Thesis

# Error Correcting Optical Mapping Data

Submitted by

Darshan Washimkar

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2016

Master's Committee:

Advisor: Christina Boucher

Sangmi Lee Pallickara
Tai Montgomery

## Abstract

### Error Correcting Optical Mapping Data

Optical Mapping is a unique system that is capable of producing high-resolution, high-throughput genomic map data that gives information about the structure of a genome (Schwartz et al., Science 1993). Recently it has been used for scaffolding contigs and assembly validation for large-scale sequencing projects — for example, the maize (Zhou et al., PLoS Genetics, 2009), goat (Dong et al., Nature Biotech. 2013), and amborella (Chamala et al., Science 2013) genomes. However, a major impediment in the use of this data is the variety and quantity of the errors in the raw optical mapping data, which are referred to as Rmaps. The challenges associated with using Rmap data—and thus, optical mapping data—is analogous to dealing with insertion and deletions in the alignment of long reads. Moreover, they are arguably harder since the data is integral and susceptible to inaccuracy. We develop cOMet to tackle error correct Rmap data, which to the best of our knowledge is the is the only non-proprietary error correction method. Our results demonstrate that cOMet has high accuracy on simulated *E. coli* (str. K-12 substr. MG1655) genome.

## Acknowledgements

I am very thankful to my adviser, Dr. Christina Boucher for the time, insight, motivation, and kindness she rendered to me. I also want to thank her for providing me an opportunity to work as teaching and research assistant. She will be an inspiration to me for her diligence and passionate attitude towards work. I would especially like to thank a friend-cum-mentor, Martin Muggli for helping me throughout my research and suggesting the edits in my thesis. I would like to express my gratitude towards my co-advisors Dr. Sangmi Lee Pallickara for always being very supportive and teaching me Big Data. I am grateful towards Dr. Tai Montgomery for accepting to be on my committee and providing his precious time. I have been fortunate to work with Dr. Christos Papadopoulos. I want to thank him for believing and supporting me through this journey. I am very thankful towards Rumpal Kaur, Melinda, Nikhil Agnihotri and Dev for proofreading my thesis. Last but not the least, I would like to express my gratitude towards my family and friends who always help me to become a better person.

# Table of Contents

# List of Tables

## List of Figures

CHAPTER 1

# Introduction

In 1993 Schwartz et al. [17] developed a system, referred to as *optical mapping*, for creating an ordered, genome-wide, high-resolution restriction map of a given organism's genome. Genome-wide optical maps have been used for discovering structural variations and rearrangements [18] as well as for scaffolding and validating contigs for several large sequencing projects. These include those for various prokaryote species [15, 22, 23], rice [24], maize [26], mouse [7], goat [8], parrot [9], and *Amborella trichopoda* [6]. However, even though their use has increased in popularity in the past several years, there is still a lack of publicly available tools for analyzing this data, a point further empathized by Mendelowitz and Pop [12] in 2014: "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 raw optical mapping data is generated by a biological experiment in which large DNA molecules cling to the surface of a microscope slide using electrostatic charge and are digested by using one or more restriction enzymes. The fragments formed by digestion are 'painted' with a fluorescent dye, to allow visibility under laser light and a CCD camera. Restriction enzymes cut the DNA molecule at restriction sites creating smaller fragments. The consolidated intensity of fluorescent dye is used in conjunction with the distance between fragment ends in estimating fragment length. The length of the fragments is determined by image processing and machine learning techniques. The resulting data from an experiment are in the form of an ordered series of fragment lengths [25]. The data for each single molecule produced by the system is referred to as an *Rmap*. Rmap data has a number of

errors due to the experimental conditions and system limitations. In an optical mapping experiment, it is very hard to get uniform fluorescent staining. This leads to an erroneous estimation of fragment sizes. Also, restriction enzymes often fail to digest all occurrences of their recognition sequence across the DNA molecule. This creates missing restriction sites. Due to DNA's fragile nature, additional brakes can masquerade as false restriction sites. The limitations of the imaging component of the optical mapping system and the propensity for the DNA to ball up at the ends introduces more error sizing error for smaller fragments. Because of all these erroneous experimental conditions, optical mapping data generated through optical mapping experiment has insertions and deletions of cut sites along with fragment size substitution errors.

Nonetheless, in order to use optical mapping data for further analysis (scaffolding, variant calling, etc.) the Rmaps have to be assembled into a genome wide optical map. This is because the single molecule maps need oversampling to improve the accuracy in the presence of the aforementioned errors, and because single molecule maps only span on the order of 500 Kbp [19]. The first step of this assembly process involves aligning one Rmap to another. In order to accomplish the challenge of dealing with missing fragment sizes has to be overcome, one that is analogous to dealing with insertion and deletions in the alignment of long reads—in fact, it is arguably harder since the data is integral (where every element is a substitution relative to its aligned element, unlike the 4 symbol categorical alphabet of DNA). At the present moment, the only non-proprietary algorithmic method for pairwise aligning Rmap reads is dynamic programming based methods the method of Valouev et al [19]. SOMA [14] also accommodates the full range of experimental errors present in the data. These methods are inherently computationally intensive. However, if the error rate of the Rmap methods

could be significantly improved, then subsequent, non-dynamic-programming based methods that are orders of magnitude faster (such as Twin [13]) could be used for alignment. This would inadvertently have the effect of improving the time required to assemble Rmap data into a genome-wide optical map.

### 1.1. Our contribution

Even though there exist methods for correcting short-read data, and Pacific Biosciences (PacBio) data that has a 15% insertion and deletion error rate, they cannot be used to correct Rmap data because each Rmap is a sequence of numerical values rather than a string of biological characters. We present cOMet, a method that error corrects Rmap data. To the best of our knowledge, this is the first such method and thus, we cannot compare against any other existing tool. Our experimental results are demonstrated on simulated Rmaps from the *E.coli* K-12 reference genome showing high percentage of error corrections.

### 1.2. Related work

Many optical mapping tools exist and deserve mentioning, including AGORA [11], SOMA [14], and Twin [13]. Twin [13] is an index-based method for aligning contigs to an optical map. Due to its use of an index data structure it is capable of aligning *in silico* digested contigs orders of magnitude faster than competing methods based on dynamic programming algorithms, however, it is not suitable for aligning raw optical mapping data. SOMA [14] is a scaffolding method that uses a consensus optical map and is specifically designed for short-read assemblies. SOMA includes an alignment method for scaffolding. It is a $O(n^2m^2)$-time dynamic programming algorithm. Gentig [2], and software developed by Valouev et al. [19]

also use dynamic programming to address the closely related task of finding alignments between optical maps. Gentig is not available for download. BACop [26] also uses a dynamic programming algorithm and corresponding scoring scheme that gives more weight to contigs with higher fragment density. Antoniotti et al. [3] consider the unique problem of validating an optical map by using assembled contigs. This method assumes the contigs are error-free. Optical mapping data was produced for Assemblathon 2 [5].

Although there has been a plethora of work on error correction of short read data [21], these methods are not appropriate for optical mapping data because they do not handle insertions and deletions appropriately (n.b. In the event of a mismatched site, the span of DNA between matched sites remains the same.) The error profile of PacBio reads is more similar to that of optical mapping data, i.e., the pervasiveness of insertions and deletions, and hence, the methods for error correcting PacBio reads, such as LSC [4], PBcR [10] and Coral [16], are more relevant to correcting Rmaps. LSC uses short read information to correct PacBio data. LSC applies a well-proven technique of homopolymer compression transformation on long and short reads in order to increase the sensitivity of short read-long read alignment. LSC also filters out compressed short reads of poor quality before aligning long reads to short reads using Novoalign [1]. Finally, the consensus information from the aligned short reads is used to fix the errors of PacBio reads. PBcR computes all-versus-all alignment between quality accuracy short-reads and PacBio long-reads sharing seed sequences of 14 bp (by default). In the next step, PBcR tiles overlapping short-reads along each long-read sequences to generate the multiple-alignment of short-read sequences. Lastly, PBcR employs the AMOS consensus module to create a new consensus sequence for each long-read sequence from multiple-alignment generated in the previous $k$-mer in common

with one specific read which is referred to as a base read. For a next step, Coral computes the multiple alignments between the $k$-mer neighbourhood of the base read to generate a consensus sequence. Coral then uses these consensus sequences to correct errors in the reads. Although these methods are more relevant than short read error correction methods, the data in optical mapping is numerical which adds an increased level of complexity. PacBio error correction methods are not suitable for Rmap error correction for this reason.

CHAPTER 2

# Background

## 2.1. Definitions and Notation

Throughout we consider a string $\mathsf{X} = \mathsf{X}[1..n] = \mathsf{X}[1]\mathsf{X}[2]\ldots\mathsf{X}[n]$ of $|\mathsf{X}| = n$ symbols drawn from the alphabet $[0..\sigma - 1]$. For $i = 1, \ldots, n$ we write $\mathsf{X}[i..n]$ to denote the *suffix* of $\mathsf{X}$ of length $n-i+1$, that is $\mathsf{X}[i..n] = \mathsf{X}[i]\mathsf{X}[i+1]\ldots\mathsf{X}[n]$. Similarly, we write $\mathsf{X}[1..i]$ to denote the *prefix* of $\mathsf{X}$ of length $i$. $\mathsf{X}[i..j]$ is the *substring* $\mathsf{X}[i]\mathsf{X}[i+1]\ldots\mathsf{X}[j]$ of $\mathsf{X}$ that starts at position $i$ and ends at $j$.

$k$-mer: All possible substrings of a string $\mathsf{X}[1..n]$ of length `k` are called as $k$-mers of $\mathsf{X}$. There are $n - k + 1$ $k$-mers possible for X.

Related Rmaps: If two Rmaps have at least one $k$-mer common in between them, then we refer to them as related Rmaps.

## 2.2. Optical Mapping

From a very basic viewpoint, optical mapping can be seen as a process that takes in two strings: a genome $\mathsf{A}[1, n]$ and a restriction sequence $\mathsf{B}[1, b]$, and produces an array (string) of integers $\mathsf{R}[1, m]$. The array $\mathsf{R}$ is an Rmap corresponding to $\mathsf{A}$. We note that millions of Rmaps are produced for a single genome since optical mapping is performed on many cells of the organism (not a single cell) and for each cell there are thousands of Rmaps. This is analogous to next generation shotgun sequencing. The Rmaps can be assembled to produce a genome wide optical map $\mathsf{M}$, which we define as follows: $\mathsf{M}[1, m]$ where $\mathsf{M}[i] = j$ if and only if $\mathsf{A}[j..j + b] = \mathsf{B}$ is the $i$th occurrence of $\mathsf{B}$ in $\mathsf{A}$. For example, if we let $\mathsf{B} = act$ and $\mathsf{A} = atacttactggactactaaact$ then we would have $\mathsf{M} = 3, 7, 12, 15, 20$. Hence, each Rmap is

an array of distances—corresponding to the fragment sizes—between occurrences of $\mathsf{B}$ in $\mathsf{A}$ (equivalently differences between adjacent values in $\mathsf{M}$). More formally, we define a Rmap $\mathsf{R}[1, m]$ where $\mathsf{R}[i] = (\mathsf{M}[i] - \mathsf{M}[i-1])$, with $\mathsf{R}[1] = \mathsf{M}[1] - 1$. Continuing with the example above, we have $\mathsf{R} = 2, 4, 5, 3, 5$.

There are three types of errors that can occur in optical mapping: (1) missing cut sites which are caused by an enzyme not cleaving at a specific site, (2) additional cut sites which can occur due to random DNA breakage and (3) inaccuracy in the fragment size due to the inability of the system to accurately estimate the fragment size. Continuing again with the example above, a more representative example Rmap would include these errors, such as $\mathsf{R}' = 7, 6, 3, 4$. There is a 15% probability that a cut site is missing, i.e., error type (1) occurs in Rmap. For every 400 KB of Rmap, there is about 1 random break appearing as a cut site, i.e. error type (2). The inaccuracy of the fragment sizes, i.e., error type (3), follows a normal distribution with standard deviation ($\sigma$) which depends on actual length of the fragment. For example, if $L$ is an actual length of a fragment, then length measured by the optical mapping system shows a normal distribution across $L$ with standard deviation $\sigma$ given by $\sigma^2 = f(L)$ ([20]). One significant challenge in aligning one Rmap to another is overcoming missing cut sites. Continuing on from our example above, we need to be able to at minimum align the error containing $\mathsf{R}' = 7, 6, 3, 4.$ to our unobservable perfect map $\mathsf{R} = 2, 4, 5, 3, 5$. In practice, a pair of Rmaps will have twice the pairwise error rate since each will deviate from the genomic map by the above parameters.

CHAPTER 3

# Methods

Given a set of Rmaps $\mathsf{R} = \{\mathsf{R}_1, .., \mathsf{R}_n\}$, where $n$ is a number of Rmaps, the error correction problem aims to detect and correct all three types of errors in each of them. Our method consists of the following steps: quantization of fragment sizes, construction of an Rmap index for storing related Rmaps, construction of multiple alignments between Rmaps, and error correction of Rmaps.

## 3.1. Quantization of Fragment Sizes

Before running error correction, Rmaps need preprocessing to remove the first and the last fragment from each of the Rmaps. This is done because these fragments have one of their edges sheared by artifacts of the DNA prep process preceding the optical mapping process and not restriction enzymes. Thus, they can misguide alignment between two Rmaps during the error correction process. Rmaps need to contain a minimum number of fragments in order to have enough information contained in the general pattern of fragment lengths to overcome the inherent errors. Thus, the next step in this preprocessing stage is to remove very short Rmaps, i.e., ones that have less than ten fragments. These are commonly removed before assembling or analyzing this data further [5].

Lastly, in the preprocessing stage, we quantize the data to account for sizing errors. As previously discussed in Subsection 2.2 each fragment size is subject to sizing error that skews the size by a random amount. The sizing error follows a normal distribution with mean of $\mu$ and standard deviation of $\sigma$. Therefore, we use nonlinear quantization to address this problem by creating bins of variable sizes, where the upper bound of bin $m$ is given by

$U(s) = U(s-1) + 2\delta_s$, where $\delta_s = w * \sigma$ and $w$ is a multiplier for how wide bins should be in terms of standard deviation. $\sigma$ is calculated using equation (1) for a fragment of length $U(s-1)$. We note that $U(0) = 0.5$ kbp. For example, the second bin would range from 0.5 kbp to $0.5 + 2\delta_1$. All fragment sizes in the range of $U(s-1)$ to $U(s)$ are replaced with the value $s$. This is done for each fragment size and each Rmap. Hence, using this model we can see that the bin size increases with the fragment length since the standard deviation is dependent on the fragment length. We note that a copy of original, un-quantized Rmap is stored for later analysis.

### 3.2. Construction of Related Rmap Index

Rmaps originating from the same segment of the genome will likely have a number of quantized fragment $k$-mers in common. In order to avoid computing the edit distance between all pairs of Rmaps, we use the number of these common $k$-mers to discriminate between pairs of Rmaps which are related and those that are not. When a pair is classified as related, we assume they originate from the same segment of the genome. In order to correct errors from each Rmap $\mathsf{R}_i$ in $\mathsf{R}$, we first find a list of related Rmaps of $\mathsf{R}_i$. In the next step, we generate an alignment between $\mathsf{R}_i$ and each Rmap in the list related Rmaps of $\mathsf{R}_i$. These alignments are stored with respect to $\mathsf{R}_i$ which are then used to create a consensus map to correct errors from $R_i$.

To accomplish this efficiently, we first extract all unique $k$-mers occurring in quantized Rmaps and their reversed Rmaps. We construct a hash table that stores each unique $k$-mer as a key with the list of Rmaps containing occurrence of that $k$-mer as the value. We call this hash table *the k-mer index*. Next, we generate the *related Rmap index* using the $k$-mer index. For each pair of Rmaps that have at least one $k$-mer in common, we store the count

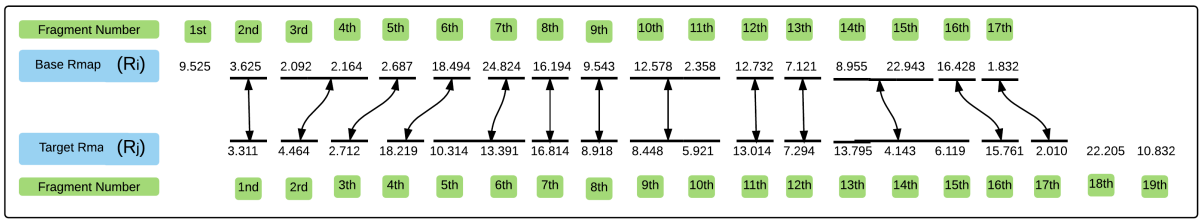


FIGURE 3.1. An example illustrating the alignment between the base Rmap $R_i$ and the target Rmap $R_j$ generated using Valouev et al. method.

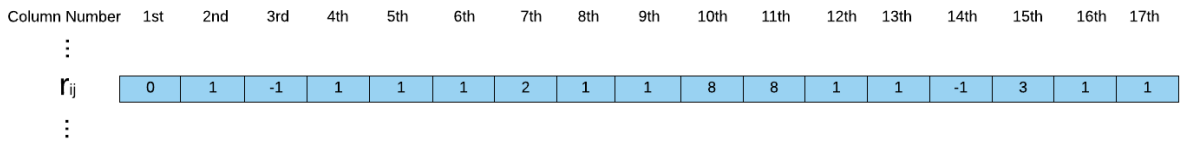


FIGURE 3.2. One of the rows from the multiple alignment grid of the base Rmap $R_i$ used for storing the alignment between $R_i$ and $R_j$ generated using Valouev et al. method.

of the $k$-mers that the Rmaps have in common. We consider Rmaps $R_j, R_k, ..., R_l$ to be the related Rmaps of Rmap $R_i$ having $m_j, m_k, ..., m_l$ $k$-mers in common respectively. The related Rmap index is an array where each entry is a hash table. The i$^{\text{th}}$ entry in the related Rmap index represents the collection of related Rmaps of $R_i$. Each hash table stores the number of $k$-mers that are common between $R_i$ and its related Rmaps. So the keys in this hash table are the related Rmaps of $R_i$ i.e. $R_j, R_k, ..., R_l$ and the values are the count of the $k$-mers in common between $R_i$ and its related Rmaps, i.e. $m_j, m_k, ..., m_l$. In order to minimize the memory requirement, all common $k$-mers between a pair of Rmaps are recorded only once in the related Rmap index. For example, if a $k$-mer $k_x$ is shared by $\mathsf{R}_i$ and $\mathsf{R}_j$ and $i < j$ then the related Rmap index will have an entry for $\mathsf{R}_i$ specifying that $k_x$ is shared with $\mathsf{R}_j$; however, the reverse is not true.

### 3.3. Rmap Alignment and Error Correction

For each Rmap $\mathsf{R}_i$ in $\mathsf{R}$, we use the related Rmap index to find all Rmaps that have $\mathtt{m}$ or more $k$-mers in common with $\mathsf{R}_i$. We denote this set of Rmaps as $\mathsf{R}^i$. We refer to $\mathsf{R}_i$ as the base Rmap and every Ramp in $\mathsf{R}^i$ as the target Rmap. Next, we use the method of Valouev et al. [19] to find all pairwise alignments between $\mathsf{R}_i$ and each Rmap in $\mathsf{R}^i$. The Valouev et al. method outputs an alignment only when the optimal alignment generated satisfies the S-score and T-score threshold [19]. If the Valouev et al. method produces an alignment between $\mathsf{R}_i$ and any Rmap in $\mathsf{R}^i$, then it is stored in a *multiple alignment grid*. The multiple alignment grid is a two dimensional array of integer values. The number of rows of the multiple alignment grid is equal to the number of Rmaps in $R^i$ for which the Valouev et al. method produced an alignment and the number of columns is equal to the number fragments in $\mathsf{R}_i$. Basically, a row in the multiple alignment grid represents an alignment between $\mathsf{R}_i$ and one of the Rmaps in $\mathsf{R}^i$, which we denote as $\mathsf{R}_j$.

Figure 3.1 shows an example of an alignment produced by the Valouev et al. method when we align the base Rmap $R_i$ and the target Rmap $R_j$. We store this alignment in one of the rows of the multiple alignment grid and we identify that row as $r_{ij}$. Each cell in $r_{ij}$ corresponds to a fragment in a $R_i$. In our example, figure 3.2 shows $r_{ij}$ used to store the alignment between $R_i$ and $R_j$. We categorize the alignment generated by the Valouev et al. method between $R_i$ and $R_j$ as one of four types: (1) one fragment of $\mathsf{R}_i$ can align with one fragment of $\mathsf{R}_j$, (2) one fragment of $R_i$ can align with $y$ fragments of $R_j$ where $y \geq 2$, (3) $x$ fragments of $R_i$ can align with $y$ fragments of $R_j$ where $x = y$ and $x \geq 2, y \geq 2$, and (4) $x$ fragments of $R_i$ can align with $y$ fragments of $R_j$ where $x \neq y$, $x \geq 2$ and $y \geq 1$. In order to minimize the memory requirement, every alignment is stored with respect to the

base Rmap using integer values. In the example shown in figure 3.1, the second fragment of $R_i$ (value 3.625) aligns with the first fragment of $R_j$ (value 3.311); This type of alignment is of the type 1 alignment produced by the Valouev et al. method. In order to store such alignments, we store a 1 in second cell of $r_{ij}$. This implies that the second fragment of $R_i$ aligns with a single fragment of $R_j$. We also store the alignment starting point for every target Rmap in $R^i$ for which the Valouev et al. method generates an alignment. So from our example, we store the alignment start point for $R_j$ which is 1 because the alignment between $R_i$ and $R_j$ starts from the first fragment for $R_j$. The seventh fragment of $R_i$ (value 24.824) aligns with two fragments, the fifth and sixth fragments of $R_j$ (values 10.314 and 13.391). This alignment is of the type 2 alignments generated by the Valouev et al. method. In this alignment, $y = 2$. In order to represent such alignments in $r_{ij}$, we store a 2 in seventh cell of $r_{ij}$, which implies that seventh fragment of $R_i$ aligns with 2 fragment from $R_j$. It also implies that there are $y - 1$, i.e. one missing cut site in $R_i$ as compared to $R_j$, i.e. deletion error. The two fragments of $R_i$, tenth and eleventh (values 12.578 and 2.358) align with the two fragments of $R_j$, ninth and tenth (values 8.448 and 5.921). So $x = 2$ and $y = 2$. Such type of alignments are of type 3 alignments generated by the Valouev et al. method. In order to represent such alignments in $r_{ij}$, we store an 8 in $x$ cells of $r_{ij}$. Hence, we store 8 in the tenth and eleventh cells of $r_{ij}$, which represent that the tenth and eleventh fragment of $R_i$ align with the two fragments of $R_j$. By default the Valouev et al. method can output a maximum of seven fragment of one Rmap aligning to some number of fragments of the other Rmap, hence 8 is the minimum number available to use which can distinguish type 3 alignments form others. The third and fourth (values 2.092 and 2.164) fragment of $R_i$ align with the second fragment $R_j$ (value 4.464). Such alignments are of type 4. For this alignment $x = 2$

and $y = 1$. In order to record such alignments in $r_{ij}$, we store a -1 in $r_{ij}$ for $x - 1$ cells and $y$ in the following cell. Hence, we store a -1 in the third cell of $r_{ij}$ and a 1 in forth cell of $r_{ij}$. In this alignment $x > y$ hence we can say that there are $x - y$ extra cut sites in $R_i$ (i.e insertion error as compared $R_j$). Similarly, when the fourteenth and fifteenth fragment of $R_i$ (values 8.955 and 22.943) align with the thirteenth, fourteenth and fifteenth fragment of $R_j$ (values 13.795, 4.143 and 6.119), then we store -1 at fourteenth cell and 3 at fifteenth cell in $r_{ij}$ which implies that the two fragments, fourteenth and fifteenth of $R_i$, align with the three fragments of $R_j$. In this alignment, because $x = 2$, $y = 3$ and $x < y$, we can infer that there are $y - x = 1$ one missing cut sites in $R_i$ (i.e. there are deletion errors as compared to $R_j$). The first fragment of $R_i$ does not align with any fragment of $R_j$. In order to represent such alignment in $r_{ij}$, we store a 0 in the first cell of $r_{ij}$. (n.b. Careful readers may note that this encoding cannot distinguish runs of four or more fragments matched from consecutive sets of two or more, however we favor this approach over more complex and precise ones because such patterns are rare.)

The multiple alignment grid helps in correcting errors from the base Ramp. For every fragment in the base Rmap, we first generate a consensus using the multiple alignment grid which we call a *consensus map*. In order to generate a consensus map, we iterate through each column of the multiple alignment grid. Every column of the multiple alignment grid corresponds to a fragment in the base Rmap. For every column, the frequency of occurrence of the values is counted and the value which occurs most frequently is consider to be the consensus for that fragment. In order to deem that alignment as the final consensus, we use a threshold value $d$. A threshold is important in cases where the base Ramp has a small number of target Rmaps aligning to it. In such cases, it is more probable that we will make

a mistake and distort a correct value rather than correcting an error. In the case of a tie for the majority alignment, we chose one of the alignments randomly, expecting that we will make a correct choice half the time; ties happens very rarely.

### 3.4. Complexity

We define $\ell$ to be the length of the longest Rmap in $\mathsf{R}$. Quantization of the Rmaps can be accomplished in $O(\ell n)$-time since there are $n$ Rmaps and each has length at most $\ell$. In order to construct the hash table containing all $k$-mers and their lists of associated Rmaps, each $k$-mer has to be indexed twice—once for the $k$-mer in the forward direction and a second time for the $k$-mer in the reverse direction. Hence, the construction of the $k$-mer index can be accomplished in $O(\ell n)$-time. Next, to generate the related Rmap index, we scan through all $\ell n$ $k$-mers and record related Rmaps. In the worst case, a $k$-mer can be common to all Rmaps, making all of the input Rmaps appear related to each other, thus this is $n^2$ related Rmaps. The related Rmap index is an array of hash tables so each related Rmap pair can be stored in constant time. Hence, the time required to create the related Rmap index is $O(\ell n^2)$. Valouev et al. method takes $O(\ell^2)$ to generate the alignment between two Rmaps. In the worst case, cOMet need to find $n^2$ alignments. It can be accomplished in $O(\ell^2 n^2)$-time. The alignment generated using Valouev et al. method can be stored in the multiple alignment grid in constant time. It may take $O(\ell n^2)$ to generate the consensus maps for $n$ Rmaps. Thus, the runtime of cOMet is $O(\ell^2 n^2)$

CHAPTER 4

# Results

The performance of cOMet was evaluated on simulated optical mapping data for *E.coli* genome. All the experiments were performed on Intel x86-64 workstations with sufficient RAM to avoid paging, running 64-bit Linux. The performance of cOMet was compared for the different values of `k` ($k$-mer value), `m` (the number of $k$-mers needed to be conserved between two Rmaps), `d` (the minimum number of Rmaps having agreed to form consensus), and the number of copies of *E. coli* genome. To evaluate the performance of cOMet, a variety of statistics are reported including the number of simulated Rmaps in the input file. The number of deletion and insertion errors introduced while simulating the Ramps are also reported. The results contain the percentage of the total number of the deletion and insertion errors corrected by cOMet. In order to evaluate the accuracy of the corrections, the percentage of accurately corrected errors are reported. This percentage is refereed to as the percentage of true positive corrections. The true positive percentage of corrected

Table 4.1. An illustration of the change in the performance of cOMet with simulated *E.coli* genome in response to varying values of `k` with 200 copies of the *E.coli* (str. K-12 substr. MG1655) reference genome, `d` equal to 2, `m` equal to 2.

| `k` | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Number of Rmaps in input file | 995 | | | | | | |
| Total number of deletion errors | 4,515 | | | | | | |
| % of corrected deletions errors | 83.72% | 82.66% | 73.47% | 57.56% | 40.11% | 23.41% | 11.52% |
| True positive % of corrected deletions errors | 83.78% | 84.03% | 82.97% | 81.26% | 78.52% | 77.01% | 73.65% |
| Total number of insertion errors | 1,924 | | | | | | |
| % of corrected insertions errors | 79.16% | 78.17% | 79.00% | 73.28% | 59.30% | 39.92% | 20.43% |
| True positive % of corrected insertions errors | 84.04% | 83.44% | 79.08% | 71.49% | 64.24% | 58.59% | 52.67% |
| Number of Rmaps aligned before running Comet | 990 | | | | | | |
| Avg quality score of Rmaps before running Comet | 69.59 | | | | | | |
| % of Rmaps aligned after running Comet | 98.89% | 97.88% | 97.37% | 96.77% | 96.36% | 97.78% | 99.39% |
| Avg quality score of Rmaps after running Comet | 70.66 | 70.59 | 68.31 | 65.66 | 65.29 | 66.11 | 67.63 |
| Number of Rmaps showing improved quality score | 625 | 610 | 506 | 347 | 247 | 131 | 72 |
| Run-time in CPU seconds | 2,043.71 | 950.97 | 337.23 | 153.98 | 72.79 | 33.9 | 15.66 |
| Peak memory usage in MB | 25.18 | 13.67 | 10.56 | 10.7 | 10.71 | 11.31 | 11.51 |

deletion and insertion errors are recorded in the results. The improvement of the quality scores of the corrected Rmaps was assessed. This was performed by first generating the error-free maps of *E.coli* reference genome, and then the errors were introduce to form the simulated Rmaps. These resulting Rmaps were aligned to the corresponding error-free maps using Valouev et al. [19]. For each alignment produced by Valouev et al., it also generated an alignment score. This score is referred to as a *quality score* of the Rmap before error correction. See Valouev et al. [19] for a complete description of the scoring function. Next, cOMet was run on erroneous Rmaps to correct errors. Each corrected Rmap was then aligned to the corresponding error-free map to obtain an improved alignment score i.e. the quality score of Rmap after error correction. The average of the quality scores along with the number of Rmaps aligned before and after running cOMet are reported in the results. In order to assess the improvement in the quality of an individual Rmap, the number of Rmaps having improved quality score after running cOMet are reported in the results. The default scoring parameters of the Valouev et al. scoring function were modified based on the error model described in section 2.2. In the results, we also reported the peak memory usage and the runtime for each run. Peak memory was measured as the maximum resident set size as reported by the operating system with sufficient RAM to avoid paging. Runtime is the user process time, also reported by the operating system.

### 4.1. Performance on Simulated E.coli Rmap Data

The *E. coli* reference genome is one of the smallest genomes and likely to contain the fewest errors and thus, is the one used for verifying the improvement in the quality scores. We simulated Rmap data using the reference genome for *E. coli* (str. K-12 substr. MG1655). The Rmaps were simulated first by locating ten uniformly distributed random loci within each

copy of the genome which are taken to be breakpoints. These breakpoints form the ends of single molecule that would undergo *in silico* digestion. Molecules smaller than 250 Kbp were discarded. The cleavage sites for the XhoI enzyme were then identified within each of these simulated molecules. The map generated for each simulated molecules due to the cleavage sites is refereed to as a error-free map. The error-free maps are later used for validating the output of our method. Next, the deletion, insertion and sizing errors were incorporated in the error-free maps to simulate Rmaps. The deletion errors, i.e., missing cut sites were simulated by removing one cut site randomly for every 6.66 cut sites. The insertion errors, i.e., adding an extra cut site, were simulated by randomly adding an extra cut site every 400 kbp. Finally, the sizing errors were added to each fragment size by first computing the standard deviation using equation 1 and then sampling from an appropriately parameterized Gaussian distribution. This method of simulating Rmaps was based on the error model described in section 2.2. Our earlier experiments show that $w = 4$ gives quantization accuracy of more than 85%, meaning the two fragments originating from the same part of genome will be quantized in the same bins. We use $w = 4$ for all other experiments.

cOMet was ran for the different values of `k`, `m`, and `d`. The best results were obtained when `k` was equal to three, `m` was equal to two, and `d` was equal to two. These values are used as the default parameters for our software. Table 4.1 illustrates the performance of cOMet for varying values of `k` when all the other parameters are kept constant. To conduct this experiment, the value of `k` was varied from two to eight. The minimum value of `k` was chosen to be two because `k` equal to one is very small and generates too many related Rmaps for each base Rmap that are not incepted from the same location in the genome . This depreciates the purpose of filtering Rmaps originating from the same part of genome using

TABLE 4.2. An illustration of the change in the performance of cOMET with simulated *E.coli* genome in response to varying values of `m` with 200 copies of the *E.coli* (str. K-12 substr. MG1655) reference genome, `k` equal to 3, `d` equal to 2.

| `m` | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Number of Rmaps in input file | 995 | | | | | | |
| Total number of deletion errors | 4,515 | | | | | | |
| % of corrected deletions errors | 82.66% | 78.80% | 78.80% | 71.47% | 71.47% | 62.97% | 62.97% |
| True positive % of corrected deletions errors | 84.03% | 83.02% | 83.02% | 82.34% | 82.34% | 81.46% | 81.46% |
| Total number of insertion errors | 1,924 | | | | | | |
| % of corrected insertions errors | 78.17% | 77.70% | 77.70% | 76.92% | 76.92% | 71.88% | 71.88% |
| True positive % of corrected insertions errors | 83.44% | 81.74% | 81.74% | 78.51% | 78.51% | 75.42% | 75.42% |
| Number of Rmaps aligned before running Comet | 990 | | | | | | |
| Avg quality score of Rmaps before running Comet | 69.59 | | | | | | |
| % of Rmaps aligned after running Comet | 97.88% | 98.08% | 98.08% | 96.26% | 96.26% | 97.07% | 97.07% |
| Avg quality score of Rmaps after running Comet | 70.59 | 69.9 | 69.9 | 69.15 | 69.15 | 68.14 | 68.14 |
| Number of Rmaps showing improved quality score | 610 | 560 | 559 | 507 | 505 | 409 | 409 |
| Run-time in CPU seconds | 950.97 | 525.89 | 523.17 | 321.35 | 319.69 | 207.65 | 208.66 |
| Peak memory usage in MB | 13.67 | 10.92 | 10.92 | 9.92 | 9.79 | 9.36 | 9.37 |

$k$-mer value. Similarly, `k` equal to or greater than nine generates very few related Rmaps for each base Rmap, thus, making these values insignificant to consider. The results indicate that cOMET corrects fewer errors and with less accuracy as the value of `k` increases. It is due to the fact that the optical mapping data has high insertion and deletion error rates (an extra cut site for every 400 Kbp of Rmap and 15% missing cut sites). Therefore, finding a $k$-mer common between a pair of Rmaps for a large value of `k` is more difficult than finding it for a small value of `k`. Statistically, when a value of `k` is large, cOMET can find fewer related Rmaps for each of the base Rmaps. Also, fewer $k$-mers could be generated if the value of `k` is large. Due to these reasons, cOMET finds fewer target Rmaps for each of the base Rmaps when value of `k` is large and thus, it corrects fewer errors with less accuracy. For `k` equal to two and three, cOMET approximately corrects the same number of errors. For `k` equal to two the run-time is 2,043 seconds and the peak memory usage is 25 MB. For `k` equal to three, the run-time is 951 seconds and the peak memory usage is 14 MB. For `k` equal to two, cOMET generate many alignments for dispensable target Rmaps. The run-time and the peak memory usage is approximately double for `k` equal to two compared to `k` equal to three

TABLE 4.3. An illustration of the change in the performance of cOMET with simulated *E.coli* genome in response to varying values of `d` with 200 copies of the *E.coli* (str. K-12 substr. MG1655) reference genome, `k` equal to 3, `m` equal to 2.

| `d` | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Number of Rmaps in input file | 995 | | | | | | |
| Total number of deletion errors | 4,515 | | | | | | |
| % of corrected deletions errors | 82.66% | 80.44% | 78.05% | 75.66% | 73.75% | 71.72% | 69.50% |
| True positive % of corrected deletions errors | 84.03% | 84.28% | 84.48% | 84.54% | 84.47% | 84.59% | 84.61% |
| Total number of insertion errors | 1,924 | | | | | | |
| % of corrected insertions errors | 78.17% | 75.52% | 73.54% | 71.78% | 70.32% | 68.50% | 66.94% |
| True positive % of corrected insertions errors | 83.44% | 84.86% | 85.58% | 86.02% | 86.10% | 86.12% | 86.49% |
| Number of Rmaps aligned before running Comet | 990 | | | | | | |
| Avg quality score of Rmaps before running Comet | 69.59 | | | | | | |
| % of Rmaps aligned after running Comet | 97.88% | 98.69% | 98.69% | 98.48% | 98.59% | 98.69% | 98.79% |
| Avg quality score of Rmaps after running Comet | 70.59 | 70.60 | 70.63 | 70.77 | 70.74 | 70.77 | 70.89 |
| Number of Rmaps showing improved quality score | 610 | 598 | 597 | 588 | 578 | 570 | 565 |
| Run-time in CPU seconds | 950.97 | 953.27 | 949.67 | 944.69 | 951.8 | 958.58 | 948.37 |
| Peak memory usage in MB | 13.67 | 13.82 | 13.67 | 13.7 | 13.82 | 13.69 | 13.81 |

while the number of error correction is approximately the same. Hence, we can conclude that `k` equal to three produces the best results. For `k` equal to three, cOMET corrected 83% deletion errors out of which 84% were accurate corrections. Similarly, cOMET corrected 78% of insertion errors out of which 83% were accurate corrections. Also, when `k` was equal to three, cOMET improved the average quality score of the Rmaps from 69.59 to 70.59. We found the improvement in the quality score for 610 Rmaps out of 990 Rmaps.

Table 4.2 shows the performance of cOMET for varying values of `m` when other parameters such as the number of copies of the genome (equal to 200), `k` (equal to three), and `d` (equal to two) are kept constant. For this experiment, the value of `m` was varied from two to eight to observe the behavior of cOMET. Here, we discuss only the minimum and maximum extreme values to analyze the performance. The experiment results show that as the value of `m` increased, the error correction percentage decreases for both deletion and insertion errors. Larger value of `m` entail that more $k$-mers need to be in common between related Rmaps to be able to consider them as a target Rmap for give base Rmap. Statistically, for a larger value of `m`, we can find fewer such related Ramps for given base Rmaps. Hence affecting the

total number of corrections. For `m` equal to eight, cOMet can find fewer target Rmaps for given base Rmap as compared to `m` equal to two. Hence, cOMet can correct fewer errors for `m` equal to eight as compared to `m` equal to two. For `m` equal to two, cOMet is able to correct approximately 83% of deletion and 78% of insertion errors with the accuracy of 84% and 83% respectively. While for `m` equal to eight, cOMet is able to correct only 63% of deletion and 72% of insertion errors with the accuracy of 81% and 75% respectively. Generating an alignment between Rmaps using the Valouev et al. method is the most time consuming stage of cOMet. Hence the larger number of target Rmaps there are for a given base Rmap, the greater run-time required by cOMet will be. cOMet took 950 CPU seconds to complete execution for `m` equal to two, while 209 CPU seconds for `m` equal to eight. Similarly, storing all unique related pairs consumes most of the memory in cOMet. Hence the peak memory usage for `m` equal to two is 14 MB while 9 MB for `m` equal to eight.

Table 4.3 illustrates the effect of different values of `d` when all other parameters such as number of copies of the genome (equal to 200), `k` (equal to three), and `m` (equal to two) are kept constant. In this experiment also, the value of `d` was varied from two to eight so as to observe the behavior of cOMet. We chose the minimum value of `d` to be two because it is the minimum value required to form a consensus. We ran cOMet for values of `d` greater than two and found a pattern in the behavior. `d` equal to eight seemed to be a reasonable value to stop. Results show that as the value of `d` increases, cOMet is able to correct fewer total deletion and insertion errors but with approximately the same accuracy. These results show that even though the larger value of `d` promises more accurate correction, it significantly impacts the overall number of corrections. We analyze the trend in the performance of cOMet by discussing the minimum and maximum value of `d`. For `d` equal to two, cOMet

TABLE 4.4. An illustration of the change in the performance of cOMet with default parameters in response to an increase in the number of copies of the genome. All data was simulated using the *E.coli* (str. K-12 substr. MG1655) reference genome.

| Copies | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| Number of Rmaps in input file | 498 | 995 | 1,501 | 1,978 | 2,511 |
| Total number of deletion errors | 2,256 | 4,515 | 6,764 | 9,042 | 11,322 |
| % of corrected deletions errors | 74.25% | 82.66% | 82.73% | 83.59% | 84.76% |
| True positive % of corrected deletions errors | 84.24% | 84.03% | 83.58% | 83.75% | 84.14% |
| Total number of insertion errors | 956 | 1,924 | 2,850 | 3,828 | 4,768 |
| % of corrected insertions errors | 78.03% | 78.17% | 79.30% | 78.00% | 78.73% |
| True positive % of corrected insertions errors | 82.71% | 83.44% | 83.94% | 84.39% | 83.7% |
| Number of Rmaps aligned before running Comet | 498 | 990 | 1,497 | 1,974 | 2,504 |
| Avg quality score of Rmaps before running Comet | 68.9 | 69.59 | 68.85 | 69.96 | 68.84 |
| % of Rmaps aligned after running Comet | 95.78% | 97.88% | 97.86% | 98.23% | 98.64% |
| Avg quality score of Rmaps after running Comet | 68.59 | 70.59 | 70.5 | 71.74 | 70.93 |
| Number of Rmaps showing improved quality score | 272 | 610 | 981 | 1,319 | 1,763 |
| Run-time in CPU seconds | 239.94 | 950.97 | 2,149.22 | 3,935.36 | 6,018.46 |
| Peak memory usage in MB | 7.42 | 13.67 | 23.3 | 36.49 | 53.86 |

is able to correct 83% deletion and 78% insertion errors while for `d` equal to eight, cOMet is able to correct only 70% deletion and 67% insertion errors. Also, note that the runtime for each of these experiments is approximately the same because the total number of the target Rmaps aligned to the given base Rmap remains the same. Similarly, the peak memory usage for each of these experiments is approximately the same because the size of related Rmap index which require most of the memory remains the same. From table 4.3, we can conclude that setting `d` to two maximizes the correction percentage.

Table 4.4 shows the performance of cOMet in response to varying the number of copies of the simulated *E.coli* genome. As the number of copies increases, the number of Rmaps in the input file also increases. More copies of Rmaps also implies a greater number of target Rmaps for every base Rmap and thus gives rise to improved results. For 100 copies of the genome, cOMet corrected approximately 74% deletion and 78% insertion errors. These correction percentages were improved when the number of copies of the *E.coli* genome were increased. For 500 copies of E.coli genome, cOMet corrected approximately 85% deletion and 79% insertion errors. For 100 copies of the genome, the average quality score of Rmaps before

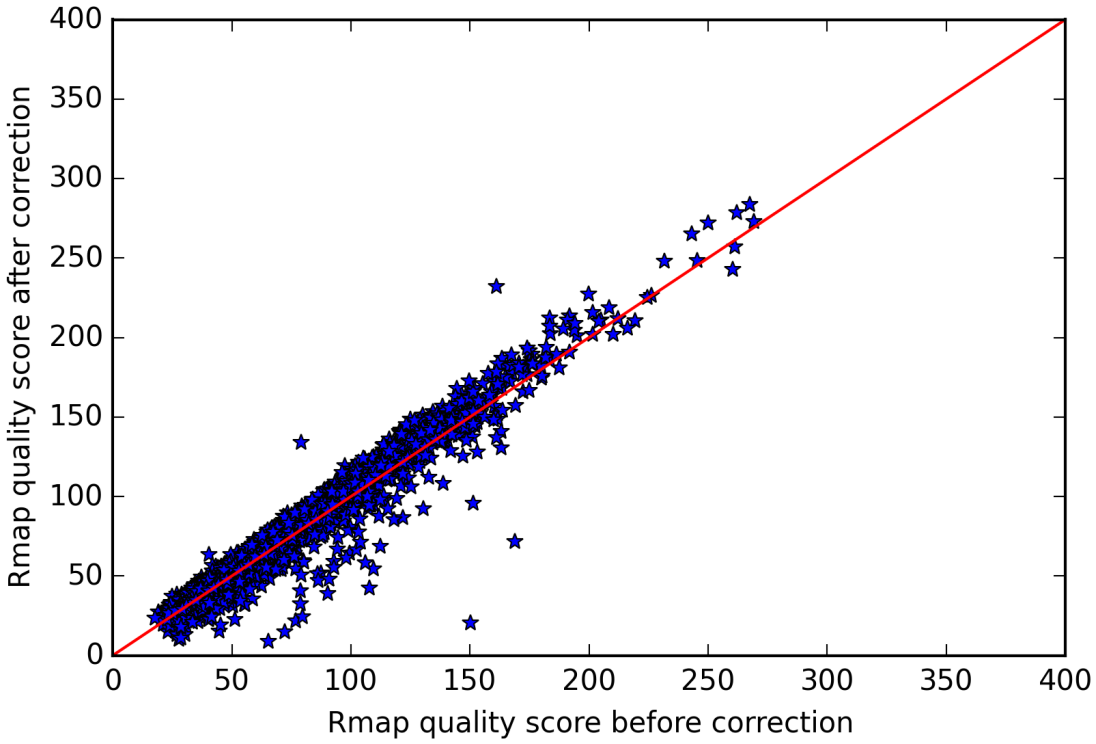


FIGURE 4.1. Plot of the quality scores of the Rmaps before versus after error correcting simulated *E.coli* genome using cOMET. Here we ran cOMET with default parameters and simulated Rmaps using 500 copies of the *E.coli* genome.

correction was 68.90 which later reduced to 68.59 after running cOMET. It implies that the cOMET was introducing the new errors instead of correcting them for some simulated Rmaps. This is because of the small number of copies of the reference genome used in simulating the Rmaps. The small number of copies implies lower coverage and hence affects the accuracy of correction for some simulated Rmaps. With an increased number of copies, the number of target Rmaps for each base Rmaps also increases. It results in increased number of alignments to be generated, thus, increasing the run time and memory usage. The results in table 4.4 shows that the cOMET finished execution in approximately 240 seconds for 100 copies, while it took 6018 seconds for 500 copies. Similarly, cOMET had peak memory usage of 54 MB for 500 copies but it took only 7 MB for 500 copies.

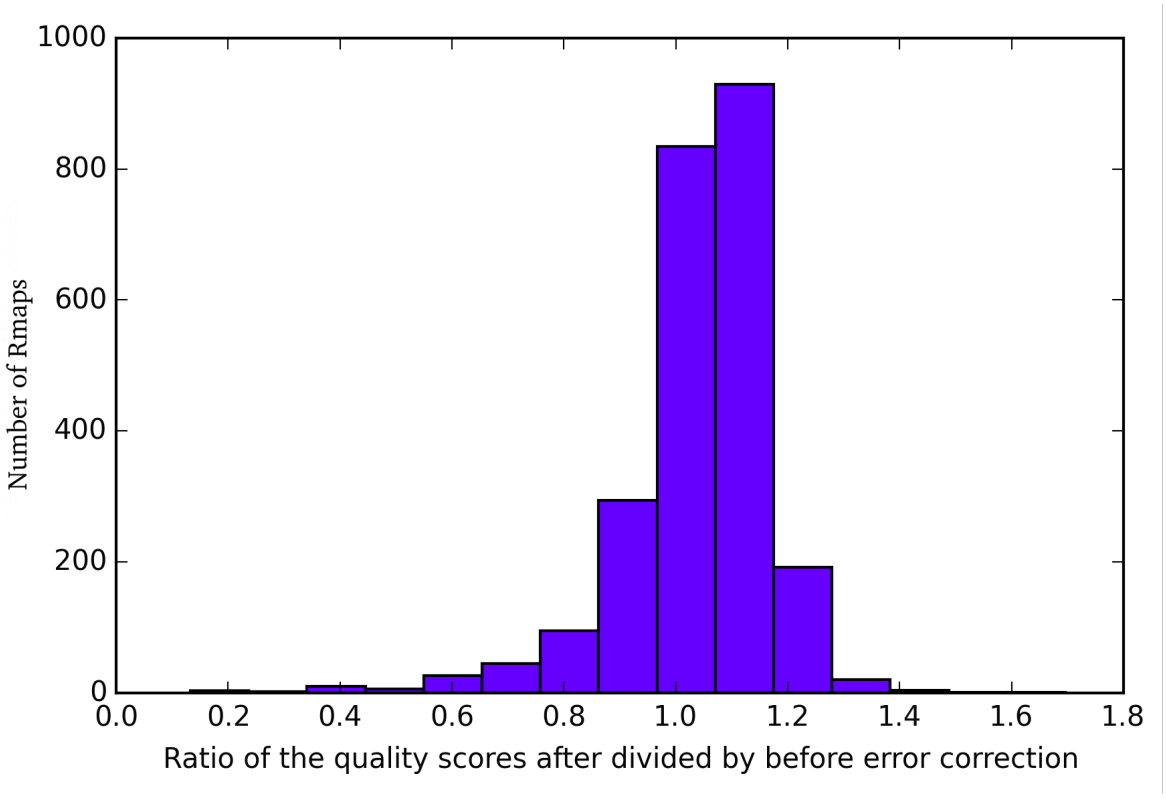


FIGURE 4.2. An histogram illustrating the ratio of the quality score of the simulated Rmap after error correction with cOMET and before error correction with cOMET. Here we ran cOMET with default parameters and simulated Rmaps using 500 copies of the *E.coli* genome.

In order to show the improvement in the quality scores, we ran cOMET on the Ramps simulated using 500 copies of the *E.coli* reference genome. We kept all other input parameters such as `k`, `m` and `d` at their default values (i.e. `k` equal to three, `m` equal to two and `d` equal to two). Results of this experiment are shown in table 4.4. It shows the improvement of approximately two points in the average quality score for simulated Rmaps after running cOMET. We found improved quality score for 1,763 Rmaps out of 2,504 Rmaps. Figure 4.1 compares the quality score for each Rmap before and after running cOMET. The alignment score produced by the Valouev et al. method is proportional to the length of the Rmap [19]. Figure 4.1 shows that cOMET performed better for longer Rmaps and improved the quality scores for most of them. Subsequently, the improvements in the quality score for each Rmaps

was analyzed by finding the ratio of the quality scores after running cOMet divide by the quality score before running it. Figure 4.2 demonstrates a histogram of such ratios for all Rmaps. From this histogram we infer that, cOMet improved the quality score by 0%-20% for most of the Rmaps. Our results show that cOMet improves the overall quality of optical mapping data by fixing a large number of errors with high accuracy.

CHAPTER 5

# Discussion and Conclusions

We present the first non-proprietary error correction method and demonstrate that it significantly improves the quality of optical mapping data. Our method utilizes redundant information present in the optical mapping data to correct all types of errors in it. In order to accomplish this efficiently, our method generates an index of Rmaps which are possibly originating from the same part of the genome. We also presented a unique and efficient data structure for storing the multiple alignments between the Rmaps which we refer to as the multiple alignment grid. We used the simulated optical mapping data generated using *E.coli* (str. K-12 substr. MG1655) reference genome to parameterize and evaluate the performance of our method.

During the process of error correction, cOMet may also introduces new errors. This primarily occurs when there are a large number of indel errors (i.e. insertions and deletions error) occurring throughout the given Rmap making it impractical to find the precise related Rmaps and thus, the precise target Rmaps for the given Rmap. Due to this, the consensus map generated for the given Rmaps in the final stage of error correction process is inaccurate. This leads to the introduction of new errors in the given Rmap instead of correcting them. We note that the Rmaps present towards the end of an input Rmap data file have lower correction percentage. This is attributed to the fact that all common $k$-mers between a pair of Rmaps are recorded only once in the related Rmap index as stated in section 3.2. Due to this, the Rmaps present towards the end of an input Rmap data file have fewer related Ramps leading to a lower error correction percentage.

Lastly, optical mapping is a relatively new technology and we expect it to improve over time, resulting in the smaller standard deviations in fragment sizes. A smaller sizing error could facilitate better quantizing accuracy (i.e. fragments from the same part of the genome being quantized to the same bin.) This would lead to more precise identification of related Rmaps, thus improving the accuracy, run-time and memory usage of cOMet. The number $k$-mers shared between a pair of Rmaps is a good indication that those Rmaps were originated from the same location in the genome. However, there is scope of improvement in this area. If a pair of Rmaps share non-overlapping $k$-mers, then it is a stronger indication that those Rmaps are originating from the same location in the genome, as compared to overlapping k-mers,. It remains an area to be explored in the future to prefer non-overlapping $k$-mers over overlapping $k$-mers when deciding the target Rmaps for given base Rmap.

# Bibliography


[1] Novoalign — novocraft. `http://www.novocraft.com/products/novoalign/`. (Visited on 04/27/2015).

[2] T. Anantharaman and B. Mishra. A probabilistic analysis of false positives in optical map alignment and validation. In *Proceedings of WABI*, pages 27–40, 2001.

[3] M. Antoniotti, T. Anantharaman, S. Paxia, and B. Mishra. Genomics via optical mapping iv: sequence validation via optical map matching. Technical report, New York University, 2001.

[4] K. F. Au, J. G. Underwood, L. Lee, and W. H. Wong. Improving pacbio long read accuracy by short read alignment. *PLoS One*, 7(10):e46679, 2012.

[5] K. Bradnam et al. Assemblathon 2: Evaluating *de novo* methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):1–31, 2013.

[6] S. Chamala et al. Assembly and validation of the genome of the nonmodel basal angiosperm *amborella*. *Science*, 342(6165):1516–1517, 2013.

[7] D. M. Church et al. Lineage-specific biology revealed by a finished genome assembly of the mouse. *PLoS Biology*, 7(5):e1000112+, 2009.

[8] Y. Dong et al. Sequencing and automated whole-genome optical mapping of the genome of a domestic goat. *Nature Biotechnology*, 31(2):136–141, 2013.

[9] J. T. Howard et al. *De novo* high-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience*, 3:11, 2014.

[10] S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.

[11] H. Lin, S. Goldstein, L. Mendelowitz, S. Zhou, J. Wetzel, D. Schwartz, and M. Pop. AGORA: Assembly Guided by Optical Restriction Alignment. *BMC Bioinformatics*, 12:189, 2012.

[12] L. Mendelowitz and M. Pop. Computational methods for optical mapping. *GigaScience*, 3, 2014.

[13] M. D. Muggli, S. J. Puglisi, and C. Boucher. Efficient indexed alignment of contigs to optical maps. In *Algorithms in Bioinformatics*, pages 68–81. Springer, 2014.

[14] N. Nagarajan, T. D. Read, and M. Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24(10):1229–1235, 2008.

[15] S. Reslewic et al. Whole-genome shotgun optical mapping of *Rhodospirillum Rubrum*. *Applied and Environmental Microbiology*, 71(9):5511–5522, 2005.

[16] L. Salmela and J. Schröder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, 2011.

[17] D. C. Schwartz, X. Li, L. I. Hernandez, S. P. Ramnarain, E. J. Huff, and Y.-K. Wang. Ordered restriction maps of saccharomyces cerevisiae chromosomes constructed by optical mapping. *Science*, 262(5130):110–114, 1993.

[18] B. Teague et al. High-resolution human genome structure by single-molecule analysis. *Proceedings of the National Academy of Sciences*, 107(24):10848–10853, 2010.

[19] A. Valouev, L. Li, Y.-C. Liu, D. C. Schwartz, Y. Yang, Y. Zhang, and M. S. Waterman. Alignment of optical maps. *Journal of Computational Biology*, 13(2):442–462, 2006.

[20] H. VanSteenHouse. personal communication, 2013.

[21] X. Yang, S. P. Chockalingam, and S. Aluru. A survey of error-correction methods for next-generation sequencing. *Briefings in bioinformatics*, 14(1):56–66, 2013.

[22] S. Zhou et al. A whole-genome shotgun optical map of *yersinia pestis* strain KIM. *Applied and Environmental Microbiology*, 68(12):6321–6331, 2002.

[23] S. Zhou et al. Shotgun optical mapping of the entire *leishmania major* Friedlin genome. *Molecular and Biochemical Parasitology*, 138(1):97–106, 2004.

[24] S. Zhou et al. Validation of rice genome sequence by optical mapping. *BMC Genomics*, 8(1):278, 2007.

[25] S. Zhou, J. Herschleb, and D. C. Schwartz. A single molecule system for whole genome analysis. *Perspectives in Bioanalysis*, 2:265–300, 2007.

[26] S. Zhou, F. Wei, J. Nguyen, M. Bechner, K. Potamousis, S. Goldstein, L. Pape, M. R. Mehan, C. Churas, S. Pasternak, D. K. Forrest, R. Wise, D. Ware, R. A. Wing, M. S. Waterman, M. Livny, and D. C. Schwartz. A Single Molecule Scaffold for the Maize Genome. *PLoS Genetics*, 5(11):e1000711, 11 2009.