. The pipeline will be used to validate these tools collectively on a database of ideal de Bruijn graphs for 351 prokaryotic genomes.

3 Approach

The software produced for this project consists of two major components. The first component finds alignments of contigs extracted from the de Bruijn graph to the optical map. The second component will use contig alignments to simplify the de Bruijn graph.

3.1 Contig-Optical Map Alignment Tool

The Contig-Optical Map Alignment Tool finds significant alignments of contigs extracted from a de Bruijn graph to a single optical map. Contigs can easily be extracted from the de Bruijn graph by reading the sequence along unambiguous paths that do not pass through any repeat node. Working with de Bruijn graphs that have been simplified through the graph compression techniques from [1], contigs are given by the concatenated DNA sequence of neighboring nodes, including the sequence on the edge. The inputs to this tool will be the DNA sequence for each contig to be aligned, the optical map data, and the recognition sequence of the enzyme used to produce the optical map. The optical map data consists of an ordered list of restriction fragments lengths, the standard deviations of the restriction fragment lengths, and several bases of sequence data on each side of the restriction site.

First, "in-silico" restriction sites are identified along each contig using the known recognition sequence of the enzyme used to produce the optical map. This produces an ordered list of restriction fragment lengths that would be produced if the contig were to be perfectly digested by the same enzyme used to produce the optical map. Next, each contig is aligned to the optical map through a dynamic programming algorithm which scores alignments based on a comparison of aligned restriction fragment lengths, the number of missed restriction sites, and the Levenshtein edit distance between the DNA sequence around the aligned restriction sites of the contig and optical map. Lastly, the statistical significance of each alignment is evaluated through a permutation test.

The total alignment score is given by a weighted sum of the χ^2 score S_{χ^2}, the number of missed restriction sites m_r, and the sum of the Levenshtein edit distance at the aligned restriction sites S_d, with constant weights C_s and C_r. The best match is given by the lowest score.

$$S = S_{\chi^2} + C_r \times m_r + C_s \times S_d \tag{4}$$

The χ^2 score S_{χ^2} measures the sum of the squared difference of restriction fragment lengths, where the length differences are measured in standard deviation units. Each true optical fragment length can be modeled as $\sim N(o_i, \sigma_i^2)$ where o_i is the mean and σ_i^2 is the variance of the fragment length obtained from repeated experimental measurements. The χ^2 scoring function for the alignment of in-silico contig restriction fragments of lengths $c_0, \ldots, c_{n-1}$ to optical map fragments of lengths $o_j, \ldots, o_{j+n}$ with corresponding standard deviations $\sigma_j, \ldots, \sigma_{j+n}$ is given by:

$$S_{\chi^2} = \sum_{i=0}^{n-1} \frac{(c_i - o_{j+i})^2}{\sigma_{j+i}^2} \tag{5}$$

The Levenshtein edit distance is the minimum distance between two character strings using a "+1" cost for each character deletion, insertion, or substitution, and is computed using dynamic programming in $\mathcal{O}(mn)$ for comparison of two stings with lengths m and n. S_d represents the total edit distance at aligned sites:

$$S_d = \sum_{i=0}^{n-2} d_{i,j+i} \tag{6}$$

where $d_{i,j+i}$ is the edit distance of the ith aligned site. Note that the sum in (6) only includes a total of $n-1$ terms since when n fragments are aligned, $n-1$ restriction sites are aligned.

3.1.1 Dynamic Programming

The scoring function given by (4) admits a dynamic programming algorithm to find the best possible alignment of the entire contig to the optical map through the jth restriction fragment of the optical map [2]. Let S_{ij} be the score of the best alignment matching the end of the ith contig fragment with the end of the jth optical map fragment. This score can be determined by considering extending previously scored alignments, and incorporating the appropriate edit distance and missed restriction site penalties for each possible extension:

$$S_{ij} = \min_{0 \le k \le i, 0 \le l \le j} C_r \times (i - k + j - l) + C_s \times d_{ij} + \frac{(\sum_{s=k}^{i} c_s - \sum_{t=l}^{j} o_t)^2}{\sum_{t=l}^{j} \sigma_t^2} + S_{(k-1)(l-1)} \tag{7}$$

where d_{ij} is the edit distance for the alignment of the ith restriction site from the contig to the jth restriction site from the optical map.

For a given k and l, the score given by (7) is for an alignment where optical map restriction fragments k thru i are considered to be one restriction fragment with $k-i$ false-positive restriction sites, and contig restriction fragments l thru j are considered to be one contig restriction fragment with $j-l$ restriction sites missing from the optical map. This gives an $\mathcal{O}(m^2n^2)$ algorithm, where m is the number of contig restriction fragments and n is the number of restriction fragments in the optical map. An example of this process is illustrated in Figure 2 below.

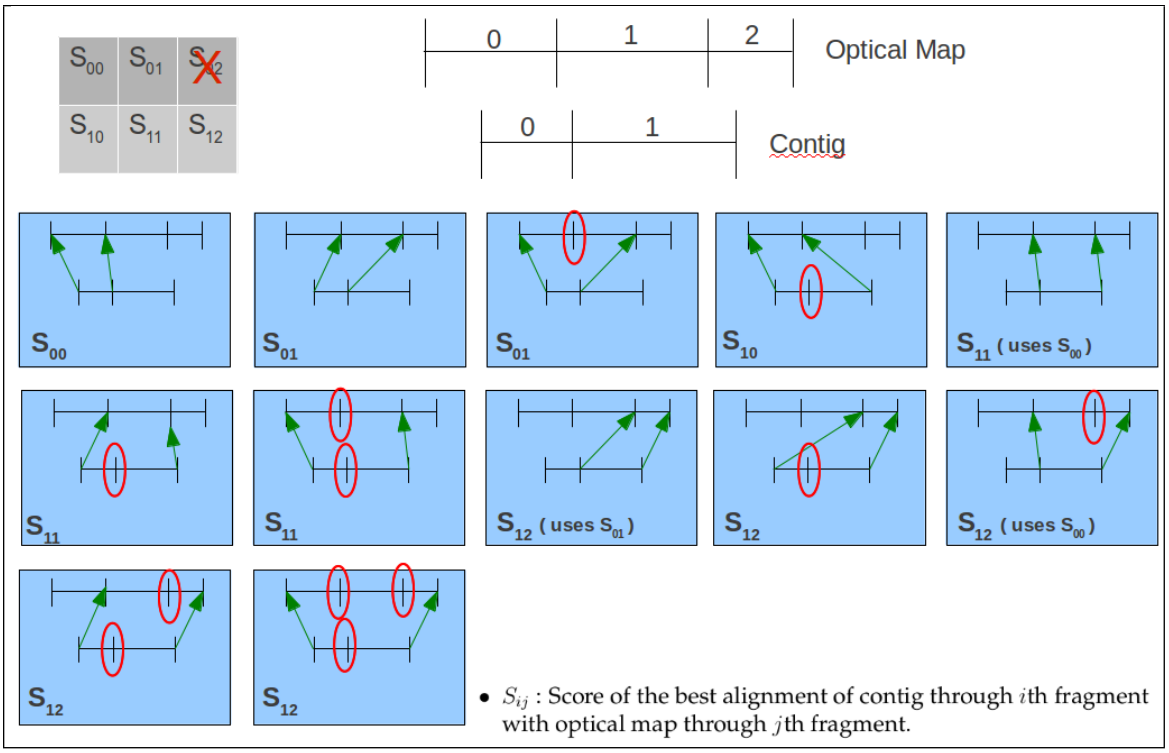


Figure 2: The different alignments that are scored in setting each element S_{ij} using equation (7) for two contig fragments $(m = 2)$ and three optical map fragments $(n = 3)$. Green arrows indicate the alignment of sites. Restriction sites circled in red count as a missed site for the alignment being scored. S_{02} represents the score of the alignment of the zeroth contig fragment to the second optical map fragment. This alignment is not considered because this means the first fragment of the contig would not be aligned to the optical map.

3.1.2 Additional Alignment Constraints

The search space is pruned by not scoring alignments where an aligned fragment has a poor χ^2 score in (7):

$$\frac{(\sum_{s=k}^{i} c_s - \sum_{t=l}^{j} o_t)^2}{\sum_{t=l}^{j} \sigma_t^2} \geq C_\sigma \tag{8}$$

C_σ is a user input, which defaults to 5.

Since DNA is double stranded, it is possible that the optical map represents the locations of restriction sites along one strand while the contig represents the sequence from the complementary strand. To test for this possibility, both the contig and the reverse-complement of the contig are aligned to the optical map and the best possible alignment is selected. The reverse-complement of the contig restriction pattern is found by reversing the order of the fragments and taking the reverse complement of the sequence around each restriction site. In addition, to capture the periodic nature of a circular chromosome, the optical map must be doubled in length where each of the original n fragments is duplicated after the nth fragment. This allows for alignments that extend beyond the right edge of the original linearized optical map, which is a valid alignment for

a circular genome.

To test the significance of the best alignment, a permutation test is performed by computing the best alignment score for the contig against a permuted version of the optical map. 500 samples from the space of permuted optical maps is used to calculate the p-value. The p-value is given by the fraction of permutation trials where the contig has a better scoring alignment to a permuted optical map than the original optical map. An alignment is deemed significant if $p < 0.05$. Note that the permutation test is an expensive operation, as it increases run time by a factor of 500. This effect is mitigated by a parallel implementation of the alignment tool.

3.2 Assembly Graph Simplification Tool

The Assembly Graph Simplification Tool takes as input the de Bruijn graph, a list of significant contig to optical map alignments as found by the Contig-Optical Map Alignment Tool, the optical map data, and the recognition sequence of the enzyme used to produce the optical map. From these inputs, the tool tries to determine the correct genomic sequence between consecutively aligned contigs and uses this information to simplify the assembly graph.

For each pair of neighboring uniquely aligned contigs A and B, the Assembly Graph Simplification Tool uses Dijkstra's algorithm to find the shortest path through the de Bruijn graph starting from the edge corresponding to contig A and ending at the edge corresponding to contig B. We rely on this heuristic of the shortest path since, in general, finding a path of a specified length through a graph is NP Hard [7]. In addition, since the contigs aligned to the optical map as neighbors, it is sensible to believe that the path through the graph between these edges should could be the shortest path. Of course, whether the shortest path is the correct path depends on the repeat structure of the assembly graph.

If a unique shortest path is found between contig A and contig B, the sequence generated by this path is aligned to the gap in the optical map between the alignments for A and B. This test is performed using the Contig Optical Map Alignment tool. If the alignment is deemed significant, then the shortest path is accepted as true and simplifications are made to the graph. This is discussed in further detail in the next two sections.

3.2.1 Dijkstra's Algorithm

Dijkstra's algorithm is an intuitive $\mathcal{O}(E + V \log V)$ algorithm for finding the shortest path length between a source and target node in a graph with non-negatively weighted edges. The algorithm makes use of the basic property that if a node Q is on a shortest path from P to R, then the shortest path from P to R includes the shortest path from P to Q. The algorithm maintains a set of of nodes that have been "visited" and a priority queue of nodes that have been "seen". In the first iteration, the current node is set to be the source node, and the current distance is set to 0. The shortest distance from the source node to each successor node of the current node is tentatively computed by adding the edge's weight to the current distance. If this distance is shorter than the previously calculated distance for the successor node, then the successor node is added to the priority queue with priority equal to the updated shortest distance. At the end of the iteration, the current node is added to the visited set, and the node with the next shortest distance is dequeued from the priority queue and selected to be the current node. This repeats until the target node is selected as the current node. The shortest distance from source to target is then the priority of the target node, which is determined when it is dequeued.

Since the de Bruijn graph is a strongly connected graph, at least one shortest path always exists between any source and any target node. However, we are interested specifically in the cases where the shortest path is unique. A small modification to Dijkstra's algorithm can be made to maintain a list of predecessor nodes for each node that is added to the priority queue of "seen" nodes. In this way, when the target is selected from the priority queue, its predecessor are known. All shortest paths in the form of an ordered listing of nodes (node path) can be reconstructed using the predecessor lists for each node in a shortest path, starting with the target node.

A shortest node path can be converted into a shortest edge path (i.e. an ordered listing of edges) by taking the shortest edges between neighboring nodes in the shortest node path. Since the de Bruijn graph is a multidigraph, there can be multiple shortest edges between any two nodes, each labeled with different genomic sequence. Therefore, a single shortest node path can yield many possible shortest edge paths, each giving a different genomic sequence. We are careful to count the number of possible shortest edge paths before enumerating all possibilities, as this can be a very expensive operation both in terms of computation and in terms of memory storage.

Since we are only interested in cases where a unique shortest path exists, we can improve efficiency by exiting early from the construction of all shortest node paths from the predecessor lists produced by the modified Dijkstra's algorithm for those cases where there are multiple shortest node paths. Likewise, we first count the number of possible edge paths, and only calculate the shortest edge path for a given node path if it is unique.

3.2.2 Evaluation of Path Candidates

If a unique shortest path is found through the graph between neighboring aligned contigs, it must be compared to the optical map fragments which fall in between the two aligned contigs. The candidate path will only be used to simplify the graph if it is deemed to be a significant match. The significance of this path is evaluated by forming the contiguous DNA sequence along the path, determining the in-silico restriction pattern, and using the Contig Optical Map Alignment Tool to calculate its alignment score to the gap in the optical map using (4). In this case, the alignment must start with the first restriction fragment of the optical map gap and end with the last restriction fragment of the optical map gap since we are trying to close the gap between neighboring aligned contigs. The candidate path is selected for closing the gap if the score of the alignment is significant, as determined through a permutation test.

3.2.3 Simplification of Graph

After all candidate shortest paths between neighboring aligned contigs are evaluated, those gap closures that are deemed significant by the Contig-Optical Map Alignment Tool are selected. If consecutive gap closures are deemed significant, then these shortest paths can be merged together into a single path. For example, if contigs A, B, and C align consecutively to the optical map and the shortest path between A and B and between B and C are both consistent with the optical map, then these shortest paths can be merged into a single path between A and C. It should be pointed out that this may not be the shortest path between A and C. Merging shortest paths in this way will make it easier to simplify the graph. These merged paths are then sorted in order of decreasing length and used to simplify the graph.

The graph is simplified by replacing a selected path, represented by a sequence of edges, with a single edge. This edge will be labeled with the DNA sequence of the selected path. The path is

replaced by removing the edges of the path from the graph. After this step, any nodes which are not in the main strongly connected component of the graph will be removed. This will include nodes with degree zero after the removal of edges in the previous step. However, nodes with non-zero degree may also be removed since it is possible that deleting edges will disconnect the graph into multiple connected components. This will happen only if the selected path is incorrect, as the ideal de Bruijn graph has the strongly connected property, and this property will hold after any correct simplification to the graph.

After replacing the selected path with a single edge and cleaning up the graph, there may now be unambiguous paths comprised of nodes of degree one, and these paths can be replaced with a single edge. This operation is an example of path compression. Path compression is performed by randomly selecting a node of degree one in the graph and then extending a path in the forward and reverse direction from this node until a node with degree greater than one is encountered. This yields a path of nodes of degree one, bounded with by a starting and ending node of degree greater than one. This path is then replaced by a single edge, and any interior nodes of the path are removed. This is repeated until the graph has no nodes of degree one, or until the graph is reduced into a single node.

Together, these graph simplification operations reduce the complexity of the graph. It should be noted that the order in which candidate paths are applied to simplify the graph does make difference. Modifications made to the graph by one accepted candidate path may invalidate changes to be made by another accepted candidate path if these paths have an edge in common. Simplifying the graph with the first candidate path will delete the edge, rendering the second candidate path invalid. In this case, if there is another edge between the same nodes with identical sequence, it is used to replace the missing edge. In the event that missing edges in an invalid candidate path cannot be replaced, the candidate path is discarded. The longest candidate paths (as measured in sequence length) are applied for graph simplification first. Path compression is only performed after all candidate paths are applied to the graph.

4 Implementation

The Contig-Optical Map Alignment Tool has been implemented in C++, and the Graph Simplification Tool has been implemented in python. In addition, a pipeline has been implemented in python which integrates these two tools together.

4.1 Contig-Optical Map Alignment Tool

The Contig-Optical Map Alignment Tool has been developed by updating source code from the open-source software package SOMA (described in [2]), which is a tool for matching and placing multiple contigs along an optical map. Code development was performed using the Vim text editor in a Linux environment. Development was performed on my home desktop computer (Ubuntu 11.10), on my MacBook Pro (Ubuntu 11.10 in VirtualBox), and on the Center for Bioinformatics and Computational Biology's computational resources. In mid-November, in an effort to keep my code in sync across these platforms, I created a local Git repository which is now hosted privately for free with `BitBucket.org`. Git is a distributed version control system that keeps a history of file changes and provides branching and merging functionality. The code was compiled using the `g++` compiler, version 4.6.1.

Development of the Contig-Optical Map Alignment tool involved several important tasks. The first task was to understand the existing alignment algorithm as implemented in the SOMA source code. While the source code itself lacked documentation, the reference [2] outlined the algorithm.

The second task was to reorganize and restructure the code to improve modularity and maintainability without changing the functionality of the original SOMA program. In particular this involved organizing global variables and constant variables into a separate header. C structs were converted into C++ classes to provide more flexibility. I also made use of the C++ Standard Template Library (STL) wherever possible in an effort to make the code easier to maintain. With each set of changes, I would recompile the software using `g++` and re-run the test case provided with the SOMA distribution to ensure that the results were unchanged.

The third task was to make functionality updates to the Contig-Optical Map Alignment Tool. This included writing functions to calculate the Levenshtein edit distance between strings and to compute the reverse complement of a DNA string. The alignment scoring function was updated to include the edit distance of sequence around each restriction site. In addition, functions to read input files and write output files were modified to handle sequence data. Lastly, the Contig-Optical Map Alignment tool was parallelized using OpenMP. The alignment of each contig to the Optical-Map is independent, and so this can be performed in parallel by different threads. Each iteration of the permutation test for a given contig is also independent, and so this is performed in parallel. Each of these `for` loops has one critical section to store the alignment result. The number of threads is a command line argument.

During testing of the tool, it was discovered that execution time with the permutation test was less than desirable. The GNU profiling tool `gprof` was used to determine which function calls took most of the execution time. It was determined that a staggering number of calls were made to the Levenshtein edit distance function, consuming approximately 50% of the execution time. The computation of the alignment was reformulated to calculate the edit distance of the sequence flanking an aligned restriction site as opposed to the edit distance of the sequence at the beginning and end of an aligned restriction fragment. This simple change allowed the calculation of edit distance to be made independently of the calculation specified by (7) (since d_{ij} does not depend on l or k). This resulted in significantly less calls to the edit distance function. During this phase of code optimization it was also discovered that opportunities existed to continue to the next iteration of a loop or break out of a loop when the condition specified by (8) was not met. Lastly, in the case of a circular chromosome, updates were made to ensure that no alignment is started beyond the right edge of the linear representation of the circular map. These code changes save unnecessary computation, yielding a more efficient program.

The software was implemented and tested using the Center for Bioinformatics and Computational Biology's computing resource `privet`: 4 x AMD Opteron(tm) Processor 850 (2400MHz), 32 GB Ram, RHEL5 x86_64.

4.2 Assembly Graph Simplification Tool

The Assembly Graph Simplification tool is a set of functions written in Python 2.7 using the Networkx library version 1.5. Development was performed using the same resources as the Contig-Optical Map Alignment Tool. Throughout development, functions were tested interactively using the iPython shell.

The implementation of Dijkstra's algorithm uses a modified version of Dijkstra source code provided in Networkx. This is redistributable under the BSD license.

The main tasks in developing the graph simplification tool were to develop methods to read (and write) a graph from (and to) the Graphviz dot file format; to find shortest paths and simplifying the graph; and to provide a additional functionality related to graph operations.

The code to import a graph from a .dot file requires both the input dot file and the FASTA file with the reference genome sequence. The .dot file labels each node and edge with a genomic location and a sequence length. Since all nodes are repeats, they are labeled with multiple genomic locations. Edges, on the other hand, have a unique genomic location. This import function extracts the sequence corresponding to a given genomic location and adds the sequence as an attribute to each node an edge. In addition, each edge is labeled with a weight which corresponds to how much sequence is added to a path if the edge were to be taken. For an edge between nodes A and B, this weight is equal to length of sequence on the edge (A, B) and any sequence on node B. In the de Bruijn graph framework, each edge labeled with sequence overlaps the sequence on the nodes by $k-2$ bases. Care is taken to avoid double counting the sequence contained in such overlaps when assigning edge weights. In addition, the import operation double checks that all nodes are genuine genomic repeats with respect to the reference genome sequence provided in the FASTA file.

In addition to the methods for finding shortest paths and for graph simplification which were already described in detail in section 3.2.1 and 3.2.3, many utility functions were written to aid in the validation tests on 351 prokaryotic genomes. These utilities include functions to compute the complexity of a graph; to determine the path through the graph between two edges corresponding to the true genomic sequence; to convert a path represented as a list of edges to a genomic sequence; to convert a single shortest node path to all possible shortest edges paths; and to check whether a given path is equivalent to the true path (in terms of genomic sequence). In addition, code was written to check the integrity of the de Bruijn graph, by asserting that the graph consists of one strongly connected component, that the in-degree is equal to the out-degree for each node, and that the sequence of edges overlaps with the sequence of nodes by $k-2$ bases. The integrity of the graph is asserted after each simplification operation.

Lastly, code is included to calculate the correctness of a path selected for gap closure as compared to the true genomic path. The path correctness is defined to be the ratio of length of the sequence which the selected path shares in common with the true path to the length of the true genomic sequence. These paths are represented as a list of edges, each labeled with sequence. The length of sequence that the paths share in common is computed by finding an alignment between edges of the selected path and the true path. Edges are only aligned together if they are labeled with identical sequence, and the alignment which maximizes the length of sequence on aligned edges is selected. This alignment is closely related to the longest common subsequence problem, the difference being that edges in the path are aligned instead of the total DNA sequence generated by each path. This is done for efficiency reasons and gives a conservative estimate of path correctness, since the length of the common sequence computed this way is less than or equal to the length calculated by the longest common subsequence approach. As an example, the true path and the selected path may differ by a single edge, and the sequence on these edges may differ by a single DNA base. However, in the computation of the common sequence, none of the sequence on these nearly identical edges is included.

5 Databases

Source code, scripts, and validation results are being maintained in a private git repository, hosted at `BitBucket.org`.

Reference genomes for the prokaryotic genomes are available from the National Center for Biotechnology Information's (NCBI) Genbank: `http://www.ncbi.nlm.nih.gov/genbank/`. These reference genomes can be used to construct artificial optical maps to test the tools developed as part of this project.

The input de Bruijn Graphs to used for validation are the simplified de Bruijn graphs found in [1] and [7], available at: `http://www.cbcb.umd.edu/ wetzeljo/matePairs/`

6 Validation

6.1 Contig-Optical Map Alignment Tool

The Contig-Optical Map Alignment tool was tested on two user generated data sets to validate its performance.

6.1.1 Validation Test 1

An optical map of 100 fragments was generated using a python script. The fragment lengths were selected at random uniformly from $[1.0, 10.0]$ kbp, the standard deviations were set to 1 bp (to simulate as little sizing error as possible), and sequence data of 1 bp preceding and following each restriction site was generated uniformly at random from the alphabet {A,C,G,T}. 10 contigs were extracted from this map, with probability of 0.5 in the forward orientation and probability 0.5 in the reverse orientation. The extraction locations were selected uniformly at random from the optical map, and the number of fragments per contig was selected uniformly at random from $[5, 20]$. In addition, 10 contigs were randomly generated using the same parameters described above. The permutation test was turned off. The Contig-Optical Map Alignment tool was run with $C_\sigma = 5$, $C_r = C_s = 12500$ to align the 10 true contigs and 10 random contigs to the generated optical map. These weights effectively ignore the χ^2 score component of the scoring function since the sequence error and restriction site error terms are weighted so highly. The algorithm correctly aligned the 10 true contigs to the correct location in the optical map, and the best scoring alignments for the random contigs were of poor quality, with many missed restriction sites. This test indicates that the Contig-Optical Map Alignment tool works correctly on perfect data.

6.1.2 Validation Test 2

10 contigs were extracted from the same optical map used in in the previous test. Sizing error was added to each contig fragment length, sampled uniformly from the range specified by $\pm 5\%$ of the mean fragment length given in the optical map. These contigs were aligned to the optical map in the forward direction, ignoring the sequence information around each site.

Copies of the contigs and the optical map were created with all restriction fragments in the reverse order. The reverse contigs were aligned to the reversed optical map.

The alignments of the reverse contigs to the reverse map were consistent with the alignment to the original map. This confirms that the alignment algorithm does not depend on the direction of the alignment.

6.1.3 Validation Test 3

An optical map of 400 fragments was generated using a python script. The fragment lengths were selected at random uniformly from $[1.0, 10.0]$ kbp, the standard deviations were selected uniformly at random to up to 5% of the mean fragment length, with a minimum standard deviation of 100 bp and a maximum of 500 bp. 30 contigs were extracted from the map, as in Test 1, but with simulated errors described as follows. Sizing error was added to each contig fragment length, sampled uniformly from the range specified by $\pm 5\%$ of the mean fragment length given in the optical map. A substitution error was made for each base in the sequence data with probability 0.1. A restriction site error is made with probability 0.1 by deleting a site in the contig or by inserting a false site in the contig halfway between consecutive sites. 10 contigs were also generated randomly as described previously. The permutation test was turned on, with 500 samples taken from the space of permuted contigs using a p-value threshold of 0.05.

The Contig-Optical Map Alignment tool was run with $C_\sigma = 5$, $C_r = C_s = 12500$. The 30 true contigs aligned to the correct position in the map. 1 out of 10 of the random contigs aligned to the map with significance (a false positive), with a p-value of 0.03. Of the 9 restriction sites in this random contig, 3 were not aligned to restriction sites in the optical map. With only 6 aligned restriction sites, the random contig had, by chance, a very good sequence score, which is why it performed well on the permutation test. The χ^2 score for the alignment was very poor, 67.7 for only 5 inner fragments.

This test was repeated using the same input optical map and set of contigs, with updated costs $C_r = 5$ and $C_s = 3$, which gives more importance to the χ^2 score. With these updated costs, the best alignment for the previous false positive was no longer deemed significant. However, these updated costs introduced a new false positive, with p-value 0.046. This new false positive contig has 9 sites, 3 of which were not aligned to the optical map.

False positives can be eliminated by putting a constraint on the fraction of restriction sites that are allowed to be missed (say 10%). This constraint can be used to prune the search space (just as the condition on fragment length specified by (8)). This constraint on missed restriction sites will also improve the run time of the alignment algorithm.

6.2 Assembly Graph Simplification Tool

6.2.1 Validation Test

The shortest paths algorithm and simplification routines were tested both on a small, generic user generated graphs with positive edge weights. These graphs do not represent de Bruijn assembly graphs. These graphs were generated by writing a Graphviz .dot file by hand. The generated graphs included features such as self loops and multiple shortest path solutions. In addition, the shortest path solutions were not always the shortest in terms of edge count. In each instance, the graph simplification algorithm identified the correct shortest path between the source and target node.

The assembly graph shortest path and simplification routines were also tested on the de Bruijn graph for *Mycoplasma genitalium* (NC_000908). Arbitrary source and target nodes were selected for the identification of shortest paths and the simplification of the graph. The simplification of the graph was verified by manually inspecting the attributes of the new edge, and checking that the nodes interior to the path were removed as appropriate.

6.3 Integration Results

The pipeline which integrates the Contig-Optical Map Alignment Tool and Assembly Graph Simplification Tool was run on a database of 351 prokaryotic reference genomes and simulated optical maps. Optical maps were simulated from each reference genome sequence using the BamHI recognition sequence (GGATCC). The optical maps were generated with one additional base on either side of the recognition sequence. The optical maps included fragment sizing error, substitution error in the additional sequence information, and missing restriction sites. Three error settings were used:

- Error-free (perfect optical maps).
- Sizing error $\sim N(0, (0.01 \times l)^2)$, $\Pr(\text{base substitution}) = 0.10$, $\Pr(\text{missing site}) = 0.05$.
- Sizing error $\sim N(0, (0.05 \times l)^2)$, $\Pr(\text{base substitution}) = 0.20$, $\Pr(\text{missing site}) = 0.10$.

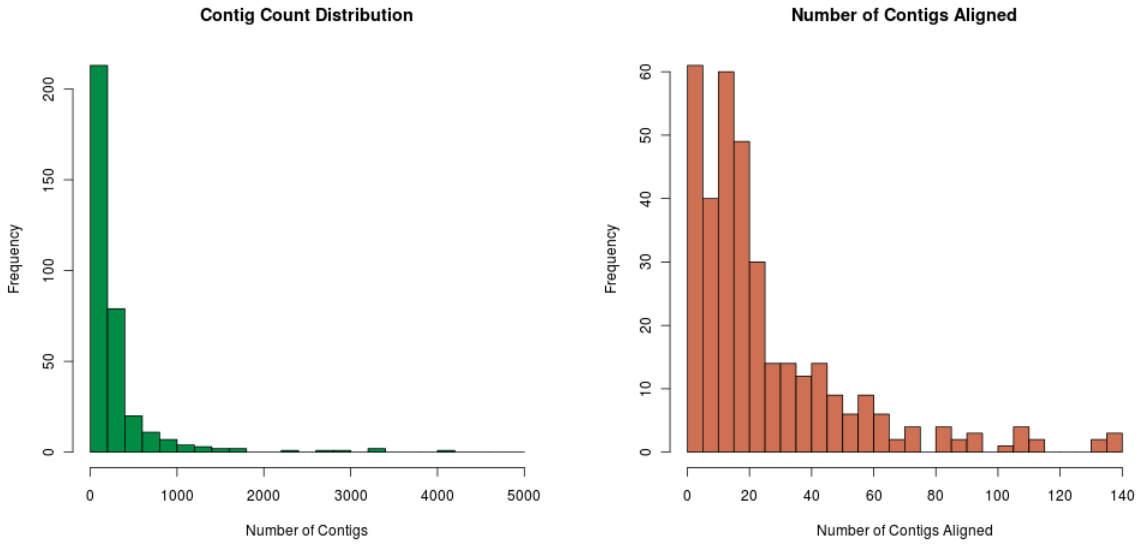
where l is the restriction fragment length. Error is applied independently to each restriction fragment, base of DNA sequence, and restriction site.

For each idealized de Bruijn graph with $k = 100$, contigs were extracted from the edges and written to a FASTA file. The FASTA file of contig sequences was input to the Contig-Optical Map Alignment Tool for alignment to the simulated optical maps. A permutation test with 500 trials was performed to evaluate the significance of the alignments, and all alignments with a p-value less than 0.05 were selected.

The distribution of contig counts per genome and the distribution of the number of uniquely aligned contigs under the error free setting is shown in Figure 3a. The number of contigs which align with significance is generally much less than the total number of contigs for a genome since most contigs are short, with no restriction sites. For a contig to be alignable, it must have two or more restriction sites. Two restriction sites guarantees the existence of an inner restriction fragment, which could potentially provide enough information to yield a unique alignment. The plot in Figure 4 shows the number of contigs out of those with two or more restriction sites that do not align uniquely. The clear outlier in the plot is *Nocardia farcinica* (NC_006361), which has 20 contigs not aligned uniquely out of 40 contigs with two or more restriction sites. These unaligned contigs are short, with an uninformative restriction pattern. For example, the contig represented by the edge between nodes 40 and 88 has a restriction pattern with fragments of length 220, 12, and 79 bp. The Contig-Optical Map Alignment Tool found four possible alignments where the inner fragment of the contig is matched to a different optical restriction fragment of length 12 bp in each of the alignments. The extra sequence information around the restriction site is also an exact match for all four possible alignments. This demonstrates how an uninformative restriction pattern can lead to many equally good alignments.

Figure 5 shows the distribution of the number of contigs aligned uniquely across error settings. As error increases, the mean number of contigs aligned uniquely decreases. However, the

alignments are fairly robust to this simulated error. In all cases, whenever a contig is deemed to align uniquely with significance, the alignment position is correct (within 0.1% of the true genomic position, as a percentage of true genome length). This provides further evidence that the Contig-Optical Map Alignment Tool is working as expected.



(a) Distribution of contigs per genome

(b) Distribution of aligned contigs per genome

Figure 3: Distribution of the number of contigs per genome and the distribution of the number of contigs which align uniquely with significance under the error-free setting.

Before running the full graph simplification on pipeline on these genomes, we investigated the number of shortest path closures between neighboring aligned contigs to the optical maps. Since the genomes are circular, we also check for a shortest path between the rightmost aligned contig and the first contig. While there are many gaps that have a unique shortest path closure, there are also some gaps that have many possible shortest path closures. We used the shortest path utilities of the Graph Simplification Tool to count the number of unique shortest edge paths between aligned edges without having to list each possible path. Figure 6a shows the distribution of shortest path closures. While many instances have a unique shortest path closure, there are some gaps that have over 1,024 shortest path closures. From the box plot in Figure 6b, it is clear there is even one gap that has 10^{14} shortest path closures. The number of possible shortest paths clearly depends on the graph structure, but also depends on the alignability of genome contigs. If there are fewer aligned contigs, the gaps between aligned contigs are larger, which means that the shortest paths will generally include more nodes, which increases the chance of encountering a tangled portion of the graph on a shortest path. These tangled portions of the graph is what allows for thousands of different shortest path closures.

We also investigated how often the true genomic path was among the set of shortest paths for a given gap closure between aligned contigs, as shown in Figure 7. When there is a unique shortest path closure, we see that it is the true genomic path path 73.0% of the time. When there are two or less shortest path closures, the true genomic path is in this set of shortest path closures in 67.0% of such instances. As the set of shortest path closures grows, the frequency with which the true

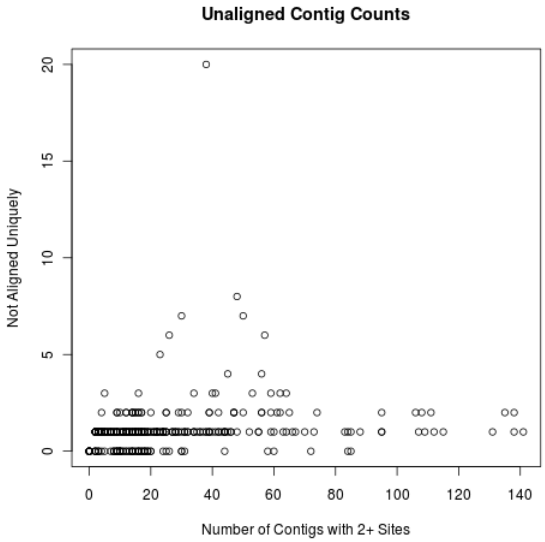


Figure 4: The number of contigs not aligned uniquely vs. the number of contigs with two or more restriction sites. Each point in this scatter plot represents one of the 351 prokaryotic genomes. A contig is included in the not aligned uniquely count if it has two or more restriction sites and it either goes unaligned or is aligned multiple times.

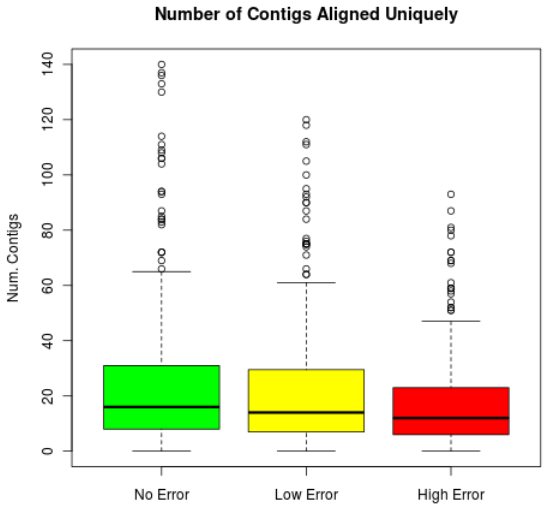
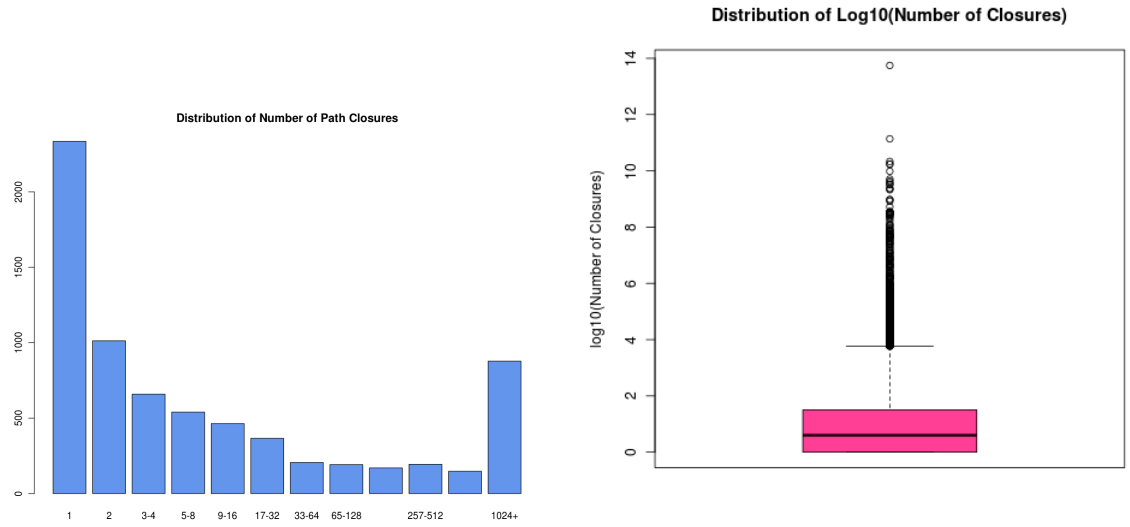


Figure 5: Distribution of the number of contigs aligned uniquely across error settings.

genomic path is a shortest path decreases. These percentages were calculated using alignments to the error-free optical map over all 351 prokaryotic genomes. A large number of shortest paths closures is indicative of a graph that has a tangled repeat structure between the aligned contigs, which makes it more likely that a shortcut exists through the graph which is not the true genomic path. This provides some intuition for why the correctness of the shortest path heuristic decreases as the number of shortest paths increases. This result provides good justification for limiting the graph simplification algorithm to those cases where a unique shortest path exists between to consecutively aligned contigs. The shortest path heuristic is often correct in these cases, and the heuristic fails to be correct more often when the number of shortest path closures is more than one.



(a) Distribution of number of shortest path closures

(b) Box plot of the number of shortest path closures

Figure 6: Distribution of the number of shortest path closures between consecutively aligned contigs.

After investigating the correctness of the shortest path heuristic, we ran the full pipeline on the database of prokaryotic genomes under the three error settings, and simplified the graphs using those shortest paths which were consistent with the optical map. Figure 8 shows that the paths used for graph simplification are often significant fractions of the reference genome. This should be expected, as the largest edges have the most informative restriction patterns and are often involved in the paths selected for graph simplification. Figure 9 illustrates the difference in sequence between selected shortest path and the true genomic path. The common sequence length between these paths is calculated by aligning edges of the respective paths, as discussed in section 4.2. The difference in sequence is computed as the difference between the sequence of the true genomic path and the selected path. Since the selected path is the shortest path, the true genomic sequence will be at least as large as the sequence of the shortest path, and so this difference should be non-negative. The fact that we observe only non-negative values (including zero values) for this difference gives additional assurance that the modified Dijkstra's algorithm

is working correctly.

Figures 10 through 12 summarize the improvement in N50 and reduction in graph complexity observed. N50 is defined as the largest contig size such that all contigs which are equal or greater in length cover at least 50% of the genome. These plots show that while some genomes show a significant improvement in N50 and reduction in complexity, other genomes remain largely unaffected. These results coupled with the results of Figure 8 illustrate that while the shortest path closures may include a significant portion of the genome as measured in genome sequence, these simplifications often leave many repeat nodes unresolved, yielding a small reduction in genome complexity.

Two layouts of assembly graphs both before and after simplification using error-free optical maps are provided in Figure 13 and Figure 14.

These simulations were run in parallel on the CBCB's Condor cluster. Six threads were used for the Contig-Optical Map Alignment Tool. Figure 15 shows the distribution of execution times for all simulations across all three error settings.

7 Future Work

The tools and pipeline presented in this project have several possible opportunities for improvement. First, the integration test on the database of prokaryotic genomes used a single restriction enzyme, BamHI. While this enzyme produced an informative restriction pattern for some genomes, it was not as informative for other genomes. An informative restriction pattern results in more contigs which align uniquely to the optical map. With more alignments, it is more likely that the Assembly Graph Simplification Tool will find unique shortest paths to use to simplify the assembly graph. Therefore, one improvement would be to simulate optical maps with different restriction enzymes and select the most informative optical map for each genome.

Another improvement would be to use multiple rounds of graph simplification. In the current implementation, one round of contig to optical map alignment is performed, followed by one round of graph simplification using the unique shortest path heuristic. After graph simplification, it is possible that newly formed contigs will align to the optical map. More specifically, in the path compression step, paths of nodes with degree one are combined into a new, single edge. This newly formed contig may now align to the optical map. In addition, after the simplifications made to the graph in the first round, it is possible that consecutively aligned contigs which previously had multiple shortest paths between them now have a unique shortest path. Multiple rounds of graph simplification could be applied until no further simplifications can be made to the graph.

Yet another potential improvement would be to combine information from paired reads with information from optical maps to simplify the graphs. Paired reads provide local information, while optical maps provide more global information. Together, optical maps with paired read information may do a better job at simplifying the assembly graph than either can alone.

Finally, it may be possible to use multiple optical maps together to simplify the assembly graph. Using multiple maps may help in situations where a single contig has multiple significant alignments, or when there are multiple candidate paths to close a gap between neighboring aligned contigs.

Figure 7: Percent of instances where the true genomic path is in the set of shortest paths between consecutively aligned contigs, as a function of the number of maximum number of closures. A given bar includes all instances where the number of shortest path closures is less than or equal to the label on the x-axis.

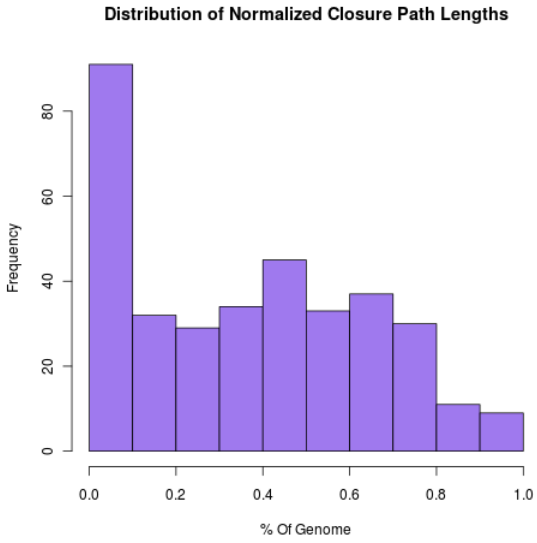


Figure 8: A histogram of the total shortest path length used to simplify the de Bruijn graph, as a percentage of the reference genome length

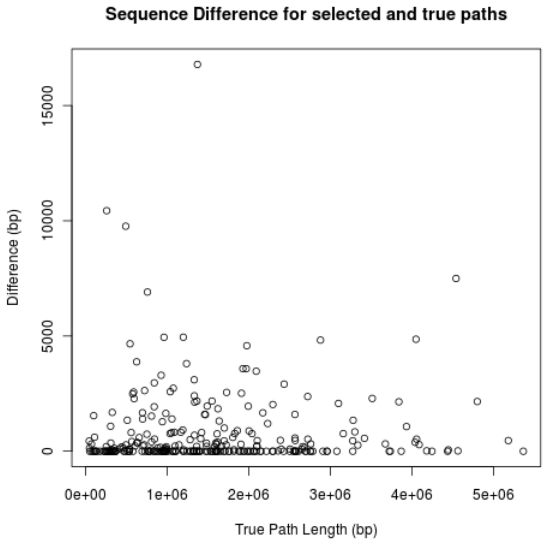


Figure 9: The amount of sequence missing in the selected shortest paths from the true genomic paths. This shows that the selected paths are highly accurate, since they differ from the true genomic paths by only a couple thousand DNA bases out of a couple million bases.

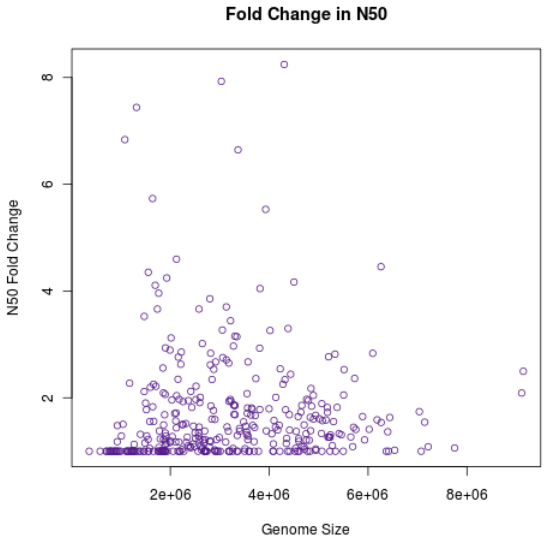


Figure 10: Fold change in N50 vs. genome sizes for simplification of assembly graphs using the error-free optical maps.

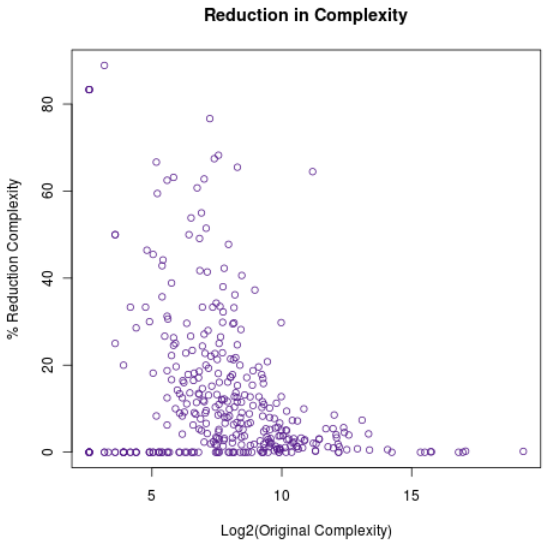


Figure 11: Percent reduction in complexity vs. logarithm of the original complexity using the error-free optical maps.

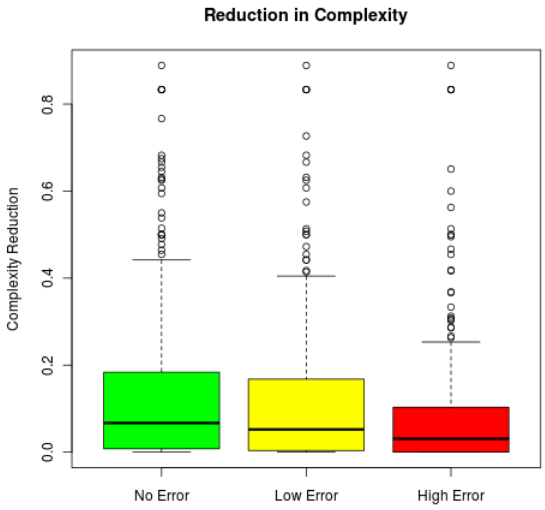
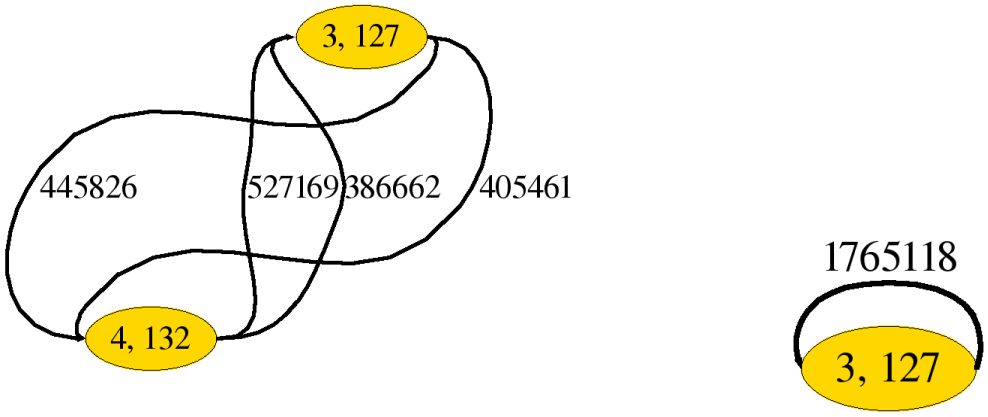
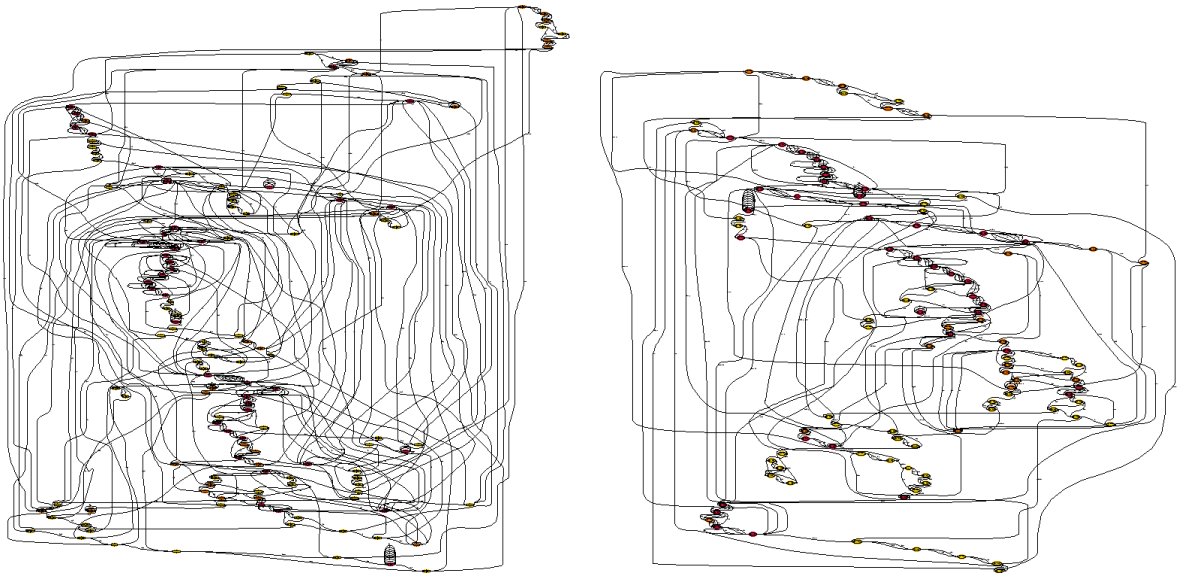


Figure 12: Box plot of fraction of complexity reduction across the three error settings.



(a) Before (b) After

Figure 13: Assembly graph simplification for *Pyrococcus abyssi* (NC_000868) using an error free optical map.



(a) Before (b) After

Figure 14: Assembly graph simplification for *Leptospira interrogans* serovar Copenhageni (NC_005823) using an error free optical map.

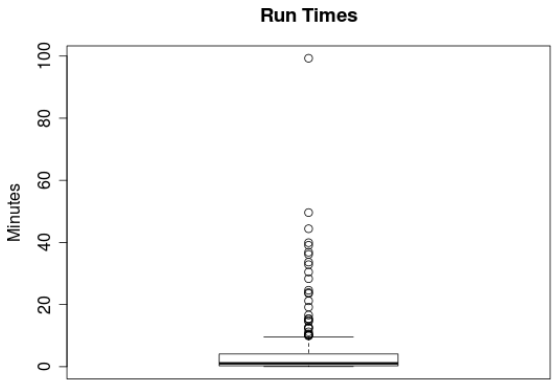


Figure 15: Box plot of execution times. The mean execution time was 4 minutes, and the median execution time was 1 minute.

8 Project Schedule & Milestones

- Phase I (Sept 5 to Nov 27)
  - Complete code for the Contig-Optical Map Alignment Tool
  - Validate performance of algorithm by aligning user-generated contigs to a user-generated optical map.
  - Write utility functions to import assembly graphs in .dot format using networkx.
- Phase II (Nov 27 to Feb 14)
  - Finish de Bruijn graph utility functions.
  - Complete code for the Assembly Graph Simplification Tool
  - Validate performance of Assembly Graph Simplification Tool on de Bruijn graph from known reference genome
  - Implement parallel implementation of the Contig-Optical Map Alignment Tool using OpenMP
- Phase III (Feb 14 to May 1)
  - Create pipeline for using Contig-Optical Map Alignment Tool and Assembly Graph Simplification Tool together on one dataset.
  - Test performance of the Contig-Optical Map Alignment Tool and Assembly Graph Simplification Tool with archive of de Bruijn graphs for reference prokaryotic genomes and artificial optical maps.
  - Compute reduction in graph complexities for each reference prokaryotic genome

9 Deliverables

- Source code for the contig alignment to optical map program
- Source code for the optical map simplification program
- Archive of simplified de Bruijn graphs for 351 prokaryotic genomes
- Simulation summary text files and log files.
- Final Report detailing software implementation and validation results.

References

[1] Carl Kingsford, Michael C Schatz, and Mihai Pop, *Assembly complexity of prokaryotic genomes using short reads.*, BMC bioinformatics **11** (2010), 21.

[2] Niranjan Nagarajan, Timothy D Read, and Mihai Pop, *Scaffolding and validation of bacterial genome assemblies using optical restriction maps.*, Bioinformatics (Oxford, England) **24** (2008), no. 10, 1229–35.

[3] P A Pevzner, H Tang, and M S Waterman, *An Eulerian path approach to DNA fragment assembly.*, Proceedings of the National Academy of Sciences of the United States of America **98** (2001), no. 17, 9748–53.

[4] Mihai Pop, *Genome assembly reborn: recent computational challenges.*, Briefings in bioinformatics **10** (2009), no. 4, 354–66.

[5] A. Samad, E. F. Huff, W. Cai, and D. C. Schwartz, *Optical mapping: a novel, single-molecule approach to genomic analysis.*, Genome Research **5** (1995), no. 1, 1–4.

[6] Michael C Schatz, Arthur L Delcher, and Steven L Salzberg, *Assembly of large genomes using second-generation sequencing.*, Genome research **20** (2010), no. 9, 1165–73.

[7] Joshua Wetzel, Carl Kingsford, and Mihai Pop, *Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies.*, BMC bioinformatics **12** (2011), 95.