ABSTRACT

Title of dissertation: ALGORITHMS FOR THE ALIGNMENT AND VISUALIZATION OF GENOME MAPPING DATA WITH APPLICATIONS TO STRUCTURAL VARIANT DETECTION

Lee M. Mendelowitz, Doctor of Philosophy, 2015

Dissertation directed by: Professor Mihai Pop
Computer Science Department

Optical mapping and nanocoding are single molecule restriction mapping systems for interrogating genomic structure at a scale that cannot currently be achieved using DNA sequencing methods. In these mapping experiments, large DNA molecules $\approx$ 500 kb are stretched, immobilized or confined, and then digested with a restriction endonuclease that cuts or nicks the DNA at its cognate sequence. The cut/nick sites are then observed through fluorescent microscopy and machine vision is used to estimate the length of the DNA fragments between consecutive sites. This produces, for each molecule, a barcode-like pattern comprising the ordered list of restriction fragment lengths

Despite the promise of the optical mapping and nanocoding systems, there are few open source tools for working with the data generated by these platforms. Most analyses rely on custom in-house software pipelines using proprietary software. In this dissertation we present open source software tools for the alignment and vizualization of restriction mapping data.

In this work we first present a review of the optical mapping and nanocoding systems and provide an overview of the current methods for aligning and assembling consensus restriction maps and their related applications.

Next, we present the Maligner software for the alignment of a query restriction pattern to a reference pattern. Alignment is a fundamental problem which is the first step in many downstream analyses, such as consensus map assembly or structural variant calling. The Maligner software features both a sensitive dynamic programming implementation and a faster but less sensitive index based mode of alignment. We compare the Maligner software to other available tools for the task of aligning a sequence contig assembly to a reference optical map and for aligning single molecule maps to a reference.

Next, we present a portable data visualization web application for visualizing pairwise alignments of restriction maps.

Finally, we present updates to the Maligner software to support partial alignments of single molecule maps, allowing for the clustering of compatible split map alignments to identify structural variants.

ALGORITHMS FOR THE ALIGNMENT AND VISUALIZATION OF GENOME MAPPING DATA WITH APPLICATIONS TO STRUCTURAL VARIANT DETECTION

by

Lee M. Mendelowitz

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:
Professor Mihai Pop, Chair
Professor David C. Schwartz
Professor Héctor Corrada Bravo
Professor Kasso Okoudjou
Professor Steve Mount, Dean's Representative

Preface

Of the six chapters in this dissertation, Chapter 1 has been published as a review paper and Chapter 2 has been prepared for publication. Both chapters have been slightly modified for this dissertation. Chapter 5 presents additional work on improving sequency assembly contiguity using paired reads which was submitted to the 2013 WABI conference.

Chapter 1

Mendelowitz, L., and M. Pop. "Computational methods for optical mapping." *GigaScience* 3.1 (2014): 33.

Chapter 2

Mendelowitz, L., Schwartz, D.C., and M. Pop. "Maligner: A fast ordered restriction map aligner." Submitted to *Bioinformatics*.

All of the source code for the methods described in this dissertation are available online on GitHub under the GNU General Public License.

The Maligner software presented in Chapter 2 and Chapter 4 is available at `https://github.com/LeeMendelowitz/maligner`. The MalignViz software for visualizing pairwise alignments is available at `https://github.com/LeeMendelowitz/malign_viz`. The SGA close-path software for removing false overlap edges from the string graph is available at `https://github.com/LeeMendelowitz/sga-close-path`.

Dedication

To my wife, Diana, for her unconditional love and support.

Acknowledgments

There are many people I would like to thank for their support and encouragement throughout my graduate school experience who have helped me grow intellectually, professionally, and emotionally. Thanks for helping making the last five years a special period of my life.

First and foremost, thanks to my advisor Mihai Pop for giving me the opportunity to work on such interesting problems and for his encouragement and professional guidance. Thanks to David C. Schwartz for pioneering and sharing such novel datasets which has motivated this work. Thanks to my committee members - Héctor Corrada Bravo, Kasso Okoudjou, and Stephen Mount - for their feedback and suggestions for improving this work.

I would like to thank all of my friends, colleagues, and housemates from the Applied Math & Statistics, and Scientific Computation program. Thanks to David Darmon, Stefan Doboszczak, Andrew Brandon, Joseph Paulson for the memories born at 8805 37th Ave. Thanks to David, Stefan, Joe, and Virginia Forstall for the lively discussions at our weekly AMSC lunches which have sustained us through our fifth year.

My officemates at the CBCB have also made this journey special. Special shout-outs to Joyce Hsiao, Kwame Okrah, Hisham Talukder, Joseph Paulson, Senthil Muthiah, Chris Hill, Victoria Cepeda, Justin Wagner, Ethan Ertel, Irina Astrovskaya, Mathieu Almeida for the frequent and very much necessary coffee breaks and spirited discussions and Jeremy Selengut for the interesting baseball discus-

sions. Thanks to Joe and Chris for encouraging various side projects, all of which have provided technical growth and have somehow managed to feed back into my dissertation research.

Thanks for Keith Hughitt and Trey Belew for our fun and very educational collaboration on CBCB Software. I think we have made things better for those that will follow.

Thanks to Eppley Recreation Center for providing such a wonderful fitness facility which was a perfect place to relieve stress, promote a positive attitude, and make new friends. Thanks to Kevin Willson and Shaun Edmonds for being an important part of my graduate school experience.

I would like to acknowledge my teammates past and present on the Ball Me Maybe softball team for giving me something to look forward in the middle of every week over the past several summers. It was nice to catch a break and play some fundamentally sound softball and, on those rare days where we didn't, make up for it with a well written game recap.

Thanks to the fantastic folks in baseball operations with the Washington Nationals for their support and encouragement over the past year and for making me feel like a part of the team even while I was away.

Finally, special thanks to my family for their love and support. To my parents Patti & Paul, my brother Neil, and my wife Diana. Diana, you have seen me at my craziest, most stressed and frustrated moments and were there for me when I needed it the most. Your love and support mean more to me than can possibly be expressed here.

Table of Contents

List of Figures

List of Abbreviations

API	application programming interface
AUC	area under the curve
BFS	breadth first search
BLAST	Basic Local Alignment Search Tool
D3	data-driven documents
DAG	directed acyclic graph
DFS	depth first search
DOM	document object model
DNA	deoxyribonucleic acid
HMM	hidden Markov model
HTML	hypertext markup language
HTTP	hypertext transfer protocol
JSON	JavaScript object notation
LOH	loss of heterozygosity
MAD	median absolute deviation
MLE	maximum likelihood estimate
MM	multiple myeloma
MVC	model - view - controller
Nmap	single molecule restriction map (from nanocoding)
OLC	overlap - layout - consensus
Rmap	single molecule restriction map (from optical mapping)
ROC	receiver operating characteristic
SGA	string graph assembler
SVG	scalable vector graphics
SNV	single nucleotide variant
TIGR	The Institute for Genomic Research
WSGI	web server gateway interface

Chapter 1: Introduction

1.1 Introduction

Prior to the advent of cheap high-throughput sequencing technologies and corresponding analytical tools, such as genome assemblers, genomic mapping approaches provided scientists with a first glimpse at the large-scale structure of the chromosomes of organisms. Among the many competing technologies for mapping (e.g., see [1] for a review of other approaches), the optical mapping technology [2] provided for the first time, the ability to identify the location and order of restriction sites along single DNA molecules $\approx$ 500 kb in length, thereby enabling the efficient construction of accurate genome-scale restriction maps. Since the initial demonstration of this system in the yeast *Saccharomyces cervisiae*, optical mapping has been used to validate and assist the reconstruction of multiple genomes ranging from bacteria [3] to the human genome [4]. This technology has also shown been demonstrated to be a powerful tool for comparative genomics allowing the detection of structural variants within genomes [4, 5]. Recently, an evolution of the optical mapping technology nanocoding was developed [6], promising higher accuracy and throughput than the original optical mapping system.

Before describing the computational approaches for analyzing optical (or nanocod-

ing) mapping data, we will briefly describe the key characteristics of these data. The mapping experiment begins with large DNA molecules (hundreds of thousands of base-pairs) which are immobilized on a surface, digested with one or more restriction enzymes, and stained with a fluorescent dye (Figure 1.1. The series of cuts or nicks produced by the restriction enzyme are detected by imaging the immobilized DNA, and the length between consecutive cut sites is estimated by integrating the fluorescence intensity. The resulting data is an ordered series of fragment lengths, corresponding to the estimation by machine imaging of the distances between nicks or cuts. These data commonly contain a number of errors, such as inaccurate estimates of restriction fragment size (due to non-uniform fluorescent staining), missing or extra restriction sites, or missing small restriction fragments (due to limitations of the experimental and/or imaging components of the system). Furthermore, these data only span individual DNA molecules. Information from multiple overlapping DNA molecules that originate from the same genomic location needs to be combined/assembled in order to construct chromosome-wide maps. The map assembly process can also correct many of the above-mentioned errors. Throughout the following we will refer to single DNA molecule optical maps (the restriction fragments sized and ordered) as *Rmaps* and to the consensus maps of the assembled Rmap contigs as *consensus optical maps*.

It should be obvious from this brief description that computational analysis software must be an integral part of the generation and use of the optical mapping data. After the machine vision software necessary to generate the initial raw data, computational tools are necessary to align to each other and assemble together in-

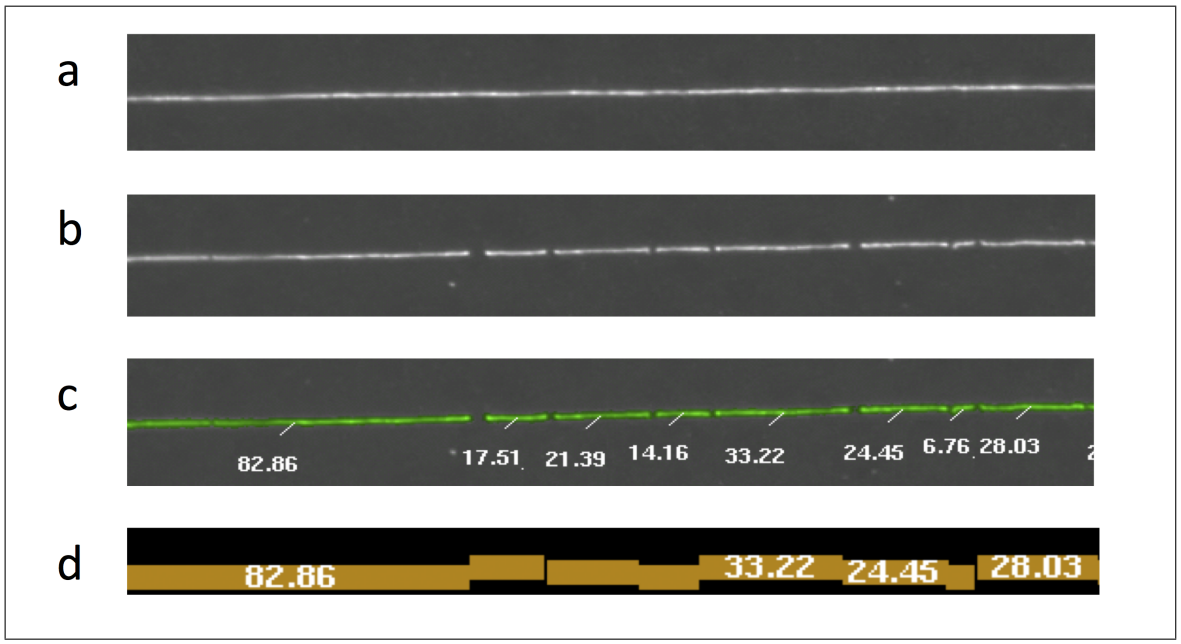

Figure 1.1: **Optical Mapping Experiment Overview** In an optical mapping experiment, stretched DNA molecules are deposited on a charged glass surface using an array of microfluidic channels (a) and digested with a methylation-insensitive restriction enzyme that cuts the DNA at specific sequence based recognition sites (b). The stretched DNA relaxes around the cut sites, but in the process, small restriction fragments can be lost through desorption. The DNA molecules are then stained with fluorescent dye and imaged. Restriction fragments are identified with machine vision and the fragment lengths are estimated by integrating fluorescent intensity (c). For each molecule this produces an ordered listing of restriction fragment lengths known as an Rmap (d).

dividual Rmaps, as well as to align the assembled maps to each other (e.g., when identifying structural variants), or to genomic sequences (e.g., to validate or assist the genome assembly process). Below we review the key principles underlying these operations as well as published software tools for using and analyzing optical mapping data.

One fundamental problem in using genome maps is the task of aligning restriction maps, either to each other, or to a genome sequence. The alignment scoring functions must take into account the error characteristics of the mapping experiment, including fragment sizing error, missing and false restriction sites, false restriction sites, and as well as missing fragments (Figure 1.2). Dynamic programming algorithms for alignment can accommodate missing restriction sites, false restriction sites, and missing fragments by allowing for different alignment extensions (Figure 1.3). Alignments methods must accommodate some sizing error since an experimental Rmap fragment size will rarely be an exact match to the corresponding fragment in another Rmap or in the reference genome. For this reason, alignment scoring functions allow for small differences, but penalize large differences in restriction fragment size. There are several different flavors of the alignment problem: (i) The alignment of individual Rmaps to detect overlaps a critical step for the *de novo* assembly of an optical consensus map, (ii) the alignment of individual Rmaps to an optical consensus map to call structural variants, or (iii) the alignment of in silico restriction maps derived from contigs or scaffolds from sequence assembly to a consensus optical map. Here we review several of the published alignment methods, as well as a method for determining alignment significance.

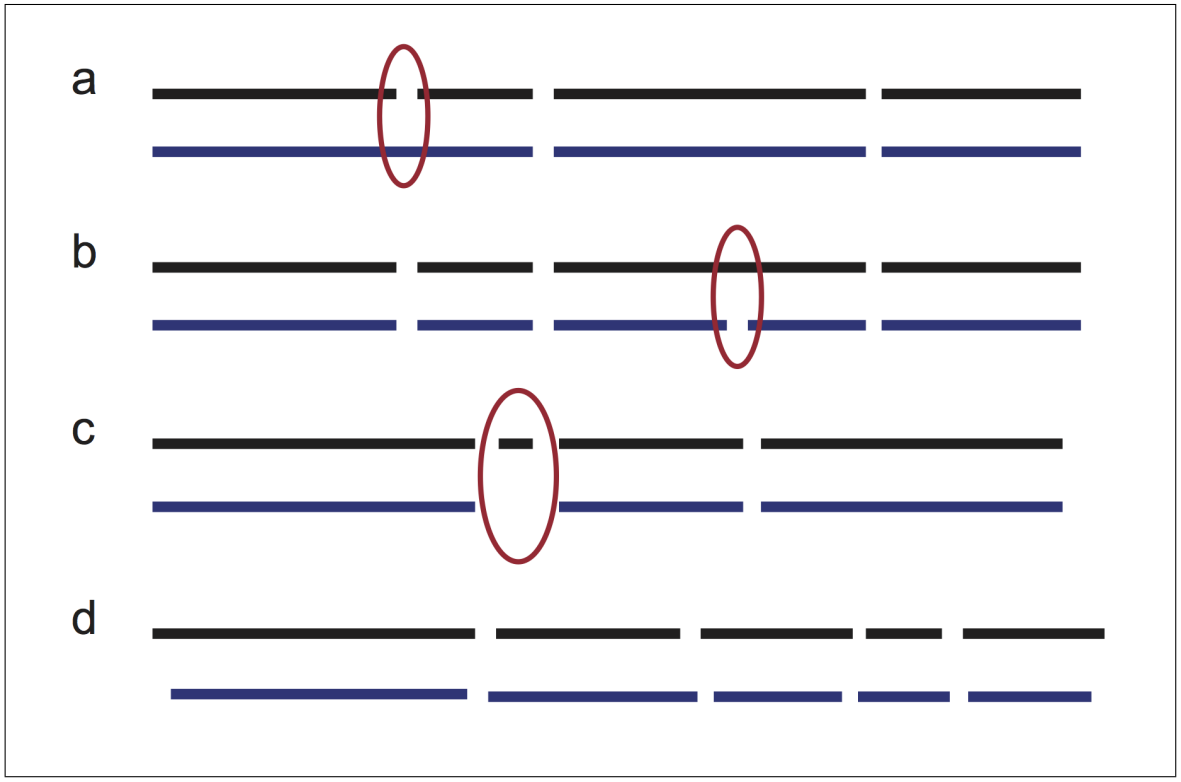

Figure 1.2: **Optical Mapping Experimental Errors** Experimental errors in the optical mapping of individual molecules include (a) missing enzyme cut sites due to incomplete digestion, (b) extra enzyme cut sites due to random breakage of the DNA molecule, (c) missing small fragments due to desorption, and (d) sizing error due to noise in measurements of fluorescence intensity. The ideal, error-free map is shown in black, and the experimentally observed map is shown in blue.

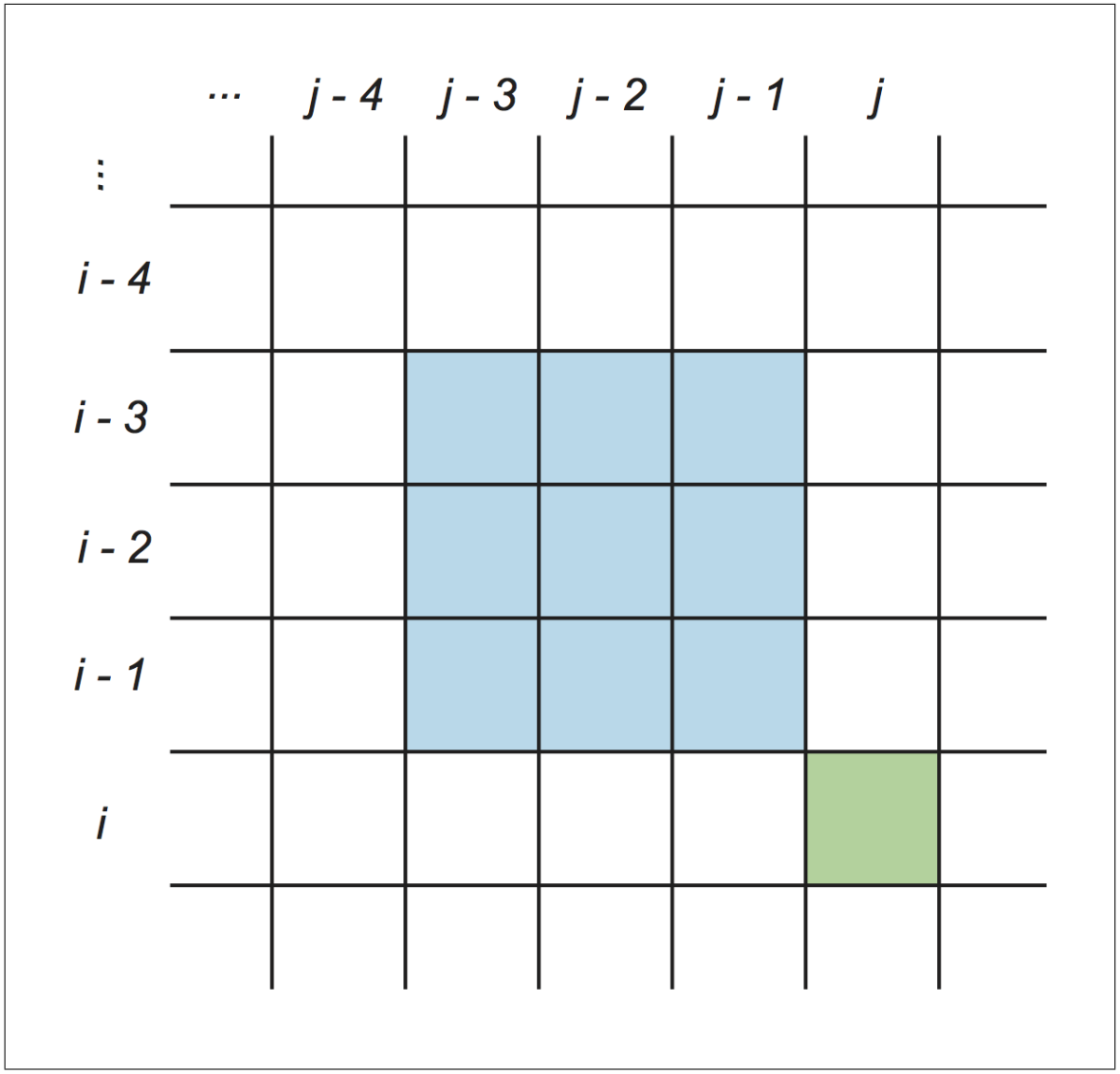

Figure 1.3: **Alignment Dynamic Programming** Optical map aligners, such as the aligner by Valouev [7] and SOMA [8] use dynamic programming to compute the optimal scoring alignment. Let cell (i, j) in the dynamic programming matrix, colored in green, represent the optimal partial alignment of the query map of m fragments through the ith restriction site to the reference map of n fragments through the jth restriction site such that site i is matched to site j. To allow for unmatched restriction sites in the alignment, the score for cell (i, j) is determined by attempting to extend previously computed alignments in an adjacent δ^2region of the matrix, colored in blue. This allows for up to $\delta - 1$ consecutive unmatched sites in both the query and the reference. The alignment method is then $\mathcal{O}(\delta^2 mn)$.

1.2 Alignment Methods

Valouev et al. [7] have developed an alignment algorithm for both finding overlaps between two optical maps and for aligning an optical map to a reference map. The scoring function is defined as a log likelihood ratio test for a model that makes the following assumptions: the size of genomic restriction fragments are distributed exponentially; the observations of each restriction site in an optical map are independent Bernoulli processes; the number of false cuts in a given genomic length is a Poisson process; and fragment sizing error is distributed normally with mean zero and variance that scales linearly with the true fragment size. A separate normal sizing error model is used for fragment sizing error for small restriction fragments below a specified threshold. Lastly, the authors put a bound on the number of restriction fragments allowed between consecutively matched restriction sites, leading to a dynamic programming algorithm which runs in time proportional to mn where m and n are the number of restriction sites in the aligned maps (Figure 1.3). This alignment tool has successfully used for overlapping Rmaps as part of *de novo* optical map assembly [9].

SOMA [8] is another alignment tool designed specifically for aligning sequence contigs from a genome assembly to a consensus optical map. First, the contigs are converted into an in silico restriction map by noting the location of the enzymes recognition sites within the contig sequence. Next, the software finds good placements of contigs to the optical map using dynamic programming algorithm. Lastly, SOMA uses this set of good alignments to select a layout of non-overlapping align-

ments to the consensus map, in effect constructing a genome-wide scaffold of contigs. The dynamic programming algorithm for alignment uses a chi-squared scoring function to penalize restriction fragment sizing error and a fixed cost penalizing each unaligned site in both the reference map and contig in silico map. The statistical significance of alignments is determined by performing a permutation test for each contig with sufficient restriction sites. For contigs with multiple significant alignments, an F-test is used to further filter out secondary alignments by comparing the ratio of the best alignments chi-square score to that of each the secondary alignment. Finally, SOMA uses a scheduling algorithm to find non-overlapping placements of the contigs to the optical map. The goal is to find the maximum weight layout, where each contig placement is weighted by the match significance, given as the p-value from either the permutation test or the F-test. Several different scheduling algorithms are considered, including a greedy algorithm which prioritizes the placement of contigs with the highest match significance, provided it does not overlap the best scoring scheduling of the remaining fragments (GREEDY); an expensive algorithm which enumerates all possible layouts using depth-first search with pruning of low scoring layouts (ASTAR); and a simple, heuristic approach which places contigs in descending order of match significance such that there are no overlaps (match filtering).

TWIN [10] is a recently developed tool for aligning in silico contigs to a consensus optical map using an FM-Index. TWIN converts contigs into a restriction pattern by performing an in silico of the contig sequence. An FM-Index is constructed on the ordered integer sequence of restriction fragment lengths given by

the consensus optical map, which allows for the efficient search for exact matches of patterns of n consecutive fragments. Once the FM-index is constructed, the run time is proportional to the number of fragments in the contig. To account for fragment sizing error, TWIN modifies the FM-Index backward search algorithm to backtrack along possible alignment choices that are consistent with the current fragment in the query. To reduce computational effort during the backtrack procedure, TWIN relies on an integer wavelet tree auxiliary data structure which allows the algorithm to focus on just those optical fragments within the current FM-index interval, that are consistent with the current query fragment. A drawback of this algorithm is its inability to handle unmatched restriction sites such as those caused by missed fragments or restriction sites.

1.2.1 Significance of Alignments

All alignment algorithms face the challenge that under any alignment scoring scheme, a given query restriction pattern may have multiple good quality alignments to the reference or consensus map. In cases when the alignment score depends on the number of restriction fragments and length of the query sequences, as in [7], a simple alignment score threshold is not sufficient to distinguish between ambiguous alignments. Sarkar et al. [11] observe that the optimal alignment scores of a query restriction pattern to permuted versions of the true reference map are highly correlated. In other words, the best alignment scores for spurious alignments depend on properties of the query map itself. The authors model the distribution of alignment

scores for spurious alignments so they can use a map specific cutoff for determining alignment significance. In particular, the authors model the optical alignment score under the null hypothesis that the alignment is spurious using multiple linear regression on the number of query map fragments N, the map length L, and their product NL. The standard deviation of the optimal alignment score against a random spurious reference is modeled as a linear function of the mean optimal alignment score. The regression model is fit by aligning a set of query maps to a single permuted reference map, avoiding the computational bottleneck of performing a permutation test for each aligned query map against a set of permuted reference maps. Sarkar et al. also use logistic regression to predict the probability that a query map will have an alignment to a reference genome given the query map's information content. This logistic model can be used to filter out query maps that are unlikely to align, saving computational resources. The authors demonstrate how an iterative optical map assembly algorithm performs better when using optical map alignments that are deemed significant using query-specific thresholds.

1.3 Algorithms for Optical Map Assembly

An optical mapping experiment produces a restriction map (Rmap) for a collection of DNA molecules on the order of 500 kb in length. As in shotgun sequencing, these molecules are produced by randomly shearing the DNA from the organism of interest. It is therefore necessary to assemble the Rmaps in order to produce a more contiguous, higher quality consensus optical map. A consensus map is formed by

computing a consensus restriction pattern for Rmaps that share compatible patterns and are therefore are highly likely to have originated from the same place in the genome. Each assembled consensus restriction pattern is known as an optical map contig. Each optical map contig is characterized by both its consensus restriction pattern and a layout that provides the position and orientation of each Rmap used in its construction.

The Gentig algorithm [12] is the first published method for the assembly of consensus optical maps for shotgun optical mapping experiments. The method uses a Bayesian formulation which seeks to maximize the *a posteriori* estimate of the consensus map assembled from the Rmaps. A prior probability distribution H on the consensus map is selected as a decreasing function of contig length, giving a prior bias for shorter (i.e., more assembled) contigs. This prior helps select assemblies that do a better job of overlapping and incorporating the experimental optical maps. Contigs are built by greedily merging the two best overlapping Rmaps or contigs, where overlaps are computed using dynamic programming. Overlaps are only considered if the match scores better than a specified threshold that controls for false overlaps between two unrelated restriction maps. Gentig constructs its prior and overlap scores using a probabilistic model which accounts for the errors inherent in optical mapping, including sizing errors, missing cut sites due to partial enzyme digestion, and false cut sites due to imaging artifacts.

While Gentig has successfully been used to assemble bacterial genomes, it does not scale well to larger genomes where the number of input Rmaps is large. Procedures have been developed to use Gentig in an iterative fashion for *de novo*

optical map assembly of larger genomes by first randomly partitioning the input Rmaps into separate groups, and then running Gentig independently on the groups to produce a set of contigs. Since there may be duplicate or overlapping contigs between the independent assemblies, Gentig is used to assemble all of the contigs together to remove any redundancy, yielding a set of seed contigs. The input Rmaps are then aligned to the seed contigs as a means to cluster the Rmaps based on similarity, and then these piles of Rmaps are independently assembled using Gentig to produce a new set of contigs. This process is repeated for several iterations, producing a final set of contigs. Variations of this method have been used to build *de novo* optical map assemblies for *Leishmania major* Friedlin (34.7 Mb) [13], *Oryza sativa* (rice, 382 Mb) [14], *Zea mays* L. (maize, 2.5 Gb) [15], and *Melopsittacus undulatus*, (budgerigar parakeet, 1.2 Gb) [16].

Valoeuev et al. [9] have implemented an optical map assembler based on the overlap - layout - consensus (OLC) paradigm of sequence assembly. The overlap graph consists of Rmaps, represented as nodes, and significant overlaps, represented as edges between the Rmaps. First, pairwise overlaps are constructed between all of the Rmaps. This is the most computationally intensive step and is performed on a computing cluster. High scoring overlaps are selected to construct the overlap graph. The graph is cleaned up by removing potential false overlaps by identifying paths through the overlap graph that are weakly supported. The set of edges is further refined by removing any edges which disagree with higher scoring information. Additional false edges are removed from the graph by considering edges that form a path between two nodes for which there is no alternative path with a consistent

distance. Lastly, chimeric maps are identified as local articulation nodes. Valouev et al. demonstrate their optical map assembler by producing consensus maps for *Yersinia pestis* KIM, *Escherichia coli* K12, *Thalassiosira pseudonana*, *Oryza sativa* ssp japonica (rice), and Homo sapiens.

1.4 Applications for Structural Variation Detection

A promising application of optical mapping technology is the characterization of structural variation within genomes. Optical mapping data span much longer genomic ranges than commonly achievable mate-pair sizes, and thus have the ability to detect large-scale variants that cannot be detected using paired end reads.

Teague et al. [4] have successfully used optical maps to detect structural variants in four normal human samples compared to the human reference genome, detecting both small variants, such as missing or extra enzyme cut sites, and as well as large-scale insertions, deletions, and inversions, ranging from thousands to millions of base -pairs in size. Variants were detected by first constructing an optical consensus map for each sample using an iterative assembly strategy initially guided by an in silico map of the human reference. First, the Rmaps were aligned to the reference in silico map as a means to cluster the Rmaps with similar restriction patterns. Next, each cluster of maps was assembled using the Gentig software to produce a contig (i.e., consensus restriction pattern) for the cluster. The assembled contigs from all of the clusters were used in place of the reference in the second iteration, and the Rmaps were again aligned and assembled to produce a new set

of Rmap contigs. This process was repeated for eight iterations, yielding a high quality consensus optical map for that sample. Structural variants between each assembled sample and the human reference were called by looking at the depth of Rmap coverage supporting each variant. A p-value was assigned to each variant call for missing cuts and extra cuts through a Binomial test and for indel calls using a Z-test derived from the sizing error model. The paper demonstrates that each of the four samples has hundreds of unique structural variants that are neither present in the other samples nor the human reference.

Optical mapping has also been used to characterize structural variants in oligodendroglioma [17], a type of brain cancer. A similar iterative assembly strategy was used to assemble a consensus optical map for two different tumor samples, HF087 and HF1551. Over 1,000 structural variants were called between each sample and human reference. In addition, a hidden Markov model (HMM) was trained on normalized Rmap coverage to determine the copy number at each chromosomal location. Loss of heterozygosity (LOH) events in which one copy of the chromosome is lost were observed in chromosomes 1, 14, 19, and 21. In addition, coverage analysis of Rmaps obtained from two adjacent slices of sample HF1551 revealed distinct LOH events for each slice, suggesting that these adjacent slices of the same tumor actually evolved from different cancer cell clones.

1.5 Genome Assembly

Consensus optical maps provide long-range information over the length of a genome that can be used to aid in genome sequence assembly and validation. Assembly algorithms are graph based, where sequences are represented as nodes and overlaps between sequences are represented as edges. Each path through the assembly graph generates a sequence, and each possible path gives a possible reconstruction of the genome. Genomic repeats introduce nodes that must be traversed multiple times, thereby tangling the assembly graph. AGORA [18] presents a method for guiding genome assembly to resolve repeats using optical maps by selecting the correct path among exponentially many paths consistent with the set of reads. AGORA works by first aligning long sequence contigs extracted from de Bruijn graph edges to the consensus optical map. All contigs with a unique placement give a genome wide scaffold (i.e., layout). Gaps in the scaffold are filled by greedily selecting a path in the de Bruijn graph between consecutively aligned contigs that is consistent with the restriction pattern of the optical map, thereby resolving repeats. The path is selected using a bounded depth-first search. Simulations with AGORA on error-free de Brujin graphs for bacterial genomes and simulated optical maps suggest that high quality consensus optical maps can accurately improve assembly contiguity.

Xavier et al. [19] have demonstrated how optical consensus maps can be used to assess assembly accuracy when selecting from a set of candidate assemblies constructed under different assembly parameter settings. In a de Bruijn graph assembly, a critical parameter is the k-mer length, which controls the length of the overlap used.

Generally, a larger k-mer setting results in a more aggressive assembly that improves assembly contiguity at the expense of accuracy, while a smaller k-mer setting gives a conservative but accurate assembly at the expense of contiguity, as the de Bruijn graph has branches for genomic repeats of length k. Xavier et al. built multiple *de novo* assemblies for Methicillin-resistant Staphylococcus aureus (MRSA) using different assemblers and a wide range of k-mer settings. Mis-assemblies wereThe authors detected mis-assemblies by finding contigs that have a split alignment to the optical consensus map,. The authors then selected the assemblies with highest contiguity (i.e., with the most resolved repeats), but which did not exhibit any mis-assemblies with respect to the optical map. Furthermore, Optical maps have also proven useful for validating existing genome assemblies and characterizing mis-assemblies. In the case of the *Oryza sativa* (rice) genome [14], an optical consensus map was used to compare the quality of two independently constructed assemblies, one by TIGR and the other by the International Rice Genome Sequencing Project. Consensus optical maps have also been used as part of the Assemblathon 2 competition [20] to assess the quality of *de novo* assemblies for a budgerigar (*Melopsittacus undulatus*) a Lake Malawi cichlid (*Maylandia zebra*), and boa constrictor (*Boa constrictor constrictor*). The consensus optical maps were iteratively assembled using Gentig. Assembly quality was assessed by aligning sequence scaffolds constructed from paired-end reads to the optical consensus map under different levels of alignment stringency. Scaffolds that globally align to the optical map under the most restrictive setting are considered correct, while scaffolds that only have local alignments are considered to have mis-assemblies.

1.6 Conclusions

In this chapter we have reviewed algorithms and tools for processing optical mapping data (alignment and assembly) and for using these data to identify structural variants, and to guide or validate genome assemblies. Due to the long range information provided by optical mapping data (potentially spanning hundreds of kilo-base-pairs or more) and the relatively complex and error-prone approaches for constructing long mate-pair libraries in the context of modern sequencing technologies, optical mapping data hold tremendous promise in supplementing and or even replacing sequencing data in the study of chromosomal rearrangements.

Despite this promise, relatively few methods exist for analyzing and using optical mapping data, and even fewer are available in effective publicly-available software packages. While Gentig has successfully been used to assemble consensus optical maps for bacterial genomes, it does not scale well to large genomes and the software is not freely available. Beyond AGORA, which is a proof of concept implementation, no genome assemblers can make use of optical mapping information. Furthermore, there are virtually no tools available for using optical maps to characterize structural variants. The alignment tools reviewed above could and have been used for this purpose, but only through the manual curation of the raw alignment output rather than through the use of specialized structural variant discovery tools. There is, thus, a critical need for the continued development and public release of software tools for processing optical mapping data, mirroring the tremendous advances made in analytical methods for second and third generation sequencing data.

The methods presented in the following chapters of this dissertation address the gap in available tools for working with genomic mapping data.

Chapter 2: Efficient Alignment of Molecular Maps using Maligner

2.1 Introduction

Optical mapping [2] and newer nanocoding technologies [6] capture long range genomic information over the length of hundreds of kilobases by observing the location of restriction sites. By providing long range, coarse information about the genomic structure, optical mapping data provides complementary information to that of DNA sequencing, which in comparison provides short range information with base pair resolution. Optical mapping data has been used to correct sequence mis-assemblies [14], assess the quality of sequence assemblies [20], and detect large structure variants in normal [4] and cancer [17] genomes that can not be reliably detected using current paired end sequencing protocols, due to short reads and the repetitive nature of the human genome [21].

Despite such promise, very few software tools are freely available for working with genomic mapping data for large genomes. SOMA [8] is an open-source software tool for aligning assembled sequence contigs to an optical map but does not scale to large genomes and often gives incorrect contig placements due to the greedy nature of its alignment algorithm. TWIN [10] is a recently developed tool for efficiently aligning sequence contigs to an optical map, but as we show, it does not

allow for alignments which have unmatched sites, limiting its applicability towards experimental datasets.

Here we present Maligner, an open-source software package for aligning single molecule restriction maps (Rmaps) and *in silico* of contigs from a sequence assembly to a reference restriction map at speeds that are comparable or faster than currently available tools. Maligner has two modes of alignment, one which uses traditional dynamic programming (malignerDP) and an index based mode of alignment) that runs orders of magnitude faster but is more stringent in the alignments that it accepts (malignerIX). In addition, we present a novel method for normalizing the alignment scores across queries based on computing the MAD across the best random alignments, which allows for the selection of an alignment score cutoff that is applicable across queries, thereby obviating the need for a computationally expensive permutation test for determining alignment significance.

2.2 Background

The optical mapping system uses methylation insensitive restriction enzymes to digest single molecules, which are then stained with fluorescent dye and imaged. The images are processed using machine vision to identify the individual molecules and estimate the fragment lengths. For a full description of the optical mapping system, see Section 1.1.

Consider a DNA molecular of length L bp which is digested with a restriction enzyme which cuts (or nicks) the DNA at at n integral positions p_0, p_1, ..., p_{n-1}

where p_i represents the zero-based base pair location of the ith cut site. We can represent this restriction pattern by an ordered listing of $n+1$ fragment lengths: $f_0, f_1, \ldots, f_n$ where $f_0 = p_0$, $f_i = p_i - p_{i-1}$ for $0 < i \leq n-1$ and $f_n = L - p_i$. Since the DNA molecule is produced by random shearing of chromosomal DNA, the fragments f_0 and f_n which appear at the start and end of the molecule are not bounded by restriction sites at both ends. We refer to these fragments as boundary fragments. On the other hand, $f_1, f_2, \ldots, f_{n-1}$ are interior fragments, as they are bounded on both ends by restriction sites.

A *restriction map* $\mathcal{M}$ is given by its ordered listing of fragments $[m_0, m_1, \ldots, m_n]$. The number of restriction fragments in the map is given by $|\mathcal{M}|$. In this case, $|\mathcal{M}| = n+1$.

A *chunk* is an ordered listing of consecutive restriction fragments from a single map $\mathcal{M}$. For example: $[m_1, m_2, m_3]$ is a chunk of three consecutive fragments from restriction map $\mathcal{M}$. We represent a chunk more concisely by the triple $c = (\mathcal{M}, s, e)$ which corresponds to the consecutive fragments m_i from $\mathcal{M}$ where $i \in [s, e)$. For example, $c = (\mathcal{M}, 1, 4)$ corresponds to the consecutive fragments $[m_1, m_2, m_3]$. A chunk $(\mathcal{M}, s, e)$ is a boundary chunk if $s = 0$ or $e = |M|$.

Two chunks $c_1 = (\mathcal{M}_1, s_1, e_1)$ and $c_2 = (\mathcal{M}_2, s_2, e_2)$ are *adjacent* if $\mathcal{M}_1 = \mathcal{M}_2$, and $e_1 = s_2$ or $e_2 = s_1$. The number of fragments in a chunk is denoted as $n(c) = e - s$, and the number of interior sites in a chunk is $e - s - 1$. The length of a chunk is given by the sum of the lengths of the restriction fragments in the chunk: $l(c) = \sum_{m_i \in c} m_i$. A chunk is empty if $e = s$, meaning the chunk contains zero fragments and has zero length.

A *matched chunk* is an ordered pair of chunks. We refer to the first chunk as the *query chunk* and the second chunk as the *reference chunk*. Given query chunk $c_q = (\mathcal{M}_q, s_q, e_q)$ and reference chunk $c_r = (\mathcal{M}_r, s_r e_r)$, the matched chunk is the ordered pair $mc = (c_q, c_r)$. A matched chunk is a boundary chunk if either c_q or c_r are boundary chunks. Two matched chunks $mc_1 = (c_{q_1}, c_{r_1})$ and $mc_2 = (c_{q_2}, c_{r_2})$ are adjacent if both c_{q_1} and c_{q_2} are adjacent and c_{r1} and c_{r2} are adjacent.

An alignment of a query map $\mathcal{Q}$ to a reference map $\mathcal{R}$ is given by an ordered listing of matched chunks $\mathcal{A} = [mc_1, mc_2, \ldots, mc_k]$ where:

1. All of the query chunks are from map Q and the reference chunks are from map R
2. The matched chunks are adjacent
3. The starting indices of the reference chunks are monotonically increasing
4. The starting indices of the query chunks are monotonically increasing (for forward alignments) or monotonically decreasing (for reverse alignments).

We say that restriction site i in the query is aligned to restriction site j in the reference if the alignment has a non-boundary matched chunk (C_q, C_r) with $C_q = (M_q, s_q, e_q)$ and $C_r = (M_r, s_r, e_r)$ where: (i) $s_q = i$ and $s_r = j$ or $e_q = i$ and $e_r = j$ for forward alignments or (ii) $s_q = i$ and $e_r = j$ or $e_q = i$ and $s_r = j$ for reverse alignments. In the context of an alignment, we refer to the interior sites of a query chunk or reference chunk as unmatched sites. An example of an alignment using this notation is shown in Figure 2.1.

2.3 Results

In this section we present results demonstrating the utility of the M-Score statistic in discerning correct alignments from spurious alignments. Next, we compare the Maligner software to other available software tools by aligning *in silico* digested contigs from an *E. coli* K12 sequence assembly to both an *in silico* digest of the reference sequence and an optical map. We also align *E. coli* K12 Rmaps to the *in silico* reference map. Lastly, we demonstrate our methods on a large genome by aligning both sequence contigs and Rmaps to a budgerigar optical map (*Melopsittacus undulatus*), an Australian parakeet.

2.3.1 M-Score for Alignment Significance

Maligner scores alignments using an additive scoring function on each matched chunk given by Equation 2.1, with fixed costs for interior unmatched sites in the query chunk and reference chunk and cost for sizing difference between the query and reference chunk. The score for an alignment is given by the sum of the scores of the matched chunks of the alignment. This scoring function represents an "edit distance" between the query and the reference, with lower scores corresponding to alignments with greater similarity between the query and reference.

Since the scoring function is additive, the score of a query's best alignment depends on the number of fragments, its length, and quality of the query (i.e., the sizing error, site cut rate, and desorption of small fragments). Therefore, one cannot select a simple cutoff that applies to all queries for selecting significant alignments.

The M-Score, discussed in more detail in section 2.5.1.1, seeks to normalize the scores for each query based on the distribution of alignment scores for its best non-overlapping alignments. This allows for the selection of a common cutoff that applies across queries.

We assessed the performance of M-Score for selecting correct alignments on a simulated dataset under three different error settings. We simulated Rmaps from the human reference using enzyme BamHI by selecting map length uniformly at random from $[100, 500]$ kb, selecting a genomic location and orientation at random, simulating a cutting pattern as a Bernoulli process with enzyme efficiency p, applying an Rmap scaling factor $\sim 1+\mathcal{N}(0, \sigma^2)$ to model variability in molecular stretch, and finally adding fragment sizing measurement error $q \sim \mathcal{N}(r, (\alpha r)^2)$ with parameters $p = 0.85$, $\sigma = 0.02$, $\alpha = 0.02$ under the low error setting, $p = 0.75$, $\sigma = 0.05$, $\alpha = 0.05$ under the medium error setting, and $p = 0.65$, $\sigma = 0.05$, $\alpha = 0.10$ under the high error setting. We simulated 1000 Rmaps under each error setting from both the human reference and a permuted version of the human reference, requiring a minimum of 10 restriction fragments.

We aligned these simulated Rmaps with malignerDP and selected the best scoring alignment for each query against the human reference. From this set of alignments we consider an alignment of a query map sampled from the human reference aligned to its true location to be a true alignment, and all other alignments to be false alignments. We assessed how well the score, score per number of matched chunks, and M-score statistics performed at discriminating true alignments from false alignments. The ROC curves are shown in Figure 2.2 and AUC statistics

in Table 2.1, indicating that the M-Score statistic does better than the alignment score and score per matched chunk statistics. It is worth noting that the score per matched chunk statistics attempts to normalize an alignment score by the number of fragments in each query but this does not perform as well as the M-Score, which indirectly takes into account the "alignability" of a query based on the distribution of its best alignment scores.

2.3.2 *Escherichia coli* K-12

2.3.2.1 Contig Alignment

One practical use of optical mapping is to aid in the scaffolding and finishing of sequence assemblies. By placing contigs on an optical map, one is able to arrange and orient assembled sequence contigs onto a chromosome wide scaffold. To test our software for this purpose we produced a sequence assembly of *E. coli* K-12 using short reads (SRA accession SRX298884) trimmed with PRINSEQ [22] and assembled with SPAdes [23] using default parameters, giving an assembly of 149 contigs 4.58 Mb in length (N50 112 kb).

We ran SOMA, TWIN [10], malignerIX and malignerDP and evaluated the alignment accuracy and runtime performance on a set of 31 contigs (2.81 Mb) that had five or more restriction fragments (including boundary fragments) and a unique nucmer placement. For our evaluation we ignored contigs with 4 or less restriction fragments as these are difficult to uniquely align to the optical map.

We aligned the contigs to an experimentally produced optical consensus map

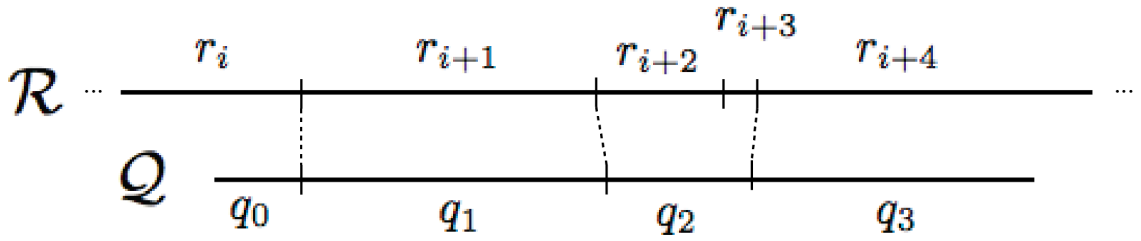

Figure 2.1: **Alignment Notation** The forward alignment of query map $\mathcal{Q} = [q_0, q_1, q_2, q_3]$ against a reference $\mathcal{R}$ illustrated above can be represented as the following ordered listing of matched chunks: $\mathcal{A} = [((\mathcal{Q}, 0, 1), (\mathcal{R}, i, i+1)), ((\mathcal{Q}, 1, 2), (\mathcal{R}, i+1, i+2)), ((\mathcal{Q}, 2, 3), (\mathcal{R}, i+2, i+4)), ((\mathcal{Q}, 3, 4), (\mathcal{R}, i+4, i+5))]$. Since the starting indices of the query chunks are monotonically increasing, the query is aligned in the forward direction.

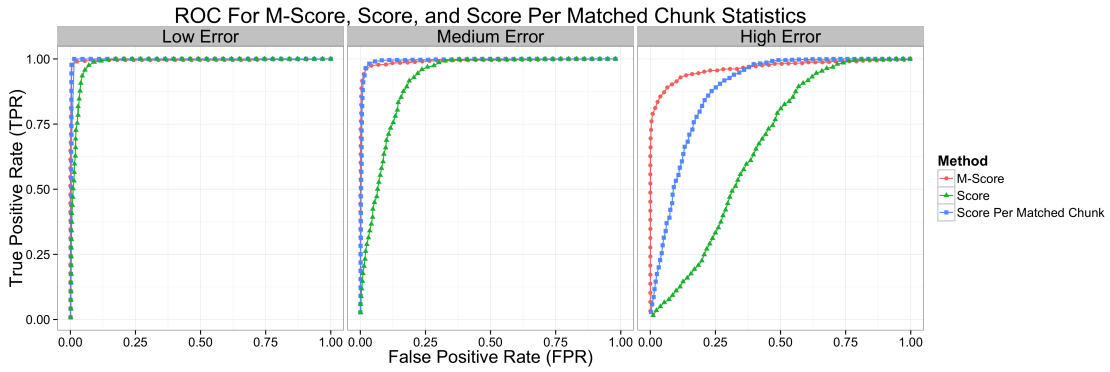

Figure 2.2: **ROC for Alignment Significance** True Positive Rate vs. False Positive Rate for discriminating correct alignments from random alignments on a set of simulated BamHI Rmaps from the human reference.

(425 restriction fragments, 4.17 Mb), assembled from individual molecular maps (Rmaps) produced using enzyme BamHI (with recognition sequence GGATCC). For comparison, we also produced an error free optical map from the *E. coli* K-12 MG1655 reference sequence by performing an *in silico* digest (448 restriction fragments, 4.64 Mb). We note that while the restriction pattern of the experimental optical map is faithful to the reference, it exhibits an overall fragment undersizing bias and is missing small fragments and sites due to desorption (Figure 2.3). Before alignment we smoothed out small restriction fragments less than 1 kb by merging them with neighboring restriction fragments.

We used nucmer [24] to determine the true placement of the contigs on the reference sequence. We took a contig placement to be correct if 90% or more of the contig aligned to a unique location with 95% identity or better.

Alignment to error-free optical map Results for alignment to the error free optical map are shown in Table 2.2. We consider a contig to be placed correctly if its starting location is within 50 kb if the location reported by nucmer. TWIN, malignerIX, and malignerDP all perform similarly, finding correct alignments for all 31 contigs. SOMA does not perform as well, only finding correct alignments for 11 of the 28 contigs. TWIN has a bug that results in reported alignment locations that have several kb in error, as shown in Figure 2.4. This issue prevented us from using a more strict criteria for alignment correctness for this comparison. We have e-mailed the authors of TWIN to inform them of this issue.

Error Setting	Score	Score per Matched Chunk	M-Score
Low	0.982	0.997	0.995
Medium	0.918	0.994	0.991
High	0.664	0.881	0.965

Table 2.1: **AUC for alignment score, alignment score per matched chunk, and M-Score** The AUC for discriminating true alignments from false alignments using the alignment score, alignment score per matched chunk, and M-score statistics for simulated Rmaps from the human reference under low, medium and high error settings. M-Score outperforms the other methods, especially under noisy conditions.

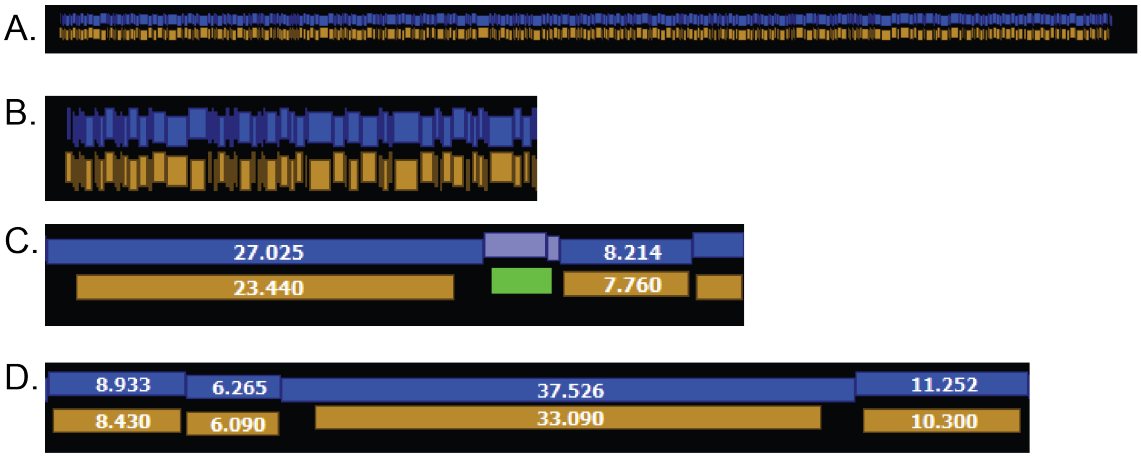

Figure 2.3: **Comparison of *E. coli* K12 experimental optical map with reference** A) Overall, the consensus optical map (gold) assembled from experimental molecular maps shows great concordance with the *in silico* digest of the reference sequence (blue). B) A close view of the left end of the same alignment shown in A. C) An example of a matched chunk containing two fragments in reference (light blue) aligned to one fragment in the optical map (green). This indicates that a small restriction fragment is missing from the optical map. D) Illustration of the overall undersizing bias of the optical map. Fragment sizes are given in kilobases.

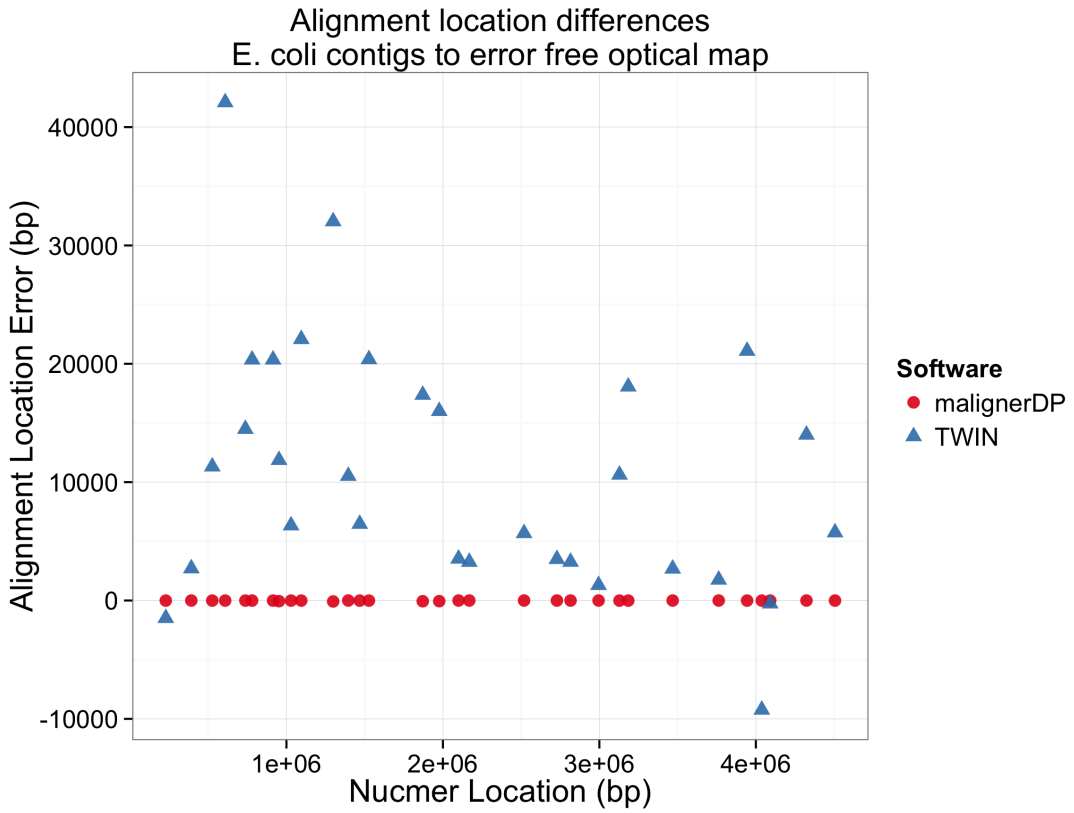

Figure 2.4: **TWIN and malignerDP Contig Location Errors.** Errors in the placement of the contig by malignerDP and TWIN, as compared to the true placement given by nucmer. TWIN and malignerDP place all 31 contigs correctly, but TWIN reports contig alignment locations with several kb of error that cannot be explained by the sizes of the contig boundary restriction fragments.

Software	Total Alignments	Contigs with Alignment	Contigs with Correct Alignment	Contigs with Unique Alignment	Contigs with Unique & Correct Alignment	Runtime
TWIN	37 (2.99 Mb)	31 (2.81 Mb)	31 (2.81 Mb)	28 (2.71 Mb)	28 (2.71 Mb)	0.47s
SOMA	28 (2.66 Mb)	28 (2.66 Mb)	11 (1.48 Mb)	28 (2.66 Mb)	11 (1.48 Mb)	14.22s
malignerIX	36 (2.93 Mb)	31 (2.81 Mb)	31 (2.81 Mb)	29 (2.75Mb)	29 (2.75 Mb)	0.03s
malignerDP	39 (3.01 Mb)	31 (2.81 Mb)	31 (2.81 Mb)	27 (2.68 Mb)	27 (2.68 Mb)	0.40s

Table 2.2: **Alignment results to error free *E. coli* optical map.** Number of contigs (and bp) aligned to the error free *E. coli* optical map. A contig is considered to be placed correctly if its location is within 50kb of the location reported by nucmer.

Alignment to experimental optical map We aligned the set of contigs to an optical consensus map assembled from experimental molecular restriction maps (Rmaps) using Gentig [12]. The smoothed consensus optical map is 4.17 Mb with 425 restriction fragments. The optical map has more noise compared to the the error free map, as there is an undersizing bias in the size of the fragments and some cut sites and small fragments are missing. We see from Table 2.3 that malignerDP is able to find correct alignments for 26 of the 31 contigs, outperforming the index based methods malignerIX (13 contigs) and TWIN (7 contigs) as well as SOMA (0 contigs).

2.3.2.2 Rmap Alignment

We used malignerDP, malignerIX, and TWIN to align a high coverage set (1,159X) of 14,734 Rmaps (average 364 kb, 20 fragments) to the error-free *in silico* digest of the *E. coli* K12 reference sequence.

malignerDP We ran malignerDP on the Rmap set and selected the subset of alignments with at most 40% unmatched sites in the reference, 15% unmatched sites in the Rmap, and an M-Score of 5 or better. 2,831 (19.2%) of the Rmaps had a unique alignment with this criteria, resulting in an overall alignment coverage of 245X. The low mapping rate is due to the fact that we used raw instrument output rather than carefully filtered Rmap datasets, for which alignment rates are much higher (D.C. Schwartz, personal communication). We chose to use raw data to demonstrate the potential of using Maligner as a component in a Q/C pipeline for optical or nanocoding mapping data. No Rmaps had duplicate alignments matching this

criteria. We visualized the alignments in GnomSpace (Figure 2.5), showing that the Rmaps align to the reference with good fidelity. malignerDP completed in 169.86 seconds (11.5ms/Rmap).

malignerIX We ran malignerIX with a maximum allowed relative fragment sizing error of 15% and absolute error of 5 kb per fragment (whichever is greater), at most 40% unmatched sites in the reference, and at most an edit score per interior chunk of 5.0. 553 (3.8%) of the Rmaps had a unique alignment with this criteria, resulting in an overall alignment coverage of 39.7X. No Rmaps had duplicate alignments matching this criteria. malignerIX completed in 5.72 seconds (0.39ms/Rmap).

TWIN We ran TWIN on the Rmap dataset. Note that this comparison is unfair, as TWIN was designed as an aligner for *in silico* restriction maps and as such it does not handle unmatched sites in the query or the reference. Running TWIN with lenient alignment settings (search radius of 5 kb, fval 1000) only produced alignments for 14 Rmaps. We could not determine the alignment locations or coverage due to runtime error's encountered in TWIN's post processing scripts. TWIN completed in 20.35 seconds.

2.3.3 Budgerigar

To show that our methods scale to larger genomes, we aligned both contigs and SwaI Rmaps to an assembled budgerigar parakeet optical map comprising 93 contigs (889 Mb).

Software	Total Alignments	Contigs with Alignment	Contigs with Correct Alignment	Contigs with Unique Alignment	Contigs with Unique & Correct Alignment	Runtime
TWIN	101 (3.44 Mb)	11 (0.49 Mb)	7 (0.26 Mb)	4 (0.23 Mb)	0 (0.00 Mb)	0.76s
SOMA	6 (0.25 Mb)	6 (0.25 Mb)	0 (0.00 Mb)	6 (0.25 Mb)	0 (0.00 Mb)	14.45s
malignerIX	81 (3.23 Mb)	15 (0.99 Mb)	13 (0.87 Mb)	7 (0.65 Mb)	5 (0.53 Mb)	0.15s
malignerDP	208 (8.35 Mb)	28 (2.37 Mb)	26 (2.24 Mb)	13 (1.60 Mb)	13 (1.60 Mb)	0.31s

Table 2.3: **Alignment results to experimental *E. coli* optical map.** Number of contigs (and bp) aligned to the experimental *E. coli* consensus optical map. A contig is considered to be placed correctly if its location given as percentage of optical map length is within 5% of the location reported by nucmer. Among these software, the only alignment method which is able to handle the experimental error characteristics of real data is malignerDP. Index based methods, being less flexible, are not able to find the same number of correct placements.

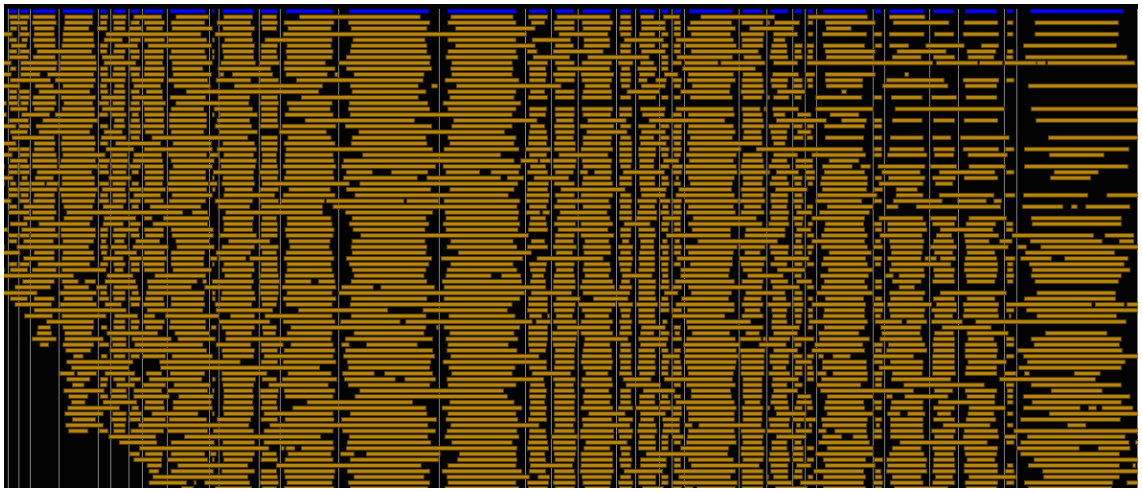

Figure 2.5: **Alignment of *E. coli* K12 Rmaps to reference sequence** A pileup of Rmap alignments (gold) to the *in silico* digest of the *E. coli* K12 MG1655 reference (blue) produced using the malignerDP software.

2.3.3.1 Contig Alignment

We aligned the budgerigar v6.3 assembly contigs to the budgerigar optical map, both published in [16]. Before aligning, we filtered the total assembly (70,863 contigs, 1.09Gb, 55.6kb N50) down to those contigs with 5 or more restriction fragments (6,008 contigs, 406 Mb). We aligned these contigs using TWIN, malignerDP, and malignerIX to evaluate the number of alignments and run time performance. We chose not to run SOMA since SOMA does not scale to genomes of this size, as documented in [10]. The alignment results are summarized in Table 2.4. We find that malignerIX runs in comparable time to TWIN, but aligns more contigs and places more contigs uniquely, as it is able to handle more unmatched sites in the reference. MalignerDP runs much slower than the index based methods, but finds more unique alignments.

2.3.3.2 Rmap Alignment

We aligned a set of 671,896 Rmaps (352 kb avg, 18.3 fragments avg, 236.6 Gb total) to the Parrot optical map using malignerDP, filtering our alignment set to those with a query unmatched site rate $\leq 15\%$, a reference unmatched site rate $\leq 40\%$, and an M-Score ≤ 20. 69,537 (10.3%, 30.9 Gb) of the Rmaps had at least one alignment fitting this criteria, 69,130 (30.5 Gb) of which were aligned uniquely. Total alignment took 334h 33m of CPU time (1.8 sec/Rmap).

2.4 Discussion and Conclusions

We have presented the malignerDP and malignerIX software for restriction map alignment and evaluated their performance on experimental datasets for *E. coli* and budgerigar parakeet. We have shown that on *E. coli*, malignerDP finds more correct alignments than other available methods. We have also demonstrated that malignerDP can align a high coverage Rmap set for a large genome within a couple hours on a moderately sized cluster.

We have also introduced the M-Score, which provides a method for normalizing alignment scores found through dynamic programming by adjusting the scores for each query based on the distribution of the best scoring but random alignments for that query. The normalization allows one to apply a score threshold across queries to for accepting or rejecting alignments, and thereby avoid a permutation test for determining alignment significance.

Finally, we have shown that while the index based methods malignerIX and TWIN run significantly faster than malignerDP, these methods are less sensitive than full dynamic programming, finding less alignments (for Rmap alignment) and less correct alignments (for contig alignment) against experimental datasets.

2.5 Methods

The maligner software has two modes of alignment: (i) a dynamic programming implementation malignerDP that allows for unmatched sites in both the query

and reference and a faster mode malignerIX which builds an index on the reference which is queried through binary search but does not allow unmatched sites in the query.

2.5.1 Dynamic Programming Based Alignment

We extend the work of Nagarajan et al. [8] to build a map aligner which can scale to large eukaryotic genomes. Given a matched chunk with query length q bp, reference length r bp, m interior unmatched query sites, and n interior unmatched reference sites, we use scoring function which represents an edit distance between a query chunk and a reference chunk:

$$\text{Score}(q, r, m, n) = S(q, r) + C_q m + C_r n \tag{2.1}$$

$$S(q, r) = \left(\frac{q - r}{\sigma(r)} \right)^2 \text{ if not boundary chunk else } 0 \tag{2.2}$$

$$\sigma(r) = \max(\alpha r, \sigma_{min}) \tag{2.3}$$

where C_q is the fixed cost for an unmatched query site, C_r is the fixed cost for an unmatched site in the reference, and $S(q, r)$ is the cost of the sizing difference between the query chunk and the reference chunk. Note that we make an adjustment for small reference fragments, which can exhibit higher relative error rates [7], by putting a lower bound σ_{min} on the sizing error scaling parameter.

Given a query map $\mathcal{Q}$ and a reference map $\mathcal{R}$, we seek an alignment which minimizes the sum of the cost of the matched chunks. Since each chunk is scored independently, this permits a dynamic programming solution given by Algorithm

1. We bound the number of consecutive unmatched sites in the query by δ_q and reference by δ_r, giving an algorithm that is $\mathcal{O}(\delta_q \delta_r mn)$

Algorithm 1 maligner dynamic programming

procedure MALIGNDP($\mathcal{Q}, \mathcal{R}, \delta_q, \delta_r$)
    ▷ Initialize score matrix.
    for $j \leftarrow 0, n+1$ **do**
        $SM(0, j) \leftarrow 0$
    end for
    ▷ Fill the score matrix.
    for $i \leftarrow 1, m+1$ **do**
        for $j \leftarrow 1, n+1$ **do**
            $SM(i, j) \leftarrow \infty$
            $BackPointer(i, j) \leftarrow$ nullptr
            for $k \leftarrow \max(i - \delta_q - 1, 0), i-1$ **do**
                for $l \leftarrow \max(j - \delta_r - 1, 0), j-1$ **do**
                    if $SM(k, l) < \infty$ **then**
                        $q \leftarrow \sum_{i'=k}^{i-1} q_{i'}$ ▷ query chunk size
                        $r \leftarrow \sum_{i'=l}^{j-1} r_{i'}$ ▷ ref chunk size
                        ▷ Compute score of this alignment extension
                        $Score \leftarrow SM(k, l) + S(q, r) + C_q[i - k - 1] + C_r[l - j - 1]$
                        if $Score < SM(i, j)$ **then**
                            $SM(i, j) \leftarrow Score$
                            $BackPointer(i, j) \leftarrow (k, l)$
                        end if
                    end if
                end for
            end for
        end for
    end for
    return $(SM, BackPointer)$
end procedure

2.5.1.1 Alignment Significance

The dynamic programming method used by malignerDP will find one alignment for each position in the reference. However, it is not clear whether the best scoring alignment is significantly better than random. One method for determin-

ing whether an alignment is significant is to perform a permutation test whereby a given query is aligned to a population of permuted references. Each alignment is assigned a p-value under the null hypothesis H_0 that the query is not related to the reference by determining the fraction of permuted references that have a better scoring alignment for that query. We do not consider the permutation test for this problem because it is computationally expensive and infeasible for alignment to large genomes.

Since our scoring function is a measure of edit distance, we cannot simply choose a single score cutoff for accepting or rejecting alignments that will apply to all queries, as the score for a quality alignment varies with the number of fragments and the size of the fragments in each query [11]. Instead, we propose computing a cutoff score for each query map based upon the distribution of alignment scores of the best non-overlapping alignments. If a query has a single acceptable alignment to the reference, we expect that the best scoring alignment will be correct and the rest of the alignments to be random. Using this intuition, we formulate the M-Score as follows:

$$m_{\mathcal{A}} = \underset{A \in \mathcal{A}}{\text{median}} \{\text{Score}(A)\} \tag{2.4}$$

$$\text{MAD}_{\mathcal{A}} = \underset{A \in \mathcal{A}}{\text{median}} \{|\text{Score}(A) - m_{\mathcal{A}}|\} \tag{2.5}$$

$$\text{M-Score}_{\mathcal{A}}(A) = \frac{\text{Score}(A) - m_{\mathcal{A}}}{\text{MAD}_{\mathcal{A}}} \tag{2.6}$$

where we take $\mathcal{A}$ to be the top 100 alignments for the given query against the reference. The M-Score normalizes all alignment scores across queries by shifting

each query's scores to the median and scaling by the median absolute deviation (MAD) of the top ranking alignments for that query. This allows us to select an M-Score cutoff for selecting significant alignments that can be applied across queries. We note that using dynamic programming as the method of alignment (as compared to index based methods) gives us the distribution of alignment scores at no additional cost since the best alignment score at each position of the reference is computed when the score matrix is populated in Algorithm 1. The M-score method essentially uses the top 100 alignments in place of the null distribution of alignment scores one could obtain through a computationally expensive permutation test.

We typically select an M-Score cutoff which maximizes the fraction of queries with a unique alignment below the cutoff. The M-Score cutoff can vary dataset from dataset based on enzyme digestion rate, uniformity of molecular stretch, and the number of restriction fragments per Rmap. Higher quality datasets will have higher quality alignments and therefore allow for a the selection of stricter M-Score cutoff. Note that more negative M-Scores correspond to better quality alignments (since lower alignment scores are better).

2.5.2 Index Based Alignment

If we do not allow for unmatched sites in the query map, we can leverage an indexed based method of alignment that avoids $\mathcal{O}(\delta_q \delta_r mn)$ dynamic programming and instead uses an index built on the reference map. The reference can be indexed by extracting all possible chunks with at most δ_r interior missed restriction sites.

Specifically, we consider the set of all chunks $\mathcal{N} = \{(\mathcal{R}, s, e) \mid e - s \leq \delta_r\}$ from the reference $\mathcal{R}$.

The adjacencies of the reference chunks $\mathcal{N}$ can be represented as a directed acyclic graph (DAG), where we include an edge $r_a \rightarrow r_b$ between chunks $r_a = (\mathcal{R}, s_a, e_a)$ and $r_b = (\mathcal{R}, s_b, e_b)$ if $s_b = e_a$. We build an index on the graph by storing all nodes $\mathcal{N}$ (i.e., chunks) in an array sorted by chunk length. For a given query chunk c_q and a lower bound function $L(C)$ and upper bound function $U(C)$ we can find all reference chunks that are compatible with c_q in length $\{C \in \mathcal{N} \mid L(c_q) \leq l(C) \leq U(c_q)\}$ from the index in $\mathcal{O}(n \log n)$ time by binary search. Our implementation uses lower and upper bound functions $L(c) = c - \max(\alpha c, \delta)$ and $U(c) = c + \max(\alpha c, \delta)$ where $0 < \alpha < 1$ specifies the relative error and $\delta > 0$ specifies the minimum absolute error tolerance.

We can leverage the reference chunk index and the DAG to find alignments for a query map $\mathcal{Q}$ with k fragments if we restrict our search space to the set of alignments with no unmatched sites in Q. Specifically, we search for one more paths of adjacent reference chunks $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_k$ where: (i) each chunk c_i is compatible with the ith fragment in $q_i = (\mathcal{Q}, i, i+1)$ for $i \in [1, k]$ and (ii) the rate of unmatched sites in the reference is less than a user selected threshold.

We find such alignments by first seeding on the largest interior restriction fragment in the query. Since genomic restriction fragment lengths are approximately exponentially distributed, the largest fragment in the query will have the fewest number of compatible fragments. For each reference chunk compatible with the seed fragments we perform bounded DFS to find all possible compatible right extensions

and left extensions with respect to the reference which compatibly align all remaining fragments in the query. With each step in the DFS, we only consider taking an edge to a reference chunk if that reference chunk is compatible with the next fragment in the query. Of all extensions found (if any), we take the left and right extension which has the smallest number of unmatched sites in the reference. We concatenate the best left extension in the reference with the seed chunk and the best right extension to produce the best forward alignment for each seed hit. We align in the reverse direction by considering aligning to the DAG corresponding to the reverse of the reference, reusing the seed hits found for forward alignment. In a final post-processing step, we apply the same score function given by Equation 2.1 to rank all alignments found and output a set of non-overlapping alignments selected in order of alignment score.

While this index based method of alignment is not as flexible as full dynamic programming because it does not allow for unmatched sites in the query, it runs orders of magnitude faster.

2.5.3 Implementation

The Maligner software is written in C++ and is available at `https://github.com/LeeMendelowitz/maligner` under the GNU General Public License.

2.6 Acknowledgments

This work has been funded by NSF IIS-1117247 to MP and NIH R01-HG-000225 to DC Schwartz, subcontract to MP. LM would like to acknowledge Martin Muggli for his assistance in running the TWIN software and Shiguo Zhou for providing the Rmaps and optical map for *E. coli* K-12.

Software	Contigs Aligned	Contigs Aligned Uniquely	Runtime
TWIN	3,889 (267.0 Mb)	1,340 (130.8 Mb)	51.01 s
malignerDP	5,093 (427.7 Mb)	2,635 (299.7 Mb)	46m 16.0s
malignerIX	5,142 (422.8 Mb)	2,148 (249.7 Mb)	51.53 s

Table 2.4: **Contig alignment to budgerigar optical map** Number of contigs (and bp) aligned to the parrot optical map.

Chapter 3: Visualization of Pairwise Restriction Map Alignments with MalignViz

3.1 Introduction

In aligning a query restriction map to a reference restriction map, alignments are selected by minimizing some objective scoring criterion. In the case of malignerDP, the scoring function given by Equation 2.1 comprises terms which penalize for unmatched restriction sites in the query and the reference as well as sizing error between aligned query and reference chunks. However, there are some practical challenges when using such an objective scoring function for selecting the best alignments. For example, there is a trade-off between the number of unaligned sites and the sizing error in the selected alignments. One run of malignerDP with one set of alignment parameters may favor reducing the number of unaligned sites with greater tolerance for sizing error, while another alignment run with different parameters may accept more unaligned sites while reducing sizing error. These trade-offs are controlled by the constants C_q, C_r, and α in Equation 2.1. The relative ranking of alignments can change as these parameters are adjusted. This raises a practical issue: how can the end user be sure that the scoring functions parameters are set

to the appropriate values such that the best scoring alignments are of reasonable quality?

A second issue that arises is properly selecting a set of unique high quality set of alignments for downstream processing. Given a query map, malignerDP will output a ranked list of non-overlapping alignments. How can one be confident that the best ranked alignment is correct and the secondary alignments are no better than random? The M-Score tries to address this by assigning a normalized score to the best ranking alignments for each query. However, one must still choose an M-Score cutoff for selecting significant alignments.

To better understand how multiple alignment candidates for a single query against a reference differ in quality, it is helpful to make use of interactive visualization tools. Visualization is an important tool as it aids in the interpretability of alignment scores and validates that the alignment tools are working correctly for the dataset at hand. Unlike the sequence alignment problem, for which pairwise sequence alignments can be visualized using plain text outputs (such as those provided by BLAST [25] and MUMmer's [24] show-aligns), pairwise restriction map alignments cannot be easily visualized and compared using a plain text format.

Few tools exist for visualizing optical map alignments. The GnomeSpace Viewer [4] developed by David Schwartz's Lab is useful for viewing a pileup of alignments against a reference. While GnomeSpace provides a reference centric visualization, is not useful for visualizing multiple pairwise alignment candidates for a single query since it does not offer features for selecting, sorting, or filtering alignments for a single query. The MapSolver^{TM} software by OpGen is not freely

available and is only available to customers who pay for OpGen's services (Clayton Collier, personal communication, June, 10 2015). From the limited available online documentation [26], MapSolver^{TM} does not appear to be useful for visualizing multiple pairwise alignment candidates. gEVAL [27] is an online genome browser which includes a simple visualization of optical mapping data against a reference, but, like GnomeSpace, it is not useful for considering different candidate alignments for a single query.

MalignViz has been created to fill this need. MalignViz is a portable web application for visualizing pairwise restriction map alignments. MalignViz has proven to be an invaluable tool while developing MalignerDP, suggesting new features such as query rescaling and realignment for improving alignment quality.

3.2 Tour of MalignViz

In this section we present a brief tour of the MalignViz application from the perspective of an end user. MalignViz organizes its datasets into separate experiments. Each experiment consists of a set of query maps, a set of reference maps to which the query maps were aligned, and a set of alignments generated by MalignerDP or MalignerIX. From the experiment dashboard page (Figure 3.1) the user is presented with a list of all available experiments, which they can edit, delete, and open. From this page new experiments can be also be created.

New experiments are created through the create experiment modal (Figure 3.2). Each experiment is given a name, and optionally a description. When creating an

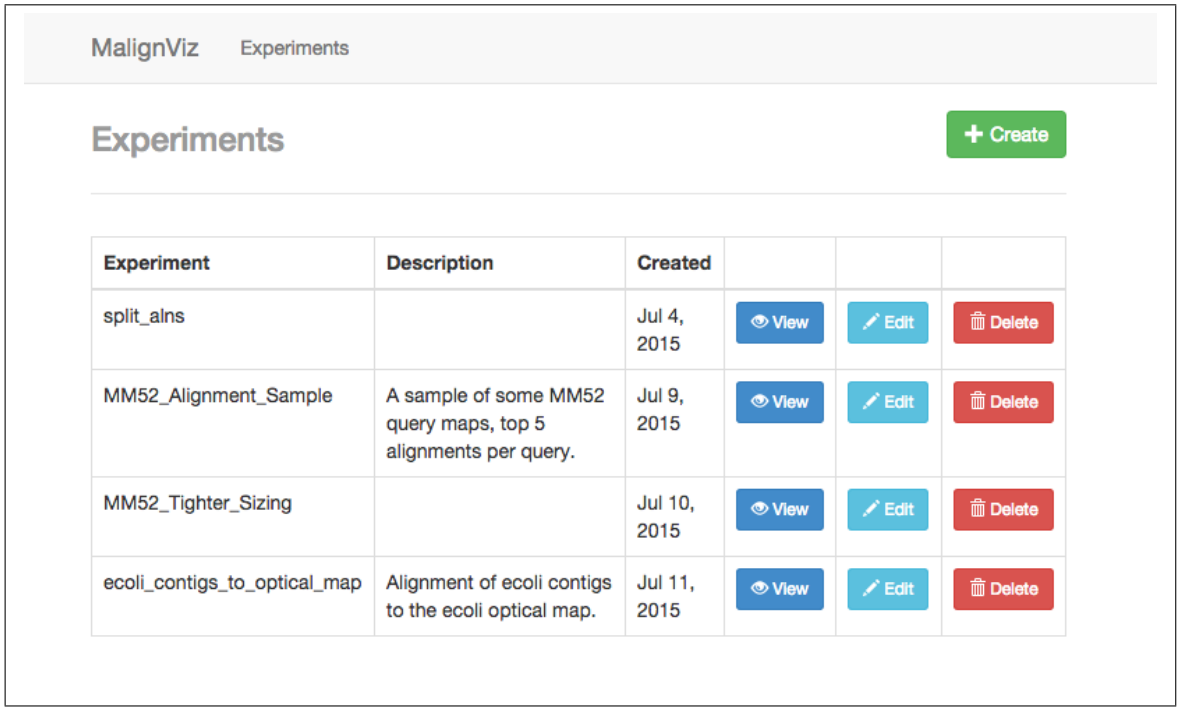

Figure 3.1: **MalignViz Experiment Dashboard** The experiment dashboard where users can create, edit, delete, and open experiments.

experiment, the user must provide a JSON file with the query maps, reference maps, and alignments. Maligner map files and alignment files can be converted from their tab delimited format to the MalignViz JSON format using the scripts distributed with the Maligner software.

The landing page for an experiment, shown in Figure 3.3, displays the experiment description and a summary indicating how many query maps, reference, and alignments are associated with the experiment. A sortable, paged table of query maps is displayed, along with a count of the number of alignments per query. Each query links to its own page.

The landing page for a query map includes information about the query map such as the number of restriction fragments and its total length and also includes

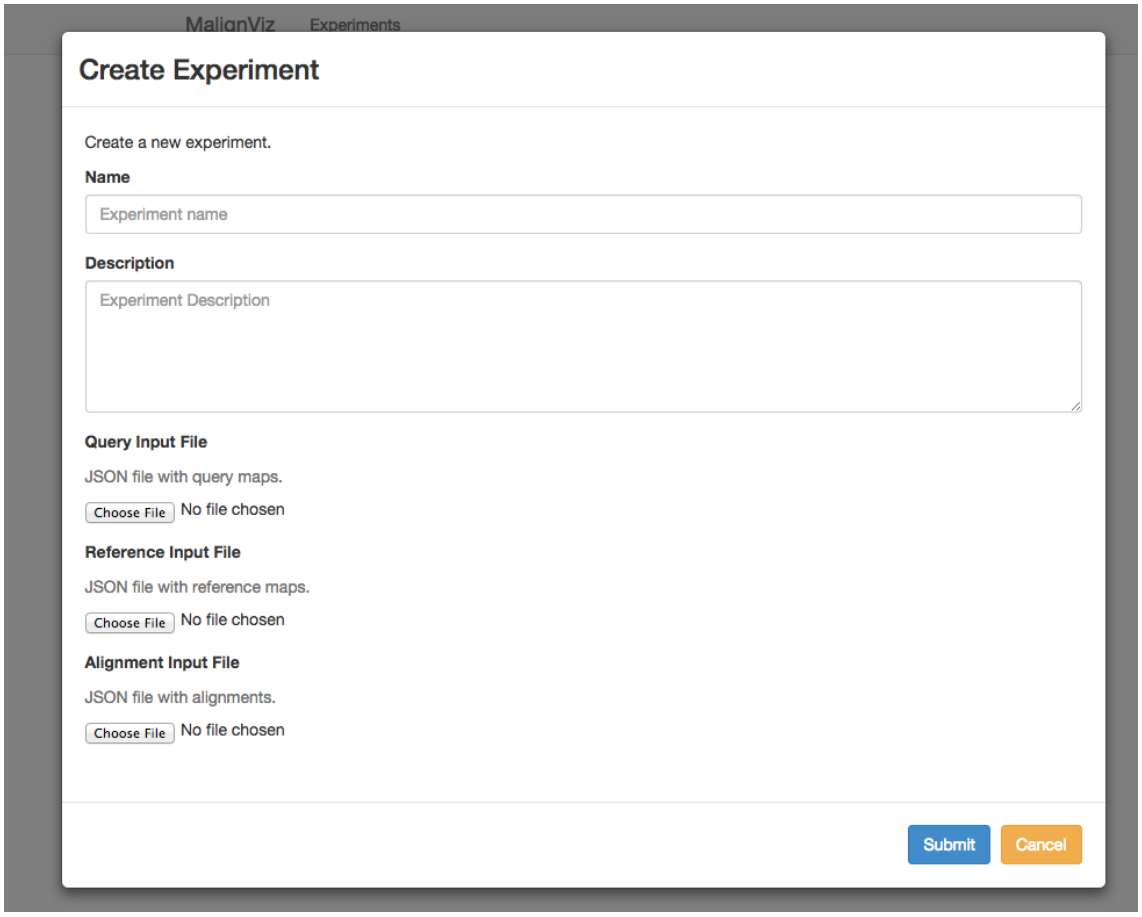

Figure 3.2: **MalignViz Create Experiment Modal** Experiments can be created through the MalignViz web interface through the create experiment modal. Users can provide an experiment name, description, and JSON files with the query maps, reference maps, and alignments.

a visualization of the map's restriction pattern (Figure 3.4). A paged, sortable table with all candidate alignments for the query map is displayed below along with alignment summary statistics, including the number of matched sites, the alignment score, the M-Score, the different components of the scoring object function, and the rate of unmatched sites in the query and reference. Each alignment entry in the table includes a button for displaying that alignment.

Below the table of alignments is an interactive visualization displaying the

pairwise alignment candidates (Figure 3.5). A tooltip is shown summarizing the components of the scoring function for each matched chunk of the alignment. The visualization shows restriction fragments scaled by their length, with the reference along the top and the aligned query map below. Errors in the alignment are clearly shown, with unmatched restriction sites colored in red and sizing differences between query chunks and reference chunks represented by white space between. The matched chunks are displayed in alternating colors of blue and black to clearly show which group of query fragments are aligned to which group of reference fragments.

As a user navigates the MalignViz application, the state of the application is maintained by deep linking. This means that URL's pointing to specific experiments, queries, or alignments can be bookmarked and revisited at a later time.

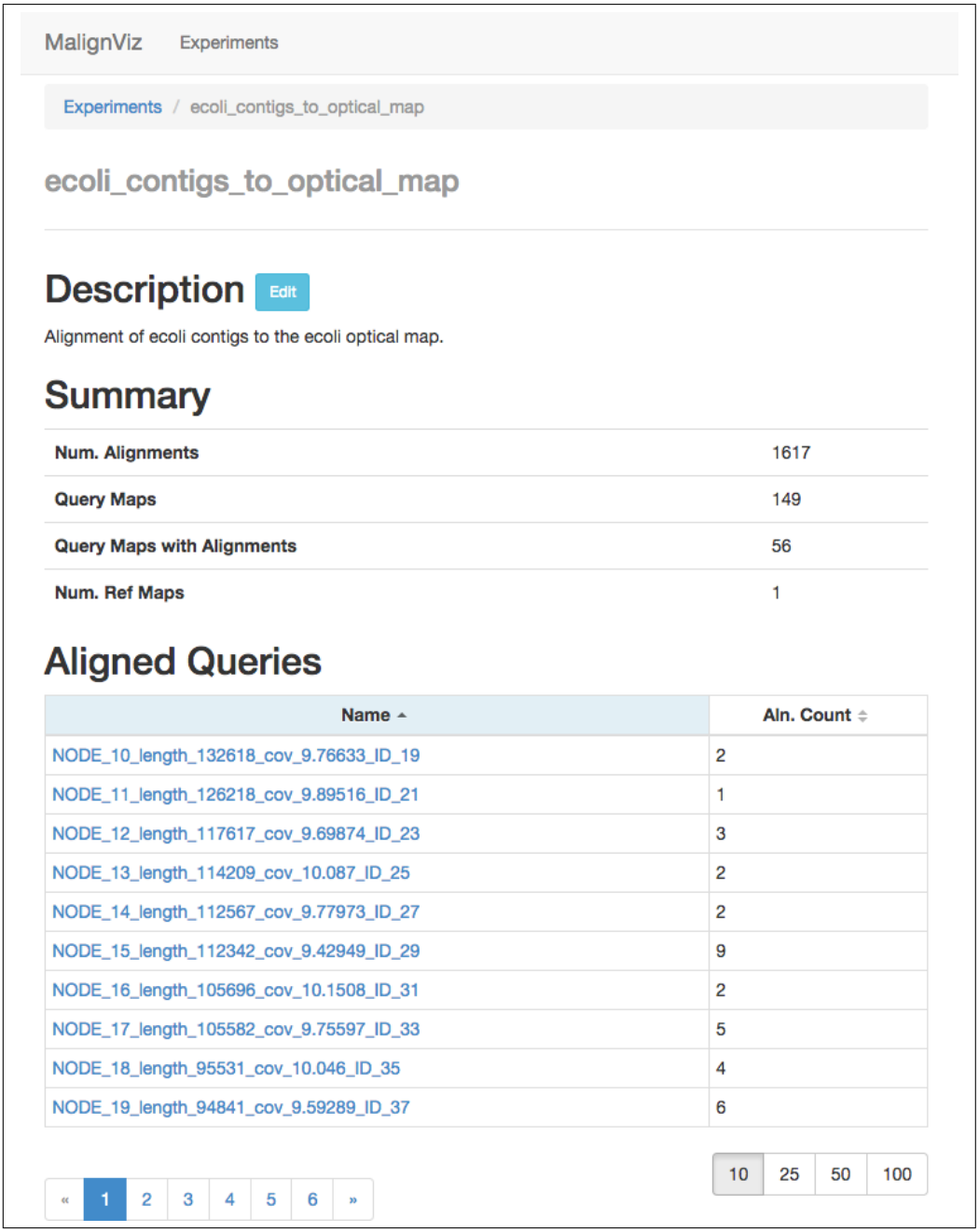

Figure 3.3: **MalignViz Experiment Landing Page** The MalignViz experiment landing page, which shows the experiment name, description, a summary of the experiment with counts for the number of alignments, number of query maps, number of query maps with alignments, and number of reference maps. Lastly, a table shows a listing of query maps with the number of alignments.

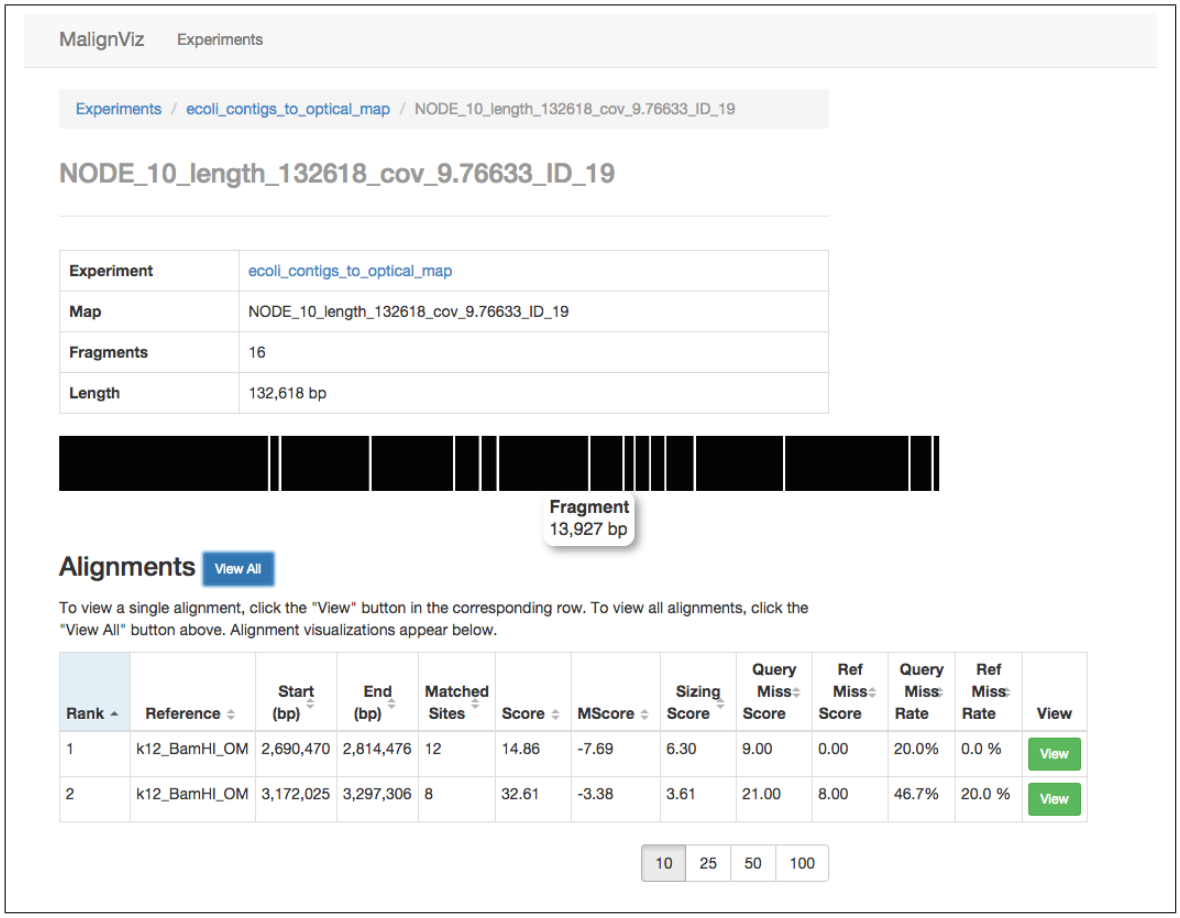

Figure 3.4: **MalignViz Query Landing Page**
The MalignViz query landing page, which shows the query map's name, the number of fragments, the total map length, and an interactive visualization of the query map. A sortable table listing all alignments along with their summary statistics is also displayed, along with controls for displaying the pairwise alignments.

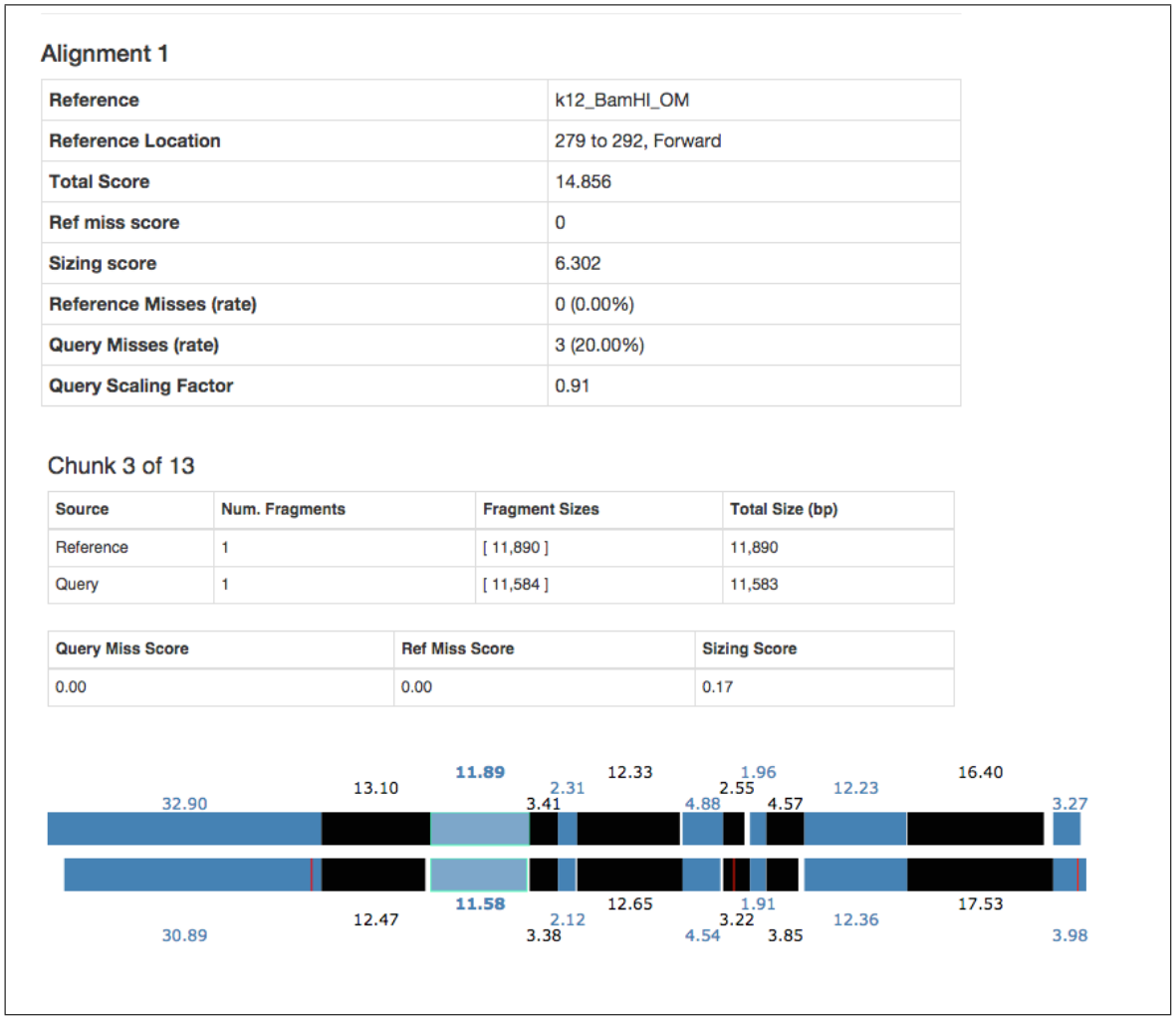

Figure 3.5: **MalignViz Pairwise Alignment** This is a screenshot of the MalignViz alignment visualization which appears on the landing page for a query. MalignViz summarizes each alignment with a table that shows the reference map, reference location, alignment score, the components of the alignment score, and other alignment summary statistics. Below the table is a tooltip which shows details for each matched chunk, including the fragment sizes and score components. Finally, at bottom is the interactive pairwise alignment visualization, with the reference along the top and the query below. Unmatched sites are shown in red, and matched chunks appear in alternating colors of blue and black.

3.3 Results

MalignViz has proven to be a useful tool for better understanding the experimental errors inherent in optical mapping. MalignViz has helped inform decisions in how to appropriately set alignment scoring function parameters and incorporate new features into the MalignerDP software, such as query rescaling and realignment.

One experimental error inherent in optical mapping is the underrepresentation of small restriction fragments due to desorption, where small fragments may float away after digestion. As a result, when aligning and placing contigs from a sequence assembly onto an optical map, there can be restriction sites in the *in silico* digest of the sequence contig that are unmatched in pairwise alignments. This effect is illustrated in Figure 3.6, which shows two unmatched sites in the sequence contig corresponding to small restriction fragments which are not represented in the consensus optical map. This visualization suggests that instead of using a fixed costs for unmatched sites, the scoring function could be modified so that unmatched sites are penalized as a function of their distance from the closest matched site.

Optical maps may also exhibit an overall oversizing or undersizing bias, where restriction fragments lengths are either overestimated or underestimated. In the case of the *E. coli* K12 consensus optical map, there is a undersizing bias. To account for this bias, MalignerDP implements a query rescaling feature, where query chunks of each candidate alignment are rescaled post-alignment so that the total length of the interior query fragments matches that of the reference. This rescaling reduces the sizing error observed for high quality alignments and helps control for the effect

of sizing bias. An alignment before and after query rescaling is shown in Figure 3.7 where a 205kb contig is aligned to a 186 kb portion of the optical map.

While query rescaling helps to control for the sizing bias, the dynamic programming may still select a sub-optimal alignment pattern with extra unmatched sites. In selecting the alignment pattern, MalignerDP first minimizes the objective function which in this case produces an alignment pattern with unmatched sites. However, after rescaling the query fragments in this alignment, it is clear that this selected alignment pattern is suboptimal, as the unmatched sites could be matched to further minimize the scoring objective function. This suggests an improvement to MalignerDP where pairwise alignments are updated after query rescaling to optimize the selected alignment pattern and further minimize the objective function.

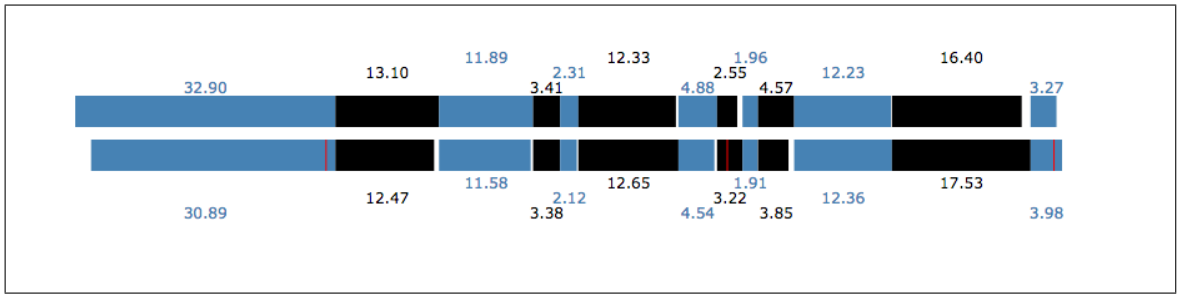

Figure 3.6:
Alignment of *E. coli* contig to optical map with missing small fragments
Small fragments are underreprepresented in optical maps due to desorption of small restriction fragments. This alignment of a *in silico* digest of an *E. coli* K12 contig from a sequence contig (at bottom) to the *E. coli* K12 optical map (at top) demonstrates this effect. The two leftmost unmatched sites in the sequence contig bound two small restriction fragments 1.2 kb and 1.3kb in length. These small fragments are not represented in the optical map.

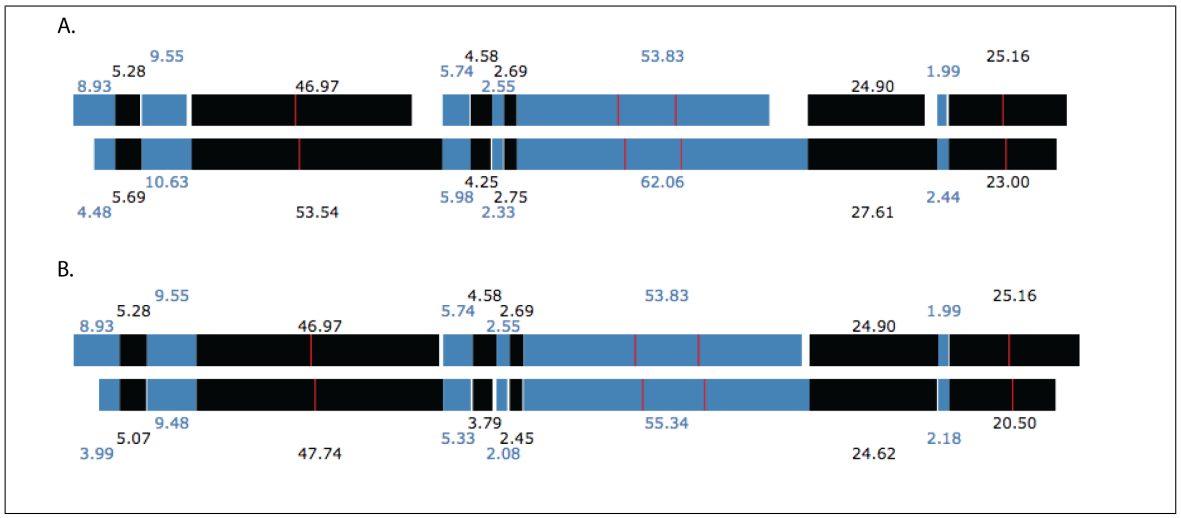

Figure 3.7:
Alignment of *E. coli* contig to optical map with sub-optimal alignment pattern
There can be an overall sizing bias in optical mapping, which results undersized or oversized restriction fragments in both Rmaps and assembled consensus optical maps. A) The alignment above shows a sequence contig 205 kb in length (bottom track) aligned to a 186 kb portion of the optical map (top track). In this pairwise alignment, the optical map is undersized compared to the sequence contig, as can be seen from the white space. B) The same alignment, after rescaling the query. While MalignerDP places the contig correctly, the alignment pattern selected by dynamic program is not optimal, as there are more unmatched sites than necessary. The unmatched sites in red in both the query and reference could be matched to produce a better overall alignment.

3.4 MalignViz Architecture

3.4.1 Back end architecture

MalignViz has been implemented as a portable web application, using freely available open source tools. The MalignViz back end stack includes an API for retrieving experiments and alignments, a database layer, and an Nginx server for serving static resources and routing requests to the API. The API has been implemented using Flask (a Python web application framework) and is served with Gunicorn (a WSGI HTTP server). The database layer uses MongoDB [28], a NoSQL document database, for storing documents representing experiments, query maps, reference maps, and alignments.

MalignViz is typical of most web applications in that it comprises several different components, each with its own dependencies and configuration files. Since the underlying software stack of Python/MongoDB/Nginx is portable, MalignViz could be configured to run natively on Linux, Mac OS X, and Windows. However, with so many components, it is cumbersome to install and configure these dependencies by hand. Due to platform specific issues that may arise, it is hard to guarantee a smooth installation for all users.

Instead, the MalignViz application has been configured to be built and deployed using Docker, a platform for running applications in isolated containers. This means that MalignViz can run in any environment that supports Docker. This includes Linux, which runs Docker natively, and Windows and Mac OS X through

the boot2docker VirtualBox virtual machine. Docker is a lightweight virtualization system which uses Linux kernel features (namespaces and control groups) for safely isolating groups of processes while allowing them to share host system resources [29]. Docker containers are lightweight thanks in part to a union filesystem which allows multiple containers to share the same base filesystem. Docker containers differ from traditional virtual machines such as VirtualBox in that the virtual machines are isolated from each other and share nothing, running separate isolated operating systems on virtualized hardware components. Containers, meanwhile, share the same running instance of the Linux kernel and can share the read only parts of the filesystem that they have in common with other containers, thanks to the union filesystem.

Docker containerization makes it simple to install and deploy the MalignViz application. MalignViz comes with a Dockerfile which installs and configures MongoDB, Nginx, Python, Flask, Gunicorn, and other required python packages on top of the base Ubuntu 14.04 Docker image. Installing and configuring MalignViz is as simple as issuing a `docker build` command. MalignViz can be run using `docker start`, which will start each application component in the appropriate order in an isolated container and expose the Nginx HTTP server so that is accessible by a web browser running on the host machine. The application can easily be shut down with `docker stop`.

3.4.2 Front end architecture

The front end of the MalignViz is a single page website implemented using the AngularJS framework [30], with visualizations implemented using the D3.js [31] library. The web page uses the Bootstrap framework [32] for responsive layout and styling. The application was developed using the Yeoman workflow [33], using `yo` with the angular generator for scaffolding the web application and the build configuration, `grunt` [34] for building the website, and `bower` [35] for installing JavaScript library dependencies (including jQuery, AngularJS, and D3.js).

AngularJS is popular JavaScript application framework which promotes modular web development by enabling the construction of a complicated application using re-usable components. AngularJS follows the Model View Controller (MVC) application structure for logically separating code for storing data, modifying data, and presenting data. In AngularJS, the controller is a JavaScript function responsible for initializing, modifying, and validating data values. Controllers make use of services, which are singletons that provide common functionality across the application. Controllers assign data to JavaScript objects (i.e. the model) which are published on the view via Angular's `$scope` service. In AngularJS, the view is specified in HTML via the document object model (DOM). Interactions with elements in the view such as buttons, links, inputs, and form elements can change model values via functions published by the controller.

3.4.2.1 Services

Services are singleton objects that provide functionality that can be used across an Angular application. Services are instantiated and provided to controllers via dependency injection by simply specifying the service as an input parameter to the controller function.

MalignViz implements an API service and a data storage service. The API service is used throughout the front end application for making requests from the Flask API. These requests include retrieving a list of experiments; creating a new experiment; and requesting query maps, reference maps, and alignments associated with an experiment. The data storage service is used for caching map data and alignment data in the browser's session storage so that the API does not have be queried with each page change, providing for a faster and smoother user experience.

3.4.2.2 Visualization Directives

Directives are the most powerful feature unique to the AngularJS framework. Directives are JavaScript functions that instruct AngularJS how to wire components together and make a web page come alive. For example, directives can be used to bind elements in a view to data provided in a model, to attach a controller to an HTML element, to attach a click handler to an element, or to arbitrarily add, remove, or manipulate elements in the document in response to changes to a model. Directives are defined via a JavaScript function which implements the activities of the directive, and directives are included in the HTML view via non-standard

HTML elements or attributes or via class names. This makes it easy for a developer to understand how an AngularJS application works: by simply looking at the HTML template for directives, one can see where controllers and callbacks are attached to the page and how data is bound to elements.

MalignViz makes use of directives for implementing its visualizations. MalignViz provides two simple visualizations: one for displaying a restriction map, and one for displaying a pairwise alignment between a query and reference restriction map.

The restriction map visualization displays a single restriction map as a fixed width SVG graphic. The restriction fragments are represented as an array of integers specifying the fragment lengths in base pair units. D3 is used to bind this data to an ordered display of SVG rectangles, drawn with width corresponding to the restriction fragment length. Restriction sites are shown as small gaps between fragments. D3 linear scales are used to convert base pair units to pixels. Since the visualization is fixed width at 800 pixels, D3 scales take care of automatically stretching the restriction map visualization to occupy this full width. This means that the base pair to pixel conversion factor used to display the visualization depends on the restriction map's total length. For this reason a tooltip is provided so that when a user hovers over a restriction fragment, the fragment length is displayed. The restriction map visualization is shown as part of the query landing page in Figure 3.4.

The pairwise alignment directive displays an alignment as an ordered listing of matched chunks, as shown in Figures 3.6 and 3.7. The alignment visualization always displays reference fragments in the forward orientation (i.e., in the same order as given by the reference restriction map, from left to right). If the query is

aligned to the forward strand of the reference, the query fragments appear in the same order as given by the query restriction map. Otherwise, if the query is aligned to the reverse strand the query fragments are displayed in reverse order.

A matched chunk comprises one or more consecutive reference restriction fragments aligned to one or more consecutive query restriction fragments. Each matched chunk is displayed with the reference restriction fragments above with the matched query fragments below. The matched chunk is displayed with both the query chunk and reference chunk left aligned, and the matched chunk is given enough space to accommodate both the query chunk and reference chunk, which causes white space to appear where there are sizing differences. Interior unmatched restriction sites appear as red lines. The fragments of the matched chunks are displayed in alternating colors to aid the eye in identifying which query fragments are aligned to which reference fragments. Below each matched chunk the sizes are displayed in kilobase pair units.

For each alignment, a table is displayed with alignment summary statistics, including the M-Score, query scaling factor, alignment score, query unmatched site rate, and reference unmatched site rate. A tooltip is used to display information about each matched chunk, including the components of the Maligner scoring function (the query unmatched site cost, reference unmatched site cost, and sizing error cost). The user can hover over each matched chunk with the mouse to update the information displayed in the tooltip.

Data is bound to the pairwise alignment directive by the query page's controller, and the D3 library is used to draw, size, and color the lines, rectangles, and

text corresponding the matched chunk visualization.

3.5 Conclusions

MalignViz has proven to be a useful visualization tool for viewing pairwise alignment candidates generated by Maligner. These visualizations are helpful for understanding how experimental errors in optical mapping such as the underrepresentation of small fragments due to desorption or sizing bias can impact alignment. Such visualizations have aided the conception of new alignment features, such as query rescaling and query realignment. These features have been incorporated into the Maligner software.

3.5.1 Availability

MalignViz is available on GitHub under a GNU GPL license at `https://github.com/LeeMendelowitz/malign_viz`.

Chapter 4: Structural Variation Detection using Restriction Map Alignment

4.1 Introduction

Genetic variation comes in many different forms, from single nucleotide variants (SNV's) to larger events such as deletions, insertions, inversions, and translocations [36–38]. Structural variants, which are defined to be a genetic variation that affects greater than 50 basepairs of DNA [39], have been implicated in many human diseases [40] including Down's syndrome [41], autism [42], Crohn's disease [43], and schizophrenia [44], among many others. Genomic instability is widely appreciated to be a hallmark of cancer [45, 46] leading to the acquisition of variants which include translocations [47, 48], copy number variation [49], and chromosome shattering [50, 51].

Current methods for detecting genetic variation include microarrays for detecting single nucleotide variants (SNV's) or discordant paired read alignment [52] for detecting small events. Short reads, while useful for identifying small insertions/deletions (indels), often lack the power to call insertions or complex structural variants in repetitive regions [39]. Short reads can easily be confused by

repetitive DNA sequences [53], which are estimated to cover approximately half of the human genome. Approximately half of the human genome comprises repetitive sequences [53], further confounding the ability to make structural variant calls.

Single molecule restriction maps provide low resolution but long range information about genomic structure in the form of ordered restriction fragment lengths. Such restriction patterns can be compared to a target reference genome to detect large scale structural variants and copy number variation.

In this chapter we will review the state of the art methods used for calling large scale structural variants using ordered restriction maps and present a new methodology for calling structural variants using both global alignments and partial alignments of single molecule maps.

4.2 Calling Structural Variants From Consensus Restriction Maps

Two recent works have used optical mapping and nanocoding mapping data to call structural variants between the genomes of individual humans and the human reference. In both works, variants were called by first assembling a consensus map and then comparing the consensus map to the *in silico* digest of the human reference.

4.2.1 Optical Map Assembly for Multiple Myeloma

In recently published work, Gupta et al. [54] assembled optical maps for samples from a multiple myeloma patient. Optical maps were assembled for a normal (non-cancerous) tissue sample and multiple myeloma plasma samples at two time

points, both before (MM-S sample) and after (MM-R sample) drug resistance was acquired. Optical maps for both the MM-S and MM-R samples were constructed through iterative assembly [4] by first clustering Rmaps via alignment to the human reference, then assembling Rmaps using Gentig software, and iterating using the assembled optical map contigs as the reference in each subsequent iteration. Copy number analysis was performed by aligning Rmaps to the reference and observing genomic coverage by position. Copy number for both the normal, MM-R, and MM-S samples were inferred using a hidden Markov model . Differences in copy number between MM-R and normal and MM-S and normal cover approximately one third of the genome, especially at chromosome ends. The drug resistant MM-R sample was observed to have additional copy number variation changes over the MM-S sample, acquired as the disease progressed. Optical map assembly elucidated dozens of large scale chromosomal rearrangements, including the truncation of chromosome 5, both balanced and unbalanced translocations between other chromosomes, and interstitial deletions.

4.2.2 Hybrid Genome Map Assembly using Long Reads

In recently published work, Pendleton et al. [55] have successfully combined sequence assembly and nanocoding map assembly to build a comprehensive analysis of a single human genome (NA12878). First, error corrected PacBio reads were assembled using the Celera assembler to produce a contig NG50 908kb. Independently, BioNano single molecule nanocoding maps were assembled to produce a genome

map contig NG50 of 4.5 Mb. The genome map contigs were then used to place and orient the sequence assembly contigs via restriction pattern alignment. The aligned sequence contigs were then used to join adjacent genome map contigs, thereby improving the hybrid scaffold contiguity. After two iterations of hybrid assembly, a hybrid sequence assembly with 28.8 Mb NG50.

The authors note that independent mechanisms appear to affect the contiguity of sequence assemblies and map assemblies. Sequence assemblies tend to break in repetitive regions at the sequence level. Nanocoding map assemblies tend to break in regions with low restriction site density or in regions where nick sites on opposite strands cause a double stranded break (termed "fragile sites"). As a result of these independent fragmentation mechanisms, iterative hybrid assembly is employed to improve assembly contiguity for both the sequence assembly and the map assembly.

The assembled genome map was used to call large scale structural variants that could not be called with sequencing. The paper highlights that the genome map was used to identify a 206kb insertion in chromosome 5 due to a tandem repeat expansion and a 577 kb inversion in chromosome 1.

4.3 Calling Structural Variants using Partial Alignments

Consensus map assembly has proven useful for calling structural variants, as detailed in the previous section. However, there may be variants present in a sample for which consensus map assembly does not give a complete picture. Cancer is a highly heterogeneous population of cells which often carry many different structural

variants [56]. As a result, there is not a single conensus map that characterizes the variation present in a population of cancer cells. For example, in [17] it is shown that optical mapping data from two adjacent slices of oligodendroglioma tumor show differences in copy number variation. In that study, the depth of coverage was not high enough to produce a consensus assembly independently for each slice, so the Rmaps were pooled to produce a single consensus map for the entire sample. For a heterogeneous population such as cancer, our hypothesis is that split alignments of single molecule maps to the human reference can help elucidate structural changes that would be difficult to capture assembly of consensus maps alone.

4.4 Methods

We have re-engineered the Maligner software to support both global alignments of the query restriction pattern and partial prefix and suffix alignments of a query restriction map to a target reference genome in both the forward and reverse directions. We have called this mode of alignment MalignerVD, for variant detection.

Alignments selected from the ith row correspond to alignments that terminate with the matching of the ith restriction site in the query. Whereas in the global alignment problem we can extract alignment scores from the last row of the dynamic programming table, prefix/suffix alignments correspond to selecting alignments which end at an interior row of the table.

To account for both prefix and suffix alignments in both the forward and

reverse direction, we must align the query to the reference with four different possible orientations: prefix forward, prefix reverse, suffix forward, and suffix reverse, as shown in Figure 4.1. Since our dynamic programming algorithm (Algorithm 1) fills in the dynamic programming table top to bottom, left to right with the query fragments along the rows and the reference fragments across the columns, we must present the query fragments and reference fragments to the alignment functions with the orientations shown in Figure 4.2 for each alignment type. When outputting alignments, care has been taken to output chunk indices relative to the original (i.e., forward) orientations of both the query and the reference restriction maps.

Before selecting the best alignments, we normalize total alignment score using the M-Score method across each row, independently for prefix alignments and suffix alignments. MalignVD outputs the best non-overlapping full alignments, non-overlapping prefix alignments, and non-overlapping suffix alignments to three separate output files, where the alignments are ranked by M-Score.

4.5 Results

We have applied our methods to a nanocoding dataset from a sample of malignant plasma cells extracted from the bone marrow of a patient with multiple myeloma. The dataset of single molecule nanocoding maps using nicking enzyme Nt.BspQI provide high genomic coverage (93.8X) is summarized in Table 4.1.

We aligned the Nmaps to an *in silico* digest of the human reference using MalignerVD and looked for evidence of copy number variation from coverage of

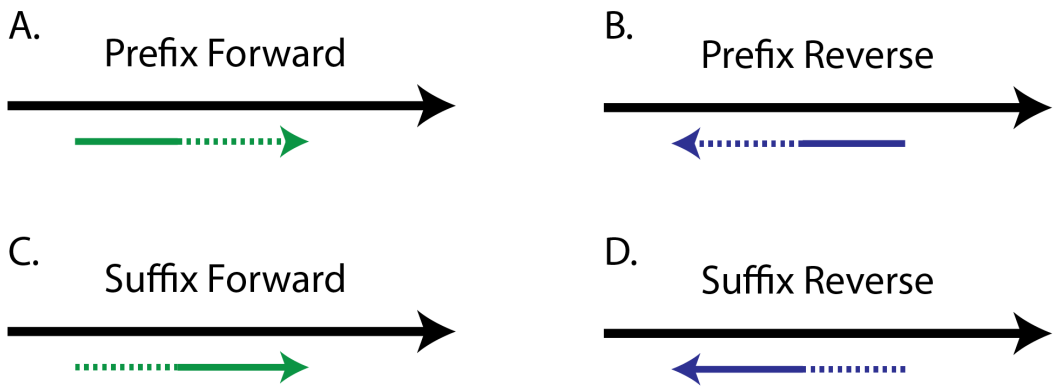

Figure 4.1: **Partial Alignment Types** MalignerVD allows for four different types of partial alignments for query restriction map (green/purple) against a reference restriction map (black). The solid segment of the query represents the aligned portion, and the dashed segment represents the unaligned portion. A) The prefix of the query is aligned in the forward direction with respect to the reference. B) The prefix of the query is aligned in the reverse direction with respect to the reference. C) The suffix of the query is aligned in the forward direction with respect to the reference. D) The suffix of the query is aligned in the reverse direction with respect to the reference.

fully aligned Nmaps with M-Score cutoff ≥ 15, query unmatched site rate $\leq 20\%$, and reference unmatched site rate $\leq 40\%$. Overall 161,226 of the Nmaps aligned uniquely, producing an overall coverage of 12.76X.

We see widespread evidence of copy number variation. Figure 4.3 shows alignment coverage by chromosome across the genome. Signatures of gains in appear in chromosomes 1, 6, 10, 11, 16, 17, 18, 21, and 22 and losses in chromosomes 1, 5, 7, 8, 9, 10, 13, 17, 19, and 21. There appears to be only one copy of chromosome 13. These variations in copy number have also been observed from global alignment coverage of optical mapping data from the same sample [54].

For maps that did not align uniquely, we looked for unique partial alignments of the query prefix or suffix. We only considered partial alignments that were trimmed by at least 10 fragments from the query end and by at least 25% of the number of fragments in the query, with an M-Score cutoff ≥ 15, query unmatched site rate $\leq 20\%$, and reference unmatched site rate $\leq 40\%$. For each query we selected unique prefix and suffix alignments, giving 7052 queries with unique prefix alignments and 6797 queries with unique suffix alignments. 403 queries had both a unique prefix and suffix alignment matching this criteria.

For each partial alignment, we considered whether the alignment is on the left side (i.e., prefix forward or suffix reverse alignments) or the right side (i.e., prefix reverse or suffix forward) of the implied breakpoint. We plotted the left side coverage and the right side coverage in Figure 4.4. We see evidence of breakpoints with partial coverage on both sides, suggesting the breakpoints are associated with insertions, inversions, or translocations. In addition, there are loci with partial

alignment coverage on just one side, which suggests a deletion from the reference.

4.6 Conclusions

We have modified the Maligner software to support partial alignments of query restriction maps against a reference. We've applied our methods to a nanocoding dataset of plasma cells obtained from the bone marrow of a male patient with multiple myeloma. From the global alignment coverage we see evidence of copy number variation which has been previously observed using optical mapping data for the same sample [54]. From the coverage of partial alignments, we have identified candidate breakpoints for insertion, translocation, deletion, and inversion events.

4.7 Availability

MalignerVD is available with the Maligner software suite. The code is available at `https://github.com/LeeMendelowitz/maligner` under the GNU General Public License.

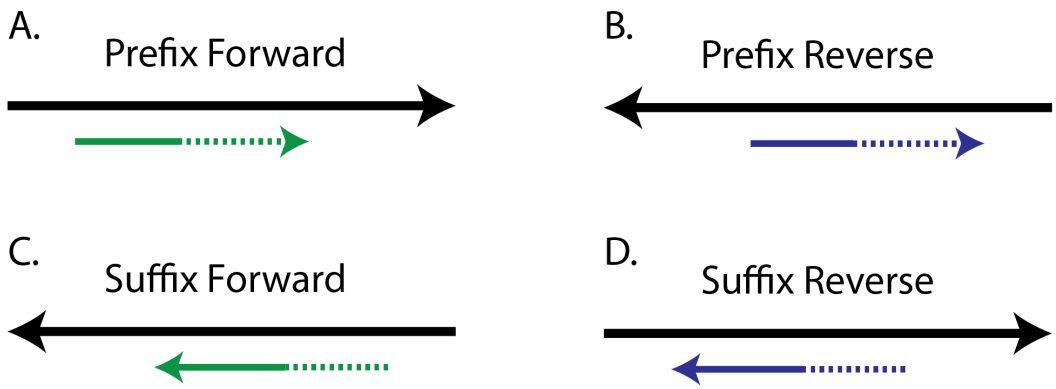

Figure 4.2: **Partial alignment types for dynamic programming** Same alignment types as shown in 4.1 with orientations displayed as required by the dynamic programming implementation. The dynamic programming proceeds by filling out the score table from top to bottom, left to right as described in Algorithm 1, and partial alignments are selected by backtracking from an interior row. The requires aligning to the reverse of the reference for the prefix reverse (B) and suffix forward (C) alignment types.

Number of Maps	1,168,842
Total Map Length	281.4 Gb
Avg. Map Length	240.8 kb
Avg. Fragments per map	20.8
Avg. Fragment Length	11.3 kb
Avg. Fragment Length (5% trimmed mean)	9.5 kb

Table 4.1: **Multiple Myeloma Nmap Summary** Summary of the Nmap data set from the Multiple Myeloma patient.

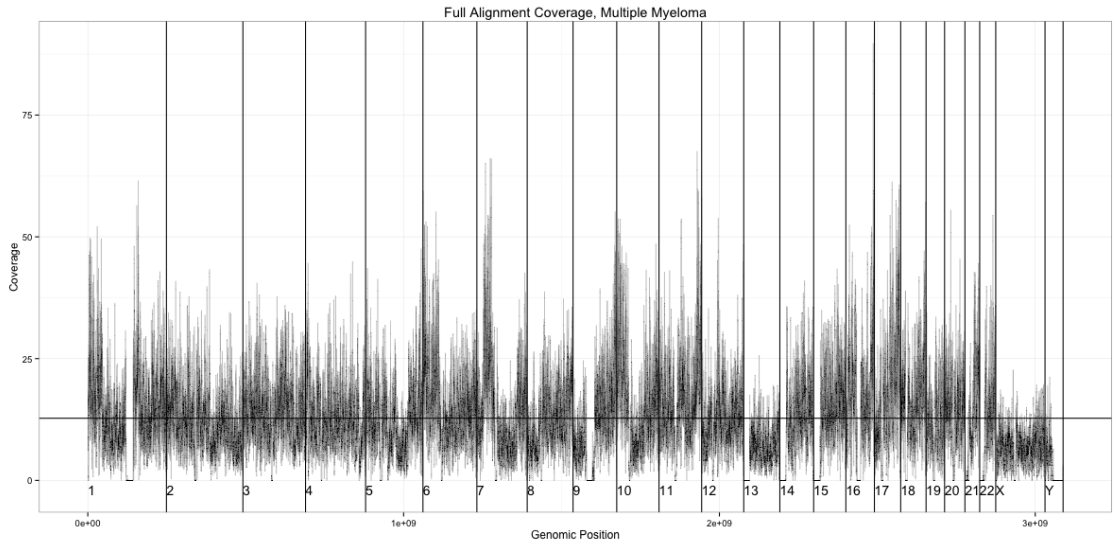

Figure 4.3: **Multiple Myeloma, Full Alignment Coverage** Full alignment coverage of multiple myeloma nanocoding dataset in 50kb bins. The vertical lines delineate the chromosomes, which are arranged in order left to right. The horizontal line represents the average global coverage (12.76X). There are clear signatures of gains in copy number in chromosomes 1, 6, 10, 11, 16, 17, 18, 21, and 22. Losses in copy number appear in chromosomes 1, 5, 7, 8, 9, 10, 13, 17, 19, and 21. There is only one copy of chromosome 13.

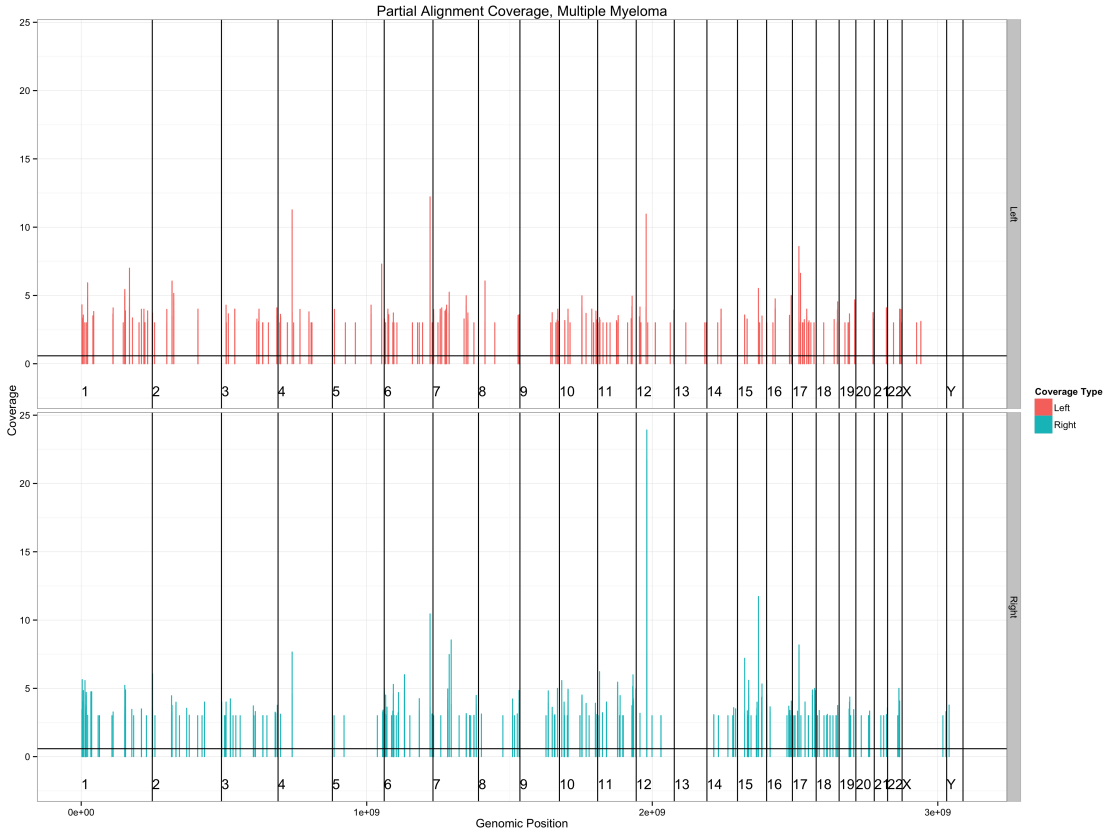

Figure 4.4: **Multiple Myeloma, Partial Alignment Coverage** Partial alignment coverage of multiple myeloma nanocoding dataset in 50kb bins. The vertical lines delineate the chromosomes, which are arranged in order left to right. Bins are marked with colored vertical bars if the partial alignment coverage on the left or right side of the implied breakpoint is above 3, where the bars indicate the depth of coverage.

Chapter 5: Additional Work: Improving String Graph Assembly Contiguity Using Paired Reads

5.1 Introduction

Genome assembly is the computational task of reconstructing the DNA sequence of an organism from a set of sequence reads obtained in a sequencing experiment. As the reads generated in a sequencing experiment are much shorter than the genome itself, assembly is often a vital first step in many genomic analyses, such as identifying genes or characterizing structural variants between genomes. The goal of *de novo* genome assembly is to reconstruct an organism's genome as contiguously and accurately as possible from a set of sequence reads, without any *a priori* assumptions as to the genomic content.

Sequencing experiments can generate a pair of reads from opposite ends of the same DNA fragment, producing a left read and right read with known relative orientation and approximately known separation. However, assemblers initially ignore read pair information, instead relying on sequence overlaps alone to piece together reads into contigs. Read pair information is generally only used *after* contig assembly, where read pairs that link different contigs are used to place contigs in linear

arrangements known as scaffolds.

Here, we explore incorporating read pair information in the string graph assembly paradigm. First, we provide an overview de Bruijn graph assembly and discuss some recent work for incorporating read pair information in this framework. Next, we discuss string graph assembly and demonstrate the effect that false overlap edges have on assembly contiguity. Finally, we present a method to use short-insert read pairs to remove false edges from the string graph to improve assembly contiguity.

5.2 de Bruijn Graph Assembly

5.2.1 Overview

Most genome assemblers designed to work with high-throughput sequencing reads rely on the de Brujin graph paradigm, including Velvet [57], Abyss [58], SOAPDenovo [59], ALLPATHS-LG [60, 61], and SPAdes [62]. A de Bruijn graph $DBG_k = (V_{DBG_k}, E_{DBG_k})$ is constructed from a read set $\mathcal{R}$ by creating a vertex for each length k substring (k-mer) that appears in $\mathcal{R}$, and a directed edge $v_1 \rightarrow v_2$ if a read has a (k+1) length substring x such that $x[0, k] = v_1$ and $x[1, k + 1] = v_2$. The de Bruijn graph is then compressed by merging branch free paths into a single unipath edge, labeled with the path sequence. The de Bruijn graph approach elegantly avoids the cost of computing pairwise overlaps between sequence reads, which is the computational bottleneck in overlap based assemblers.

However, the apparent simplicity of the de Bruijn graph paradigm comes at a cost in assembly contiguity. Useful contiguity information provided by each read

sequence is lost when each read is transformed into a read path of k-mers. Each k-mer that appears at the boundary of a genomic repeat of at least k bp results in a branch vertex $v \in V_{DBG_k}$, with indegree $\deg^-(v) > 1$ or outdegree $\deg^+(v) > 1$. Therefore, by its construction, the de Bruijn graph is fragmented by repeats that are shorter than the read length. While increasing k can reduce the number of repeat vertices, this comes at the risk of disconnecting the graph in regions of low coverage or high sequencing error for graphs constructed on experimental read sets.

Those reads that span multiple unipath edges in the compressed de Bruijn provide potentially useful information in resolving repeats, but applying this information is difficult. The Eulerian Superpath Problem [63] is the task of identifying a path through DBG_k that visits each edge once (i.e. an Eulerian tour) such that it contains each read path as a subpath. This problem is NP-Hard [64, 65]. In [63] Pevzner et al. define several graph simplification operations which preserve solutions to the Eulerian Superpath Problem, but these operations are not applicable in complex regions of the graph. Similarly, the initial version of SOAPdenovo restricts the use of read paths to situations where there is a repeat $r \in V_{DBG_k}$ with $\deg^-(r) = \deg^+(r) = N$ such that each incoming edge can be matched to an outgoing edge without conflict [59].

5.2.2 Paired Read Assembly

Paired read information is typically used only after contig assembly to make distance estimates between contigs, which are used to create linear arrangements

of contigs known as scaffolds. For example, ALLPATHS [60] performs de Bruijn graph assembly, followed by local assembly in scaffold gaps by using a subset of overlapping reads that have been localized to each gap. ALLPATHS-LG [61] also uses paired reads in a "read doubling" preprocessing step, where overlapping left and right reads of the same read pair are merged into a single longer read. However, paired reads are not explicitly used in contig construction.

Some recent work has explored the incorporation paired read information into de Bruijn graph contig assembly. Medvedev et al. [66] introduce a method to perform contig assembly and scaffolding simultaneously with a paired de Bruijn graph. In this approach, a vertex is created for each pair of k-mers with approximately known separation. Two k-mer pairs where the left k-mers are identical in sequence and the right k-mers are separated by a short path in the de Bruijn graph are collapsed into a single vertex, indicating that the k-mer pairs may belong to the same genomic region. However, this heuristic may be overly aggressive in joining k-mer pairs that originate from separate genomic regions.

Pham et al. [67] extend this idea by using paired k-mers to make distance estimates between unipath edges. For each distance estimate, a set of compatible paths is found. These pathsets are refined by filtering out paths that likely do not correspond to true genomic paths. An overlap graph is then constructed on the selected paths, which are assembled into contigs.

The success of such path-finding methods crucially depends on the contiguity of the underlying assembly graph, as complex graph regions can result in too many compatible paths. In the next section, we review the string graph, whose main

advantage over the de Bruijn graph is that it maintains the full contiguity of each read and therefore is less prone to the problem of too many paths.

5.3 String Graph Assembly

5.3.1 Definitions

The string graph, introduced by Myers in [68], is constructed from pairwise overlaps between non-contained reads. To simplify the discussion, we only consider perfect overlaps (i.e. with identity $\epsilon = 1.0$) between the reads.

Denote $\mathcal{R}$ as the set of all reads obtained in a sequencing experiment. A read $r \in \mathcal{R}$ is given by a string of characters $r[1]r[2]\ldots r\left[|r|\right]$ over the alphabet $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. The positions between characters are indexed starting at 0, so that the substring $r[a, b]$ with $a < b$ is the substring of r given by $r[a+1]\ldots r[b]$. The reverse-complement of a string s is given by $rc(s)$. A read r is contained if $\exists w \neq r \in \mathcal{R}$ such that r or $rc(r)$ is a substring of w. As contained reads and duplicate reads can be removed in a preprocessing step, we take $\mathcal{R}$ to be a set of non-contained reads.

A read interval $I = [s, e]$ specifies the coordinates of a substring for a read r, where $0 \leq I.s < I.e \leq |r|$. A read interval I is left-extreme if $I.s = 0$ and right-extreme if $I.e = |r|$. We define $r[I] = r[I.s, I.e]$ to be the substring of length $|I| = I.e - I.s$ described by the interval.

An overlap $o = \{(r_j, I_j), (r_k, I_k)\}$ with length $|o| = \max\left\{|I_j|, |I_k|\right\}$ between reads r_j and $r_k \in \mathcal{R}$ is specified by a pair of extreme read intervals I_j and I_k. Note

that because $\mathcal{R}$ is free of contained reads, the overlap intervals I_j, I_k must be either left-extreme or right-extreme, but cannot be both. In other words, I_j and I_k specify an overlapping prefix or suffix of the reads r_j and r_k.

Let $\rho_o(r_j)$ specify the end of of read r_j participating in overlap o, where $\rho_o(r_j) =$ B if I_j is left-extreme and $\rho_o(r_j) =$ E if I_j is right-extreme. There are four type of overlaps, depending on the values of $\rho_o(r_j)$ and $\rho_o(r_k)$, which are summarized in Table 5.1.

Given a read set $\mathcal{R}$ and the set of all non-contained perfect overlaps $\mathcal{O}$, we take the set of overlaps $\mathcal{O}_k = \{o \in \mathcal{O} \mid |o| \geq k\}$ The overlap graph $G_{\mathrm{OL}_k} = (V_{\mathrm{OL}}, E_{\mathrm{OL}_k})$ is constructed with $V_{\mathrm{OL}} = \mathcal{R}$. The graph is bidirected, so that $E_{\mathrm{OL}_k} \subseteq (V \times \Omega) \times (V \times \Omega)$ where $\Omega = \{\mathrm{B}, \mathrm{E}\}$ is the set of two possible vertex ends. For each overlap $o \in \mathcal{O}_k$ there exists a bidirected edge $e(o) = \{(r_j, \rho_o(r_j)), (r_k, \rho_o(r_k))\} \in E_{\mathrm{OL}}$.

An overlap graph path is given by an ordered listing of edges such that each pair of consecutive edges enters a vertex on one end and exits at the opposite end. For example, the path $p = (\{(w, B), (x, B)\}, \{(x, E), (y, E)\}, \{(y, B), (z, E)\})$ is a valid overlap graph path. Such a path is read-coherent, as it establishes an orientation for each read occurrence. A read r is oriented forward if the path enters r at end B or leaves r at end E. In this example path p, vertex x is forwards, and w, y, and z are reverse. Furthermore, each path p gives a tiling of the reads which spells a sequence, where we use the reverse-complement of a read if has a reverse orientation on the path.

An edge $e(o) = \{(x, \rho_o(x)), (z, \rho_o(z))\}$ is a transitive edge if the overlap o between x and z is implied by some read-coherent path

$$p = \left\{\left(x, \rho_{o\prime(x)}\right), \left(y, \rho_{o\prime(y)}\right)\right\}\left\{\left(y, \rho_{o\prime\prime}(y)\right), \left(z, \rho_{o\prime\prime(z)}\right)\right\}$$

where $\rho_o(x) = \rho_{o\prime}(x)$ and $\rho_o(z) = \rho_{o\prime\prime}(z)$. If such overlaps $o\prime$ and $o\prime\prime$ exist, then the edge $e(o)$ is redundant and can be removed from the graph without changing the set of sequences which could be generated by paths in G_{OL_k}.

A path p is branch-free if $\deg(x, \rho_x) = \deg(y, \rho_y) = 1$ for each edge $\{(x, \rho_x)), (y, \rho_y)\} \in p$. The string graph $SG_k = (V_{SG_k}, E_{SG_k})$ is obtained from G_{OL_k} by removing all transitive edges and then compressing all branch free paths into a single vertex.

5.3.2 False Edges

Let $\mathcal{G}$ be the DNA sequence of a genome, and $\mathcal{R}_\mathcal{G}$ be a set of non-contained error free sequence reads sampled from $\mathcal{G}$ or $rc(\mathcal{G})$. Denote the string graph built from $\mathcal{R}_G$ using minimum overlap k as $SG_k(\mathcal{R}_\mathcal{G}) = (V_{SG_k}, E_{SG_k})$. An edge $e = \{(x, \rho_x)), (y, \rho_y)\} \in E_{SG_k}$ is called a false edge if the sequence of the single-edge path $p = (e)$ is not a substring of $\mathcal{G}$. Such an edge is false because it links two nonadjacent substrings of $\mathcal{G}$ which happen to have a perfect overlap of at least k bp.

Consider the case of error-free reads $\mathcal{R}_\mathcal{G}$ of length L sampled from each position of a genome $\mathcal{G}$. In the ideal de Bruijn Graph $DBG_k(\mathcal{R}_\mathcal{G})$, each edge represents a k+1 bp substring of $\mathcal{G}$. Therefore, the ideal de Bruijn graph has no false edges.

In the ideal string graph $SG_k(\mathcal{R}_\mathcal{G})$, not all edges represent a substring of $\mathcal{G}$ as some edges will be false. While false edges are detrimental to assembly contiguity,

removing such edges provides an opportunity to further improve assembly contiguity. This same opportunity does not exist in the de Bruijn graph paradigm, as all edges are true edges.

If it were possible to remove all false edges from $SG_k(\mathcal{R}_\mathcal{G})$, then any remaining branch is due to a vertex r with multiple true overlaps at the same end. As these overlap edges are true edges, each edge generates a different substring of $\mathcal{G}$. Therefore, r is a repeat substring of $\mathcal{G}$. From the construction of the string graph, we know that r is a sequence of a least L bp. This proves that, after removing all false edges, the string graph only has a branch for a repeat of length L or greater.

To illustrate the prevalence of false overlap edges, we sampled perfect error-free reads of length $L = 100$ from every possible point of origin in the finished *E. coli* K-12 MG1655 reference genome. We constructed the ideal string graph on this read set using the String Graph Assembler (SGA) [69], requiring perfect overlaps of length $>= k$ for several different values of k. We note that true non-transitive overlaps must have length 99 bp, as we sampled reads from each genomic position. The characteristics of the assembly graphs are summarized in Table 5.2. As the table shows, there are many false overlaps for commonly used values of the minimum overlap k. For example, for $k = 50$ bp, we see that 26.6% of the overlaps in the graph are false.

Removal of all false edges from each of these string graphs —regardless of the initial value of k —yields a string graph with 444 vertices, 598 edges, and an NG50 of 125,651 bp. In this simulation, we are able to identify false edges because we know the genomic sequence from which the reads were sampled. In the next section, we

describe a method for identifying false overlap edges from the string graph using paired reads.

5.4 Methods

In *de novo* genome assembly, $\mathcal{G}$ is not known, and so false edges cannot be identified by their defining property that they connect sequences which are not adjacent in $\mathcal{G}$.

Here we propose a simple method for using paired read information to identify false edges from the string graph. Our objective is to remove edges which are not consistent with paired reads and retain overlaps which are supported by paired reads. Given a set of pairwise distance estimates between string graph vertices, we eliminate untrusted overlaps as follows: (i) find the set of all possible closure paths consistent with each distance estimate, (ii) use closure paths to assign a read pair score to each edge, and (iii) remove string graph edges with a read pair score less than a specified threshold T.

5.4.1 Distance Estimates

Pairwise distance estimates between vertices are made by aligning reads to the set of string graph contigs given by V_{SG}. We assume that the lengths of the fragments sampled from $\mathcal{G}$ are independent and identically distributed (i.i.d.), where the length of the ith sequence fragment is a random variable $F_i \sim \mathcal{N}(\mu, \sigma^2)$. In this discussion we assume that the left and right reads have forward and reverse

description	$\rho_o(r_j)$	$\rho_o(r_k)$	relation
prefix - prefix	B	B	$r_j[I_j] = rc(r_k[I_k])$
prefix - suffix	B	E	$r_j[I_j] = r_k[I_k]$
suffix - prefix	E	B	$r_j[I_j] = r_k[I_k]$
suffix - suffix	E	E	$r_j[I_j] = rc(r_k[I_k])$

Table 5.1: **Types of overlaps** Description of four possible perfect overlaps $o = \{(r_j, I_j), (r_k, I_k)\}$ between reads r_j and r_k, and the corresponding sequence relation. The reverse-complement of s is denoted $rc(s)$.

k (bp)	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert E_F \rvert$	$\lvert E_F \rvert / \lvert E \rvert$	$\lvert E \rvert / \lvert V \rvert$	NG50 (bp)
20	12,277	39,077	26,646	0.682	3.183	8,081
30	4,815	13,075	8,106	0.620	2.715	21,805
40	2,172	3,515	1,189	0.338	1.618	37,630
50	1,321	2,010	535	0.266	1.522	58,833
60	1,025	1,538	359	0.233	1.500	63,676
70	888	1,328	286	0.215	1.495	78,692
80	702	1,011	155	0.153	1.440	78,692
90	540	756	62	0.082	1.400	107,887
without false edges	444	598	0	0.000	1.347	125,651

Table 5.2: **Ideal string graph characteristics** Characteristics of the ideal string graph for error-free 100 bp reads from *E. Coli* K12 MG1655, as a function of the minimum overlap length k. V is the set of string graph vertices, E is the set of string graph edges (overlaps), $E_F \subset E$ is the set of false overlaps.

orientation respectively, but distance estimates can be made using any fixed relative orientation between read pairs.

Read pairs which align to the same vertex with the correct relative orientation are used to compute the sample mean $\hat{\mu}$ and the sample variance $\hat{\sigma}^2$. Concordant alignments to the same contig are used to build the empirical distribution of insert sizes as a histogram. Outliers can distort the sample mean for the insert size distribution, so insert sizes $f_i \notin [\hat{\mu} - 3\hat{\sigma}, \hat{\mu} + 3\hat{\sigma}]$ are iteratively removed until $\hat{\mu}$ and $\hat{\sigma}$ converge.

Read pairs where the left read and right read align to different contigs are used to make distance estimates between contigs. Consider the case where the left read of the ith read pair is aligned $d_{i,1}$ bases from and oriented towards end $\rho_{i,1}$ of vertex $v_{i,1}$, and the right read is aligned $d_{i,2}$ bases from and oriented towards end $\rho_{i,2}$ of vertex $v_{i,2}$. Each such read pair defines a link l_i between end $\rho_{i,1}$ of vertex $v_{i,1}$ and end $\rho_{i,2}$ of vertex $v_{i,2}$, with an estimated gap of $\hat{g}_i = \hat{\mu} - (d_{i,1} + d_{i,2})$. If $\hat{g}_i < 0$, then the gap is estimated to be an overlap.

Links between between the same ends of the same contig pair are clustered to form a single distance estimate. Potentially erroneous links are filtered out by identifying the maximum subset $\mathcal{B}$ of mutually compatible links, where two links l_i and l_j are compatible if $|g_i - g_j| < 6\hat{\sigma}$. If necessary, we break ties by selecting the subset with the smallest variance in distance estimates. The distance estimate computed on the filtered set of links $\mathcal{B}$ is given by $D(\mathcal{B}) = (\hat{g}, s_{\hat{g}})$, where:

$$\hat{g} = \frac{1}{|\mathcal{B}|} \sum_{l_i \in \mathcal{B}} \hat{g}_i \qquad s_{\hat{g}} = \frac{1}{\sqrt{|\mathcal{B}|}} \hat{\sigma} \tag{5.1}$$

It can be shown that $\hat{g}$ is the maximum likelihood estimate (MLE) for the gap between contigs, under the assumption that the fragment lengths are i.i.d and normally distributed.

5.4.2 Finding paths

For each distance estimate, we seek the set of all closure paths from v_i exiting at end ρ_i to v_j entering at end ρ_j with path length $d_{\text{path}} \in [d_{\min}, d_{\max}]$ where $d_{\min} = \hat{g} - 3s_{\hat{g}}$, $d_{\max} = \hat{g} + 3s_{\hat{g}}$. We first attempt to find the set of all closure paths using a bounded breadth first search (BFS).

For complex graph regions where the BFS fails due to too many candidate paths, we use a bounded depth-first search (DFS) on a subgraph of edges that are guaranteed to belong to a compatible path. First, we collect the set of edges reachable from v_i out of end ρ_i on a path of distance $\leq \frac{d_{\max}}{2}$ and the set of edges reachable from v_j out of end ρ_j on a path of distance $\leq \frac{d_{\max}}{2}$ using Dijkstra's algorithm. Taking the subgraph formed by the union of these edge sets, we use Dijkstra's algorithm to compute the shortest distance from both v_i out of end ρ_i and v_j out of end ρ_j to each subgraph edge and remove any edges that cannot be on a path with $d_{\text{path}} \leq d_{\max}$. This is repeated until the subgraph edge set converges, such that all remaining edges are guaranteed to be on a compatible path. Lastly, we use a bounded DFS to identify all compatible paths. If there are too many paths, or if the DFS takes too many steps, we terminate the search and return no closure paths.

5.4.3 Identifying False Edges

A read pair score $s(e)$ is assigned to each string graph edge e to measure the number of read pairs which support the edge. All edges are initialized with a read pair score of zero. For a distance estimate supported by n read pairs with $m \geq 1$ closure paths, we add a weight of $\frac{n}{m}$ for each instance that an edge appears in a closure path. Even if there are multiple closure paths for a distance estimate, a string graph edge can receive the full read pair score of n if it appears in each of the m closure paths.

Each string graph edge with a read pair score less than T is classified as a false edge and is removed from the string graph. Lastly, the string graph is simplified by compressing unipaths using the remaining trusted edges.

5.5 Results

We have implemented these methods as a new module within SGA called `sga close-path`. We have tested our methods on a dataset of 20.9 million paired, 101 bp reads with insert size 180 bp sequenced from *E. Coli* K-12 MG1655 with Illumina HiSeq 2000 (SRR447664).

We use SGA v.0.10.0 to build the string graph SG_k on the read set. We first preprocess the reads by soft-clipping based on quality scores, followed by error correction and filtering of duplicate reads. We overlapped the reads using perfect overlaps of minimum length 20, and then assembled the reads using different minimum overlaps of length $k = 20, 30, \ldots, 90$. The recipe to produce the string graph

```
sga preprocess -p 1 -q 30
sga correct -k 31 --discard --learn
sga filter -k 27 -x 3 --homopolymer-check --low-complexity-check
sga overlap -m 20
sga assemble -m k
```

Table 5.3:
SGA assembly parameters for *E. Coli* K-12 MG1655 dataset. Values $k = 20, 30, \ldots, 90$ were used for the minimum overlap in `sga assemble`

is provided in Table 5.3.

The most contiguous SGA assembly on this dataset was obtained using minimum overlap of $k = 60$ bp (NG50: 52.6 kb). However, we applied `sga close-path` to the string graph assembled with minimum overlap of $k = 40$ bp (NG50: 30.9 kb) as previous experience has shown that sequencing errors and variation in coverage results in short edges appearing in true genomic paths, and these shorter edges must be retained in order to preserve assembly contiguity.

We used `bwa-mem` [70] to align the trimmed reads from `sga pre-process` to the `sga` $k = 40$ string graph. 98.2% of read pairs had a unique alignment of 98% identity or higher for both the left and right read. Read pairs with multiple alignments were ignored in downstream analysis. The insert size distribution was computed using reads that aligned concordantly to the same contig, giving $\hat{\mu} = 180.1$, and $\hat{\sigma} = 24.9$ bp. Read pairs that aligned to different contigs provided 98,590 links, which were clustered into 5,452 distance estimates.

We ran `sga close-path` using the 3,607 distance estimates supported by at least two or more links. A path closure was found for 98.5% of the links, and 97.4% of the links had a unique path closure. The closure paths were used to assign a score to each edge. The cumulative distribution of edge scores is shown in Figure 5.5. We

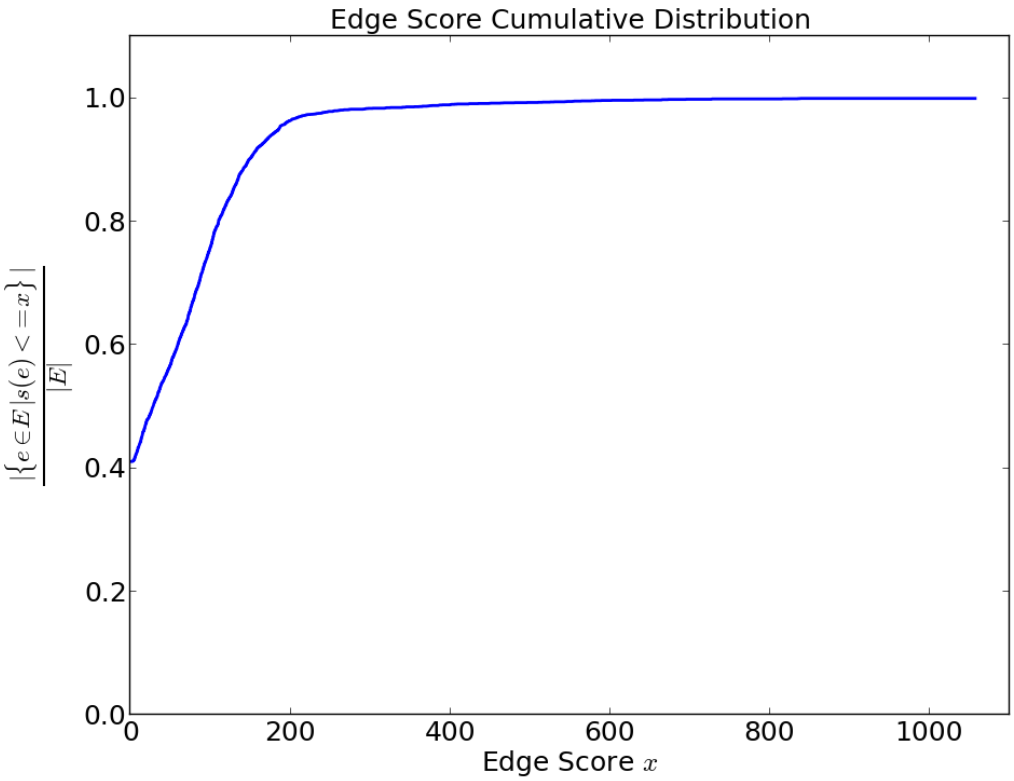

Figure 5.1:
Cumulative distribution of edge scores The cumulative distribution of edge scores $s(e)$ for all edges e in the `sga` $k = 40$ string graph. 41.1% of the edges have a score of 0.0, indicating that they are not covered by any closure paths.

classified 1,510 out of 3,648 edges as untrusted by requiring a minimum edge score of $T = 5$, and these untrusted edges were removed from the graph. We used `sga assemble` to compress any newly formed unipaths, yielding a simplified graph with 287 vertices, 350 edges, and NG50 78.7 kb.

We evaluated our ability to correctly identify false edges by aligning contigs to the reference genome using `nucmer` [71] to determine their true point of origin. Using only alignments with minimum identity 99% covering 99% of the contig sequence, we label each string graph edge as a false edge if the contigs connected by the edge

do not have compatible overlapping alignments to the reference. This identified 1044 out of 3648 edges (28.6%) as false edges, which matches our expectations from the simulation results reported in Table 5.2.

Figure 5.5 shows the true positive rate vs. false positive rate for predicting false edges for all thresholds $T > 0$. These empirical results show that our method is effective at filtering out nearly all false edges, but some true edges are filtered out as well. For our choice of $T = 5$, 98.1% of all false edges and 19.2% of all true edges were removed from the graph. The removal of true edges might be attributed to missing distance estimates (and therefore missing closure paths) due to the policy of ignoring reads with multiple alignments, but this needs to be explored further.

We evaluated the correctness of the `sga close-path` $k = 40$ assembly by aligning contigs to the reference genome using `nucmer`, which found alignments of identity 98% or better allowing for indels of at most 5 bp. `dnadiff` was used to generate a one-to-one tiling of contig alignments against the reference. Alignment records for a contig with a split alignment were merged into a single record if the alignments appeared consecutively in the tiling and did not imply an assembly error larger than 100 bp. Corrected NG50 was computed on this post-processed set of alignment records.

The `sga close-path` $k = 40$ assembly produced two contigs that had split alignments to the reference. Manual inspection revealed that these split alignments were due to small scale indels of 8 bp and 91 bp due to different copy numbers of tandem repeat sequences. These small assembly errors were not penalized in our computation of corrected NG50.

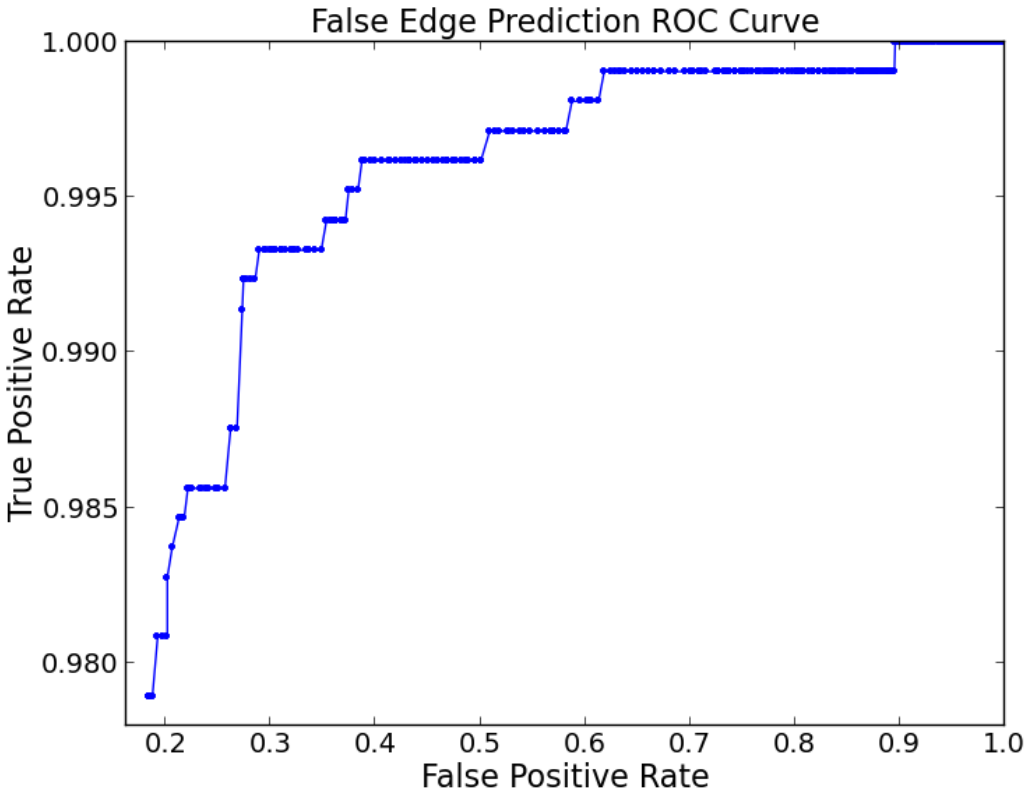

Figure 5.2: **String Graph False Edge ROC Curve** True positive rate vs. false positive rate for prediction of false edges in the `sga` $k = 40$ string graph for all choices of threshold $T > 0$. An edge is predicted to be false if $s(e) < T$.

We compare the `sga close-path` assembly to a `SOAPdenovo2` assembly of the same error corrected reads. As `sga close-path` does not use paired reads to build scaffolds, we compared our assembly to contig assemblies produced by `SOAPdenovo2 contig`. We tried assembling with k = 31, 41, 51, and 61, and report the most contiguous contig assembly here.

5.5.1 Implementation

The contig distance estimation code has been implemented in Python. The closure path algorithms have been implemented in `C++` as part of the String Graph Assembler. The source code is available at `https://github.com/LeeMendelowitz/sga-close-path`.

5.6 Conclusions

In this work, we have discussed the relative strengths and weaknesses of both de Bruijn graph and string graph assembly paradigms. We have demonstrated that a significant fraction of overlap edges in the string graphs computed on 100 bp reads are typically false overlaps induced by short repeats. We have presented a simple method for detecting and removing these edges using paired reads, and demonstrate its effectiveness on a bacterial dataset. For this dataset `sga close-path` produced a more contiguous contig assembly than `sga` or `SOAPdenovo2` without introducing additional assembly errors.

While `sga close-path` has been demonstrated on a dataset with insert size

180 bp, improvements can be made to adopt this method to longer insert libraries, where there are likely to be many more closure paths for each distance estimate. For instance, it would be useful to pool together information from multiple distance estimates to constrain the search for closure paths in repetitive graph regions.

5.7 Acknowledgements

LMM would like to acknowledge Jared T. Simpson for helpful discussions on SGA and the merits of string graph assembly. This work has been supported by grants NSF IIS-1117247 to MP and NIH R01-HG-000225 to DC Schwartz, subcontract to MP.

Assembly	Size (bp)	$\|V\|$	$\|E\|$	NG50 (kb)	Corr. NG50 (kb)	Errors
`sga` $k = 40$	4,799,648	2,312	3,648	30.9	30.9	0
`sga` $k = 60$	4,670,902	1,055	1,540	52.6	52.6	0
`sga close-path` $k = 40$	4,593,022	287	350	78.7	78.7	0
`SOAPdenovo2` $k = 51$	4,597,804	750	1,500	33.0	33.0	0

Table 5.4: **Contig assembly results for the *E. Coli* K-12 MG1655** Contig assembly results for the *E. Coli* K-12 MG1655 dataset (SRR447664). Assembly errors count the number of relocation events greater then 100 bp, as determined by `nucmer` alignments to the reference genome. Corrected N50 was computed by breaking contigs for each such error, none of which were observed in these contig assemblies. The reference genome size $G = 4{,}639{,}675$ bp.

Chapter 6: Conclusions

In this dissertation I have presented algorithms developed for the alignment and visualization of restriction mapping data.

Maligner is open-source software for the alignment of both *in silico* digests of sequence assembly contigs and single molecule restriction maps to a reference restriction map. Maligner provides two modes of alignment: both a bounded, dynamic programming implementation and a faster but less sensitive index based method which does not allow for unmatched sites in the query. We have compared both implementations to other available software tools and demonstrate that Maligner finds more correct alignments in comparable run time. We also show that the dynamic programming implementation of Maligner is the most appropriate choice when working with noisy experimental restriction mapping data. In addition, we have introduced the concept of M-Score for the normalization of alignment scores across queries to assist with selecting significant alignments, thereby avoiding a computationally expensive permutation test. Finally, we show that our methods scale to larger genomes by aligning a high-coverage optical mapping set for budgerigar.

We have also provided an overview of MalignViz, which is a portable web application for visualizing pairwise alignments of restriction maps generated by the

Maligner software. MalignViz has proven to be a useful tool for suggesting features that have been incorporated back into the Maligner software, such as query rescaling and query re-alignment for correcting matched sites.

We have presented MalignerVD, which is a re-engineering of the Maligner dynamic programming implementation to allow for partial alignments of a prefix or suffix of the query restriction pattern to a reference, with the application of calling breakpoints for structural variants. We have applied these methods to a multiple myeloma nanocoding dataset and observed widespread changes in copy number variation.

Lastly, we have presented work for improving the contiguity of sequence assembly by removing false edges from string graphs using paired reads. Such a method could be integrated with restriction mapping data, as in AGORA [18], to further resolve repeats and improve sequence assembly contiguity.

Bibliography

[1] Blake C Meyers, Simone Scalabrin, and Michele Morgante. Mapping and sequencing complex genomes: let's get physical! *Nature Reviews Genetics*, 5(8):578–588, 2004.

[2] David C Schwartz, Xiaojun Li, Luis I Hernandez, Satyadarshan P Ramnarain, Edward J Huff, and Yu-Ker Wang. Ordered restriction maps of saccharomyces cerevisiae chromosomes constructed by optical mapping. *Science*, 262(5130):110–114, 1993.

[3] Jieyi Lin, Rong Qi, Christopher Aston, Junping Jing, Thomas S Anantharaman, Bud Mishra, Owen White, Michael J Daly, Kenneth W Minton, J Craig Venter, et al. Whole-genome shotgun optical mapping of deinococcus radiodurans. *Science*, 285(5433):1558–1562, 1999.

[4] Brian Teague, Michael S Waterman, Steven Goldstein, Konstantinos Potamousis, Shiguo Zhou, Susan Reslewic, Deepayan Sarkar, Anton Valouev, Christopher Churas, Jeffrey M Kidd, et al. High-resolution human genome structure by single-molecule analysis. *Proceedings of the National Academy of Sciences*, 107(24):10848–10853, 2010.

[5] Shiguo Zhou, Andrew Kile, Michael Bechner, Michael Place, Erika Kvikstad, Wen Deng, Jun Wei, Jessica Severin, Rodney Runnheim, Christopher Churas, et al. Single-molecule approach to bacterial genomic comparisons via optical mapping. *Journal of bacteriology*, 186(22):7773–7782, 2004.

[6] Kyubong Jo, Dalia M Dhingra, Theo Odijk, Juan J de Pablo, Michael D Graham, Rod Runnheim, Dan Forrest, and David C Schwartz. A single-molecule barcoding system using nanoslits for dna analysis. *Proceedings of the National Academy of Sciences*, 104(8):2673–2678, 2007.

[7] Anton Valouev, Lei Li, Yu-Chi Liu, David C Schwartz, Yi Yang, Yu Zhang, and Michael S Waterman. Alignment of optical maps. *Journal of Computational Biology*, 13(2):442–462, 2006.

[8] Niranjan Nagarajan, Timothy D Read, and Mihai Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24(10):1229–1235, 2008.

[9] Anton Valouev, David C Schwartz, Shiguo Zhou, and Michael S Waterman. An algorithm for assembly of ordered restriction maps from single dna molecules. *Proceedings of the National Academy of Sciences*, 103(43):15770–15775, 2006.

[10] Martin D Muggli, Simon J Puglisi, and Christina Boucher. Efficient indexed alignment of contigs to optical maps. In *Algorithms in Bioinformatics*, pages 68–81. Springer, 2014.

[11] Deepayan Sarkar, Steve Goldstein, David C Schwartz, and Michael A Newton. Statistical significance of optical map alignments. *Journal of Computational Biology*, 19(5):478–492, 2012.

[12] Thomas S Anantharaman, Bud Mishra, and David C Schwartz. Genomics via optical mapping iii: Contiging genomic dna and variations. In *The Seventh International Conference on Intelligent Systems for Molecular Biology*, volume 7, pages 18–27. Citeseer, 1999.

[13] Shiguo Zhou, Andrew Kile, Erika Kvikstad, Mike Bechner, Jessica Severin, Dan Forrest, Rod Runnheim, Chris Churas, Thomas S Anantharaman, Peter Myler, et al. Shotgun optical mapping of the entire leishmania major friedlin genome. *Molecular and biochemical parasitology*, 138(1):97–106, 2004.

[14] Shiguo Zhou, Michael C Bechner, Michael Place, Chris P Churas, Louise Pape, Sally A Leong, Rod Runnheim, Dan K Forrest, Steve Goldstein, Miron Livny, et al. Validation of rice genome sequence by optical mapping. *BMC genomics*, 8(1):278, 2007.

[15] Shiguo Zhou, Fusheng Wei, John Nguyen, Mike Bechner, Konstantinos Potamousis, Steve Goldstein, Louise Pape, Michael R Mehan, Chris Churas, Shiran Pasternak, et al. A single molecule scaffold for the maize genome. *PLoS Genet*, 5(11):e1000711, 2009.

[16] Ganeshkumar Ganapathy, Jason T Howard, James M Ward, Jianwen Li, Bo Li, Yingrui Li, Yingqi Xiong, Yong Zhang, Shiguo Zhou, David C Schwartz, et al. High-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience*, 3(1):1–9, 2014.

[17] Mohana Ray, Steve Goldstein, Shiguo Zhou, Konstantinos Potamousis, Deepayan Sarkar, Michael A Newton, Elizabeth Esterberg, Christina Kendziorski, Oliver Bogler, and David C Schwartz. Discovery of structural alterations in solid tumor oligodendroglioma by single molecule analysis. *BMC genomics*, 14(1):505, 2013.

[18] Henry C Lin, Steve Goldstein, Lee Mendelowitz, Shiguo Zhou, Joshua Wetzel, David C Schwartz, and Mihai Pop. Agora: assembly guided by optical restriction alignment. *BMC bioinformatics*, 13(1):189, 2012.

[19] Basil Britto Xavier, Julia Sabirova, Moons Pieter, Jean-Pierre Hernalsteens, Henri de Greve, Herman Goossens, and Surbhi Malhotra-Kumar. Employing whole genome mapping for optimal de novo assembly of bacterial genomes. *BMC research notes*, 7(1):484, 2014.

[20] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):1–31, 2013.

[21] Can Alkan, Bradley P Coe, and Evan E Eichler. Genome structural variation discovery and genotyping. *Nature Reviews Genetics*, 12(5):363–376, 2011.

[22] Robert Schmieder and Robert Edwards. Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, 27(6):863–864, 2011.

[23] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[24] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004.

[25] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[26] Mapsolver. `http://opgen.com/genomic-services/softwares/mapsolver`, 2015 (accessed July 12, 2015).

[27] geval: Genome evaluation browser. `http://geval.sanger.ac.uk/`, 2015 (accessed July 12, 2015).

[28] Mongodb. `https://www.mongodb.org`, 2015 (accessed July 11, 2015).

[29] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[30] Angularjs superheroic javascript mvw framework. `http://angularjs.org`, 2015 (accessed July 11, 2015).

[31] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[32] Bootstrap: The world's most popular mobile-first and responsive front-end framework. `http://getbootstrap.com/`, 2015 (accessed July 11, 2015).

[33] The web's scaffolding tool for modern webapps — yeoman. `http://yeoman.io`, 2015 (accessed July 11, 2015).

[34] Grunt: The javascript task runner. `http://gruntjs.com/`, 2015 (accessed July 11, 2015).

[35] Bower. `http://bower.io/`, 2015 (accessed July 11, 2015).

[36] Eray Tuzun, Andrew J Sharp, Jeffrey A Bailey, Rajinder Kaul, V Anne Morrison, Lisa M Pertz, Eric Haugen, Hillary Hayden, Donna Albertson, Daniel Pinkel, et al. Fine-scale structural variation of the human genome. *Nature genetics*, 37(7):727–732, 2005.

[37] 1000 Genomes Project Consortium et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.

[38] Jeffrey M Kidd, Gregory M Cooper, William F Donahue, Hillary S Hayden, Nick Sampas, Tina Graves, Nancy Hansen, Brian Teague, Can Alkan, Francesca Antonacci, et al. Mapping and sequencing of structural variation from eight human genomes. *Nature*, 453(7191):56–64, 2008.

[39] Can Alkan, Bradley P Coe, and Evan E Eichler. Genome structural variation discovery and genotyping. *Nature reviews. Genetics*, 12(5):363–76, May 2011.

[40] Pawel Stankiewicz and James R Lupski. Structural variation in the human genome and its role in disease. *Annual review of medicine*, 61:437–455, 2010.

[41] Jan O Korbel, Tal Tirosh-Wagner, Alexander Eckehart Urban, Xiao-Ning Chen, Maya Kasowski, Li Dai, Fabian Grubert, Chandra Erdman, Michael C Gao, Ken Lange, et al. The genetic architecture of down syndrome phenotypes revealed by high-resolution analysis of human segmental trisomies. *Proceedings of the National Academy of Sciences*, 106(29):12031–12036, 2009.

[42] Christian R Marshall, Abdul Noor, John B Vincent, Anath C Lionel, Lars Feuk, Jennifer Skaug, Mary Shago, Rainald Moessner, Dalila Pinto, Yan Ren, et al. Structural variation of chromosomes in autism spectrum disorder. *The American Journal of Human Genetics*, 82(2):477–488, 2008.

[43] Klaus Fellermann, Daniel E Stange, Elke Schaeffeler, Hartmut Schmalzl, Jan Wehkamp, Charles L Bevins, Walter Reinisch, Alexander Teml, Matthias Schwab, Peter Lichter, et al. A chromosome 8 gene-cluster polymorphism with low human beta-defensin 2 gene copy number predisposes to crohn disease of the colon. *The American Journal of Human Genetics*, 79(3):439–448, 2006.

[44] Maria Karayiorgou, Tony J Simon, and Joseph A Gogos. 22q11. 2 microdeletions: linking dna structural variation to brain dysfunction and schizophrenia. *Nature Reviews Neuroscience*, 11(6):402–416, 2010.

[45] Douglas Hanahan and Robert A Weinberg. Hallmarks of cancer: the next generation. *cell*, 144(5):646–674, 2011.

[46] Donna G Albertson, Colin Collins, Frank McCormick, and Joe W Gray. Chromosome aberrations in solid tumors. *Nature genetics*, 34(4):369–376, 2003.

[47] Samuel F Bunting and Andre Nussenzweig. End-joining, translocations and cancer. *Nature Reviews Cancer*, 13(7):443–454, 2013.

[48] André Nussenzweig and Michel C Nussenzweig. Origin of chromosomal translocations in lymphoid cancer. *Cell*, 141(1):27–38, 2010.

[49] Xiaohui Ni, Minglei Zhuo, Zhe Su, Jianchun Duan, Yan Gao, Zhijie Wang, Chenghang Zong, Hua Bai, Alec R Chapman, Jun Zhao, et al. Reproducible copy number variation patterns among single circulating tumor cells of lung cancer patients. *Proceedings of the National Academy of Sciences*, 110(52):21083–21088, 2013.

[50] Philip J Stephens, Chris D Greenman, Beiyuan Fu, Fengtang Yang, Graham R Bignell, Laura J Mudie, Erin D Pleasance, King Wai Lau, David Beare, Lucy A Stebbings, et al. Massive genomic rearrangement acquired in a single catastrophic event during cancer development. *cell*, 144(1):27–40, 2011.

[51] Josep V Forment, Abderrahmane Kaidi, and Stephen P Jackson. Chromothripsis and cancer: causes and consequences of chromosome shattering. *Nature Reviews Cancer*, 12(10):663–670, 2012.

[52] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[53] Todd J Treangen and Steven L Salzberg. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature reviews. Genetics*, 13(1):36–46, January 2012.

[54] Aditya Gupta, Michael Place, Steven Goldstein, Deepayan Sarkar, Shiguo Zhou, Konstantinos Potamousis, Jaehyup Kim, Claire Flanagan, Yang Li, Michael a. Newton, Natalie S. Callander, Peiman Hematti, Emery H. Bresnick, Jian Ma, Fotis Asimakopoulos, and David C. Schwartz. Single-molecule analysis reveals widespread structural variation in multiple myeloma. *Proceedings of the National Academy of Sciences*, 112(25):201418577, 2015.

[55] Matthew Pendleton, Robert Sebra, Andy Wing Chun Pang, Ajay Ummat, Oscar Franzen, Tobias Rausch, Adrian M Stütz, William Stedman, Thomas Anantharaman, Alex Hastie, Heng Dai, Markus Hsi-Yang Fritz, Han Cao, Ariella Cohain, Gintaras Deikus, Russell E Durrett, Scott C Blanchard, Roger Altman, Chen-Shan Chin, Yan Guo, Ellen E Paxinos, Jan O Korbel, Robert B Darnell, W Richard McCombie, Pui-Yan Kwok, Christopher E Mason, Eric E Schadt, and Ali Bashir. Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nature Methods*, (May), 2015.

[56] Andriy Marusyk and Kornelia Polyak. Tumor heterogeneity: causes and consequences. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1805(1):105–117, 2010.

[57] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821–9, May 2008.

[58] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven J M Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–23, June 2009.

[59] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, Songgang Li, Huanming Yang, Jian Wang, and Jun Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–72, February 2010.

[60] Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya a Shlyakhter, Matthew K Belmonte, Eric S Lander, Chad Nusbaum, and David B Jaffe. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810–20, May 2008.

[61] Sante Gnerre, Iain Maccallum, Dariusz Przybylski, Filipe J Ribeiro, Joshua N Burton, Bruce J Walker, Ted Sharpe, Giles Hall, Terrance P Shea, Sean Sykes, Aaron M Berlin, Daniel Aird, Maura Costello, Riza Daza, Louise Williams, Robert Nicol, Andreas Gnirke, Chad Nusbaum, Eric S Lander, and David B Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences of the United States of America*, 108(4):1513–8, January 2011.

[62] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey a Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, Alexey V Pyshkin, Alexander V Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max a Alekseyev, and Pavel a Pevzner. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology : a journal of computational molecular cell biology*, 19(5):455–77, May 2012.

[63] 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(17):9748–53, August 2001.

[64] Paul Medvedev, Konstantinos Georgiou, Eugene W. Myers, and Michael Brudno. Computability of models for sequence assembly. *Algorithms in Bioinformatics*, pages 289–301, 2007.

[65] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology : a journal of computational molecular cell biology*, 16(7):897–908, July 2009.

[66] Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of computational biology : a journal of computational molecular cell biology*, 18(11):1625–34, November 2011.

[67] Son K Pham, Dmitry Antipov, Alexander Sirotkin, Glenn Tesler, Pavel a Pevzner, and Max a Alekseyev. Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in Genome Assembly. *Journal of computational biology : a journal of computational molecular cell biology*, 20(4):359–371, July 2012.

[68] Eugene W. Myers. The fragment assembly string graph. *Bioinformatics (Oxford, England)*, 21 Suppl 2:ii79–85, September 2005.

[69] Jared T Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, pages 1165–1173, January 2012.

[70] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[71] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, January 2004.