Error correcting optical mapping data

Mukherjee, Kingshuk

2018-05-25


http://hdl.handle.net/10138/247093
https://doi.org/10.1093/gigascience/giy061

Downloaded from Helda, University of Helsinki institutional repository.

This is an electronic reprint of the original article.

This reprint may differ from the original in pagination and typographic detail.

Please cite the original version.
Error correcting optical mapping data

Kingshuk Mukherjee, Darshan Washimkar, Martin D. Muggli, Leena Salmela and Christina Boucher

1Department of Computer and Information Science and Engineering, University of Florida, Gainesville,
2Department of Computer Science, Colorado State University, Fort Collins and 3Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki

Abstract

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. Recently it has been used for scaffolding contigs and for assembly validation for large-scale sequencing projects, including the maize, goat, and Amborella genomes. However, a major impediment in the use of this data is the variety and quantity of errors in the raw optical mapping data, which are called Rmaps. The challenges associated with using Rmap data are analogous to dealing with insertions and deletions in the alignment of long reads. Moreover, they are arguably harder to tackle since the data are numerical and susceptible to inaccuracy. We develop cOMet to error correct Rmap data, which to the best of our knowledge is the only optical mapping error correction method. Our experimental results demonstrate that cOMet has high prevision and corrects 82.49% of insertion errors and 77.38% of deletion errors in Rmap data generated from the Escherichia coli K-12 reference genome. Out of the deletion errors corrected, 98.26% are true errors. Similarly, out of the insertion errors corrected, 82.19% are true errors. It also successfully scales to large genomes, improving the quality of 78% and 99% of the Rmaps in the plum and goat genomes, respectively. Last, we show the utility of error correction by demonstrating how it improves the assembly of Rmap data. Error corrected Rmap data results in an assembly that is more contiguous and covers a larger fraction of the genome.

Keywords: optical mapping; error correction

Introduction

In 1993 Schwartz et al. developed optical mapping, a system for creating an ordered, genome-wide, high-resolution restriction map of a given organism’s genome [1]. Since this initial development, genome-wide optical maps have found numerous applications including discovering structural variations and rearrangements [9], scaffolding and validating contigs for several large sequencing projects [4,3, 6], and detecting misassembled regions in draft genomes [7]. Thus, optical mapping has assisted in the assembly of a variety of species including various prokaryote species [8, 9, 10], rice [11], maize [2], mouse [12], goat [3], parrot [6], and Amborella trichopoda [4]. The raw optical mapping data are generated by a biological experiment in which large DNA molecules cling to the surface of a microscope slide using electrostatic charge and are digested with one or more restriction enzymes. The restriction enzymes cut the DNA molecule at occurrences of the enzyme’s recognition sequence, forming a number of DNA fragments. The fragments formed by digestion are painted with a fluorescent dye to allow visibility under laser light and a CCD (Charged Coupled device) camera. Computer vision algorithms then estimate fragment length from consolidated intensity of fluorescent dye and apparent distance between fragment ends.

The resulting data from an experiment are in the form of an ordered series of fragment lengths [13]. The data for each single molecule produced by the system is called an Rmap. Rmap data have a number of errors due to the experimental conditions.
and system limitations. In an optical mapping experiment, it is unlikely to achieve perfectly 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 manifests as missing restriction sites. Additionally, due to the fragile nature of DNA, additional breaks can incorrectly appear as restriction sites. Last, 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 sizing error for smaller fragments. Interested readers will find more details about the causes of these errors in Valouev et al. [14] and Li et al. [15]. Because of all these experimental conditions, Rmap data generated through optical mapping experiment have insertion (added cut sites) and deletion (missed cut sites) errors along with fragment sizing errors.

In most applications of optical map data, the Rmaps need to be assembled into a genome-wide optical map. This is because the single molecule maps need redundant sampling to overcome the presence of the aforementioned errors and because single molecule maps are unable to span the order of 500 Kbp [14]. The first step of this assembly process involves finding pairwise alignments among the Rmaps. In order to accomplish this, the challenge of dealing with missing fragment sizes has to be overcome. This challenge is analogous to dealing with insertions and deletions in the alignment of long reads [16]. In fact, it is arguably harder since the data are numerical. At present, the only nonproprietary algorithmic method for pairwise alignment of Rmaps is the dynamic programming-based method of Valouev et al. [14], which runs in $O(n^2 \times \beta)$ time where $n$ and $\beta$ are the number of fragments in the two Rmaps being aligned. To align an optical map dataset containing $n$ Rmaps, the complexity becomes $O(n^3 \times \epsilon^2)$ where $\epsilon$ is the average size of an Rmap.

This method is inherently computationally intensive. However, if the error rate of the data could be improved, then non-dynamic programming-based methods that are orders of magnitude faster such as TWIN [17], OMA Blast [18], and Maligner [19] could be used for alignment. This would greatly improve the time required to assemble Rmap data. Thus, we present cOMet in order to address this need. To the best of our knowledge, it is the first Rmap error correction method. Our experimental results demonstrate that cOMet has high precision and corrects 82.49% of insertion errors and 77.38% of deletion errors in Rmap data generated from the Escherichia coli K-12 reference genome. Out of the deletion errors corrected, 98.26% are true errors. Similarly, out of the insertion errors corrected, 82.19% are true errors. Furthermore, we show that the assembly of Rmaps is more contiguous and covers a larger fraction of the genome if the Rmaps are first error corrected. It also successfully scales to large genomes, improving the quality of 78% and 99% of the Rmaps in the plum and goat genome, respectively.

**Background**

From a computer science perspective, optical mapping can be seen as a process that takes in two strings, a nucleotide sequence $S$, and a restriction sequence $B$, and produces an array (string) of integers $R$. The array $R$ is an Rmap corresponding to $S$, and contains the string-lengths between cuts produced by $B$ on $S$. Formally, $R$ is defined as follows: $R[j] = \gamma - x$ where $\gamma$ represents the location (starting index) of $j^{th}$ occurrence of $B$ in $S$, and $x$ represents the location of $(j-1)^{th}$ occurrence of $B$ in $S$, and $R[1] = \gamma - 1$ and $R[m] = n - x$. For example, say we have $B = \text{act}$ and $S = \text{atattactgactactaacta}$. The locations of $B$ in $S$ are as follows: 3, 7, 12, 15, 20. Then, $R_j$ will be represented as $R_i = 2, 4, 5, 3, 2$. The size of an Rmap denotes the number of fragments in that Rmap. Therefore, the size of $R_i$ is 6.

We note that millions of Rmaps are produced for a single genome since optical mapping is performed on many cells of the organism and each cell provides thousands of Rmaps. The Rmaps can be assembled to produce a genome-wide optical map. This is analogous to next-generation shotgun sequencing where Rmaps are analogous to reads and a genome-wide optical map is analogous to the assembled whole genome.

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 $R_i = 7, 6, 3, 4$.

The error rates of optical maps depend on the platform used for generating the maps. Li et al. [15] recently studied the error rates of optical maps produced by the Irys system from BioNano Genomics. According to their study, a missing cut site type of error, i.e., error type (1), happens when a restriction site is incompletely digested by the enzyme and causes two flanking fragments to merge into one large fragment. The probability of complete digestion of a restriction site can be modeled as a Bernoulli trial whose probability of success is a function of the size of the two flanking fragments. Additional cut sites, i.e., error type (2), results from random breaks of the DNA molecule. The number of false cuts per unit length of DNA follows a Poisson distribution. The inaccuracy of the fragment sizes, i.e., error type (3), is modeled using a Laplace distribution. If the observed and actual size of a fragment are $o_k$ and $r_k$, respectively, then the sizing error is defined as $e_k = o_k/r_k$ and $e_k \sim \text{Laplace}(\mu, \beta)$, where $\mu$ and $\beta$, the parameters of the laplace distribution, are functions of $r_k$. In practice, when aligning a pair of Rmaps, one should allow for twice the error rate of a single Rmap since each Rmap will deviate from the genomic map by the above parameters.

Valouev et al. [14] provides a dynamic programming algorithm for pairwise alignment, which generates a score for every possible alignment between two Rmaps and returns the alignment that achieves the highest score, which is referred to as the S-score. It is computed within a standard dynamic programming framework, similar to Smith-Waterman alignment [20]. The scoring function is based on a probabilistic model built on the following assumptions: the fragment sizes follow an exponential distribution, the restriction sites follow an independent Bernoulli process, the number of false cuts in a given genomic length is a Poisson process, and the sizing error follows a normal distribution with mean zero and variance following a linear function of the true size. Last, a different sizing error function is used for fragments less than 4 kbp in length since they do not converge to the defined normal distribution. The score of an alignment is calculated as the sum of two functions: one function that estimates and scores the sizing error and a second that predicts and scores the presence of additional and/or missing cut sites between the fragments. The S-score will be used later to evaluate the error rates of the Rmaps.
Methods

Given a set of n Rmaps \( R = (R_1, \ldots, R_n) \), our method aims to detect and correct all errors in \( R \) by considering each \( R_i \in R \) and finding a set of Rmaps that originate from the same part of the genome as \( R_i \). This step is performed heuristically in order to avoid aligning every pair of Rmaps in \( R \).

Preprocessing

Our first step is to remove the first and last fragments from each Rmap in \( R \). These fragments have one of their edges sheared by artifacts of the DNA prep process (preceding the optical mapping process) and not by restriction enzymes. Unless removed, they can misguide alignment between two Rmaps during the error correction process. In addition, short Rmaps, i.e., those that have fewer than 10 fragments, are removed at this stage since any Rmap that contains fewer than 10 fragments is typically deemed too small for analysis even in consensus maps [21]. Next, the data are quantized so that a given genomic fragment is represented by the same value across multiple Rmaps despite the noise. Our quantization method assigns a unique value to a range of fragment sizes by dividing each fragment size by a fixed integer, denoted as \( b \), and rounding to the nearest integer. For example, if an Rmap \( R_i = \{36, 13, 15, 20, 16, 5, 21, 17\} \) is quantized using \( b = 3 \), then the quantized Rmap will be \( R_i^{\text{quantized}} = \{12, 4, 5, 7, 5, 2, 7, 6\} \). Say another Rmap, \( R_j = \{17, 23, 34, 12, 14, 21, 14, 5\} \) has overlap with \( R_i \); however, due to noise in the data, this relation is not apparent. By quantizing \( R_j \) using the same \( b = 3 \), we get \( R_j^{\text{quantized}} = \{6, 8, 11, 4, 5, 7, 5, 1\} \). This allows us to uncover a region (in this case, \( \{4, 5, 7, 5, 1\} \)) that is common to both the Rmaps. It should be noted that, in some cases, a fragment may have different values across two Rmaps even after quantization (e.g., the fragment values 36 from \( R_i \) and 34 from \( R_j \) are quantized to 12 and 11, respectively). The quantized data are used to find the set of related Rmaps, as explained in the next section.

The setting of parameter \( b \) depends on the amount of sizing error in the optical map data. With zero sizing error, \( b \) can be set at 1. As sizing error increases, the value of \( b \) is increased accordingly. If the value of \( b \) is too small, we are not able to uncover relations between overlapping Rmaps. If the value is too large, then unrelated Rmaps have common regions in their quantized states, which makes them appear related. Considering the error rate of optical maps from BioNano genomics, the default value of \( b = 4000 \).

Finding related Rmaps

We refer to two Rmaps as related if their corresponding error-free Rmaps originate from overlapping regions of the genome. Next, we define a k-mer as a string of k consecutive fragments from a (quantized) Rmap. For example, if we have the Rmap \( R = \{3, 3, 5, 2, 6, 5, 1\} \) and \( k = 4 \), then the following k-mers can be extracted from \( R \): \( \{3, 3, 5, 2\}, \{3, 5, 2, 6\}, \{5, 2, 6, 5\}, \{2, 6, 5, 5\}, \{6, 5, 5, 1\} \). In order to avoid aligning all pairs of Rmaps to find the related Rmaps, we use the number of common k-mers to discriminate between pairs of Rmaps that are related and those that are not. To accomplish this efficiently, we first extract all unique k-mers in each quantized Rmap and construct a hash table storing each unique k-mer as a key and the list of Rmaps containing an occurrence of that k-mer as the value. We call this the k-mer index. Next, we consider each \( R_i \) in \( R \) and use the k-mer index to identify the set of Rmaps that have m or more k-mers in common with \( R_i \). Unfortunately, although this set contains all related Rmaps, it also likely contains Rmaps that are not related to \( R_i \). Therefore, we filter this set of Rmaps using a simple heuristic that tries to match each Rmap in this set with \( R_i \) in order to ascertain if it is related to \( R_i \). The heuristic traverses through two Rmaps (\( R_i \) and one Rmap from the set, say \( R_j \)) attempting to match subsets of the fragments from each until it either reaches the end of one Rmap or it fails to match the fragments. We start the traversal from the first matching k-mer between \( R_i \) and \( R_j \). We denote the position of the next fragment to be matched in \( R_i \) and \( R_j \) as \( x \) and \( y \), respectively, and assume that each fragment prior to these positions is matched. Next, we consider all combinations of matching the fragments at positions \( x, x + 1, \) and \( x + 2 \) of \( R_i \), with fragments at positions \( y, y + 1, \) and \( y + 2 \) of \( R_j \). We evaluate the cost of each combination based on the difference in the total size of fragments from \( R_i \) and \( R_j \). That is for \( \alpha, \beta = [0, 2] \),

\[
\text{cost}(x + \alpha, y + \beta) = \sum_{g \in b} R_i[g] - \sum_{h \in b} R_j[h]
\]

where \( R_i[g] \) and \( R_j[h] \) denote the g-th and h-th fragments of \( R_i \) and \( R_j \), respectively. We select the combination with the least cost; if there exists a tie, we select the match that has the least number of added or missing cut sites (i.e., the combination with the least value of \( \alpha + \beta \)). If this selected match leads to a cost that is greater than a specified threshold (which was set to 25% of the larger-sized fragment in practice), then we conclude that there is not a match at these positions and return that \( R_i \) and \( R_j \) are unrelated. Otherwise, we increment \( x \) and \( y \) accordingly and move onto the next fragments. If this heuristic continues until the last fragment of either \( R_i \) or \( R_j \) is reached, then we return that \( R_i \) and \( R_j \) are related. Using this heuristic, we filter out the Rmaps that were deemed to be related based on the number of k-mers in common with \( R_i \), but are in fact unrelated to \( R_i \).

The setting of parameters \( k \) and \( m \) are correlated. If the value of \( k \) is increased, that makes the k-mers more specific, hence, the value of \( m \) is lowered. On the other hand, if the value of \( k \) is reduced, then we increase the value of \( m \). The value of \( k \) is increased when there are fewer insertion and deletion errors and decreased otherwise. The default values are \( k = 4 \) and \( m = 1 \).

Rmap alignment

Next, for each \( R_i \) in \( R \), we use the alignment method of Valouev et al. [14] to find the S-score of all pairwise alignments between \( R_i \) and each Rmap in its set of related Rmaps. The Rmaps that have an alignment score, i.e., S-score less than a defined threshold (which we denote as \( S_0 \)), are removed from the set of related Rmaps, and the alignments of the remaining Rmaps are stored in a multiple alignment grid, denoted as \( A_k \). This grid is a two-dimensional array of integer pairs, where the number of rows is equal to the number of remaining Rmaps in the set of related Rmaps of \( R_i \) and the number of columns is equal to the number of fragments in \( R_i \). An element of this array, \( A_k[j, k] \), stores an integer pair in the form of \( (x, y) \) representing that \( x \) fragments of \( R_i \) in the optimal alignment between \( R_i \) and \( R_j \). Figure 1 illustrates an example of \( A_k \). The first fragment of \( R_j \) does not match with any fragment of \( R_i \); therefore, \( (0,0) \) is stored at this position. Fragments 2, 5, 6, 8, and 9 of \( R_j \) each matches with one fragment of \( R_i \), e.g., 1, 3, 4, 7, and 8, respectively. To represent these matches, we store a \((1,1)\) in the 2nd, 5th, 6th, 8th, and 9th columns of row 1. Fragments 3 and 4 of \( R_i \) match with...
Error correcting the optical mapping data

One fragment of \( R_i \), i.e., the 2nd fragment. To represent this, we store \((2,1)\) in \( A_i[j, 3] \) and \( A_i[j, 4] \). Fragment 7 of \( R_i \) matches with two fragments of \( R_j \), i.e., the 5th and 6th fragments. To represent this, we store \((1,2)\) in \( A_i[j, 7] \). Fragments 10 and 11 of \( R_i \) match with two fragments of \( R_j \), i.e., the 9th and 10th fragments. To represent this, we store \((2,2)\) in \( A_i[j, 10] \) and \( A_i[j, 11] \). Finally, fragments 12 and 13 match with three fragments of \( R_i \), i.e., fragments 11, 12, and 13. In this case, we store \((2,3)\) in positions \( A_i[j, 12] \) and \( A_i[j, 13] \).

The setting of parameter \( S_i \) controls the number of Rmaps that are included in the multiple-alignment grouping of an Rmap. If we increase the value of \( S_i \), fewer Rmaps will be added to the grid, but the ones included will be of higher quality (i.e., have greater overlap with the Rmap under consideration). The default value for the parameter \( S_i = 8 \). We show in the Experiment section how we select this value.

Error correcting using the consensus

The multiple alignment grid is used to find the consensus grid, denoted as \( C_i \), for Rmap \( R_i \). The grid \( C_i \) is a one-dimensional array of integer pairs with size equal to the number of fragments in \( R_i \). The grid is constructed for each \( R_i \) in \( R \) by iterating through each column of \( A_i \) and finding the most frequent integer pair, breaking ties arbitrarily. The most frequent integer pair is stored at each position of \( C_i \) if the frequency is above a given threshold \( d \); otherwise, \((0,0)\) is stored. Figure 2 illustrates the construction of a consensus grid from an alignment grid. The type of error in each fragment of \( R_i \) can be identified using \( C_i[k] = (x, y) \) as follows: if \( x \) and \( y \) are equal, then a sizing error occurs at the \( k \)-th fragment of \( R_i \), otherwise, if \( x > y \) then an additional cut site exists, and, last, if \( x < y \) then a missing cut site exists. Next, we use \( C_i \) and \( A_i \) to correct these errors in \( R_i \). For each fragment of \( R_i \), we consider the consensus stored at the corresponding position of \( C_i \), identify the positions in the corresponding column of \( A_i \) that are equal to it, and replace the fragment of \( R_i \) with the mean total fragment size computed using the values at those positions in \( A_i \). If \( C_i \) is equal to \((0,0)\) at any position, then the fragment at that position in \( R_i \) remains unchanged since it implies that there is no definitive result about the type of error in that position. In addition, if consecutive positions in \( C_i \) are discordant, then the fragments in those positions in \( R_i \) also remains unchanged. For example, if there is a \((2,1)\) consensus at some position of \( C_i \), then we expect the preceding or successive position to also have a \((2,1)\) consensus. However, if this is not the case, then we do not error correct those fragments since the consensus is discordant at those positions. Figure 2 shows this error correction. As it is illustrated, to error correct the second fragment of \( R_i \), we compute the average of the matched fragments from related Rmaps 2, 3, 4, 5, and 6 and replace the second fragment of \( R_i \) with that value as shown in Fig. 2. Similarly, to correct the third fragment in this example, we identify that \((2,1)\) is in the consensus, which implies that the majority of the related Rmaps are such that two fragments of \( R_i \) match with one fragment from the set of related Rmaps and therefore replace the third and fourth fragments with the average from the corresponding Rmaps and positions.

The threshold \( d \) determines the accuracy and precision of error correction. A high value of \( d \) improves precision but lowers accuracy as many fragments are left uncorrected. Similarly, a low value of \( d \) improves accuracy but lowers precision. The default setting is \( d = 3 \).

Complexity

We define \( \ell \) to be the length of the longest Rmap in \( R \). Quantization of the Rmaps takes \( O(\ell \times n) \) time. Constructing the \( k \)-mer index also takes \( O(\ell \times n) \) time. The \( k \)-mer index stores the occurrences of each quantized \( k \)-mer across all Rmaps. Let \( u \) be the maximum frequency of a \( k \)-mer. That is, a \( k \)-mer occurs in max \( u \) Rmaps (in practice \( u < n \)). Then, the complexity of finding related Rmaps from the \( k \)-mer index is \( O(n \times \ell \times u) \). For each Rmap, the filtering heuristic runs in time linear to the size of the Rmap. Therefore, filtering the set of related Rmaps also takes linear \( O(\ell \times n) \) time. The most expensive step is the pairwise alignment that uses the Valouev et al. aligner. As mentioned earlier, this aligner is based on DP and therefore has a \( O(\ell^2) \) time complexity to perform one pairwise alignment. If the maximum cardinality of the set of related Rmaps for any Rmap is \( u \), then the total complexity of this step is bounded by \( O(n \times u \times \ell^2) \). The value of \( u \) depends on the coverage of the optical map data. The alignment generated using the Valouev et al. method is stored in the multiple alignment grid in constant time, and it takes \( O(n \times u \times \ell) \) time to generate the consensus maps for \( n \) Rmaps and error-correct them. Thus, the runtime of COMET is \( O(n \times u \times \ell^2) \).

Datasets

We performed experiments on both simulated and real data. For the real data, we used the Rmap data from the plum [22] and domestic goat [3] sequencing projects. These datasets were built on the OpGen mapping platform and are more error prone. We also experimented on a human dataset [23] built on the new BioNano platform. This dataset is built using the latest optical mapping
Multiple alignment grid (A_i)

<table>
<thead>
<tr>
<th>Ai</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
<th>7th</th>
<th>8th</th>
<th>9th</th>
<th>10th</th>
<th>11th</th>
<th>12th</th>
<th>13th</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
<td>(2.1, 2.107)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(1.2)</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
<td>(1.1)</td>
<td>(1.2)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(0.0)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(4.344)</td>
<td>(2.1)</td>
<td>(4.344)</td>
<td>(1.1)</td>
<td>(4.888)</td>
<td>(1.1)</td>
<td>(1.2)</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
</tr>
<tr>
<td>3</td>
<td>(0.0)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(4.129)</td>
<td>(2.1)</td>
<td>(4.129)</td>
<td>(1.1)</td>
<td>(4.132)</td>
<td>(1.1)</td>
<td>(1.2)</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(1.1)</td>
</tr>
<tr>
<td>4</td>
<td>(0.0)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(4.311)</td>
<td>(2.1)</td>
<td>(4.311)</td>
<td>(1.1)</td>
<td>(8.964)</td>
<td>(1.1)</td>
<td>(1.2)</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
</tr>
<tr>
<td>5</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(4.661)</td>
<td>(2.1)</td>
<td>(4.661)</td>
<td>(2.1)</td>
<td>(10.692)</td>
<td>(2.1)</td>
<td>(3.3)</td>
<td>(3.3)</td>
<td>(1.1)</td>
<td>(1.1)</td>
</tr>
<tr>
<td>6</td>
<td>(0.0)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(4.710)</td>
<td>(2.1)</td>
<td>(4.710)</td>
<td>(1.1)</td>
<td>(9.432)</td>
<td>(1.1)</td>
<td>(1.2)</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
</tr>
</tbody>
</table>

Consensus grid (C_i)

| C_i | (0.0) | (1.1) | (2.1) | (2.1) | (1.1) | (1.1) | (1.2) | (1.1) | (1.1) | (1.1) | (2.2) | (2.2) | (1.1) |

Error correction

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C_i</td>
<td>(0.0)</td>
<td>(1.1)</td>
<td>(2.1)</td>
<td>(2.1)</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(1.1)</td>
<td>(2.2)</td>
<td>(2.2)</td>
<td>(1.1)</td>
<td>(1.2)</td>
</tr>
</tbody>
</table>

Table 1: Summary of the real and simulated data

<table>
<thead>
<tr>
<th>Genome</th>
<th>Size</th>
<th>No. of Rmaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>E. coli</td>
<td>4.6 Mbp</td>
<td>2,504</td>
</tr>
<tr>
<td>Plum</td>
<td>284 Mbp</td>
<td>749,895</td>
</tr>
<tr>
<td>Goat</td>
<td>2.66 Gbp</td>
<td>3,447,997</td>
</tr>
</tbody>
</table>

Rmaps with fewer than 10 fragments were omitted from all the experiments. cOMet was run on the remaining 2,504, 548,779, and 3,049,439 Rmaps for the E. coli, plum, and goat genomes, respectively.

Table 2: Summary of the real and simulated BioNano data.

<table>
<thead>
<tr>
<th>Genome</th>
<th>Size</th>
<th>No. of Rmaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>E. coli</td>
<td>4.6 Mbp</td>
<td>123,251–157,743</td>
</tr>
<tr>
<td>Human</td>
<td>3.2 Gbp</td>
<td>793,199</td>
</tr>
</tbody>
</table>

OMSim was used to simulate eight different BioNano datasets, each of which had varying error rates and, thus, had a different number of Rmaps.

Table 3: Results for the data simulated from E. coli K-12 sub-str. MG 1655.

| Total no. of insertion errors corrected | 556 |
| TPR of corrected insertions | 82.49% | (457) |
| FPR of corrected insertions | 0.21% | (99) |
| Total no. of deletion errors corrected | 5,894 |
| TPR of corrected deletions | 77.38% | (5,792) |
| FPR of corrected deletions | 0.25% | (102) |

The data was simulated according the algorithm described in Datasets. This simulation resulted in 2,505 Rmaps containing 7,485 deletion and 554 insertion errors.

Experiments and Discussion

We performed all experiments on Intel E5-2698v3 processors with 192 GB of random access memory (RAM) running 64-bit Linux. The input parameters to cOMet include b (quantization bucket size), k (k-mer value), m (the number of k-mers needed to be conserved between two Rmaps), and d (the minimum number of Rmaps required to form consensus at a position). The default parameters are b = 4,000, k = 4, m = 1, and d = 3 and led to the best result across all datasets.

In addition, we simulated Rmap data from E. coli K-12 sub-str. MG 1655 as follows. First, the reference genome was copied 200 times, and uniformly distributed random loci were selected for each of these copies. These loci form the ends of a single molecule that would undergo in silico digestion. Next, molecules smaller than 150 Kbp were discarded, and the cleavage sites for the RsrII enzyme were then identified within each of these simulated molecules. These error-free Rmap data are used for validating the output of our method. Last, deletion, insertion, and sizing errors were incorporated into the error-free Rmaps according to the error model discussed by Li et al. [15]. The error model was described earlier in the Background section. This simulation resulted in 2,505 Rmaps containing 7,485 deletion and 554 insertion errors.

Last, we simulated optical map data from a simulation software called OMSim [24] that generates synthetic optical maps that mimic real Bionano Genomics data. The software takes two parameters as input: the false positive rate (FPR) rate, which is the number of additional cut sites erroneously inserted per 100kbp, and the false negative rate (FNR), which is the percentage of times a cut site is missed. Using this method, we simulated eight datasets of Rmaps from E. coli K-12 sub-str. MG 1655 using the restriction enzyme BspQI. The default FPR and FNR for BspQI are 1% and 15%, respectively. We generated additional datasets with the following error rates (FPR, FNR): (0.5%, 15%), (1.0%, 15%), (2.0%, 15%), (5.0%, 15%), (1.0%, 5%), (1.0%, 25%), (2.0%, 5%), and (2.0%, 25%).
Determining the value of $S_t$

The setting of the parameter $S_t$ depends on the sensitivity of the Valouev et al. aligner. If the alignment score between two Rmaps is less than $S_t$, then the aligned Rmaps are deemed to be unrelated. We say an Rmap, $R_u$, is overlapping with an Rmap, $R_i$, if at least 50% of $R_u$ overlaps with $R_i$. That is, either the first half or the second half of $R_u$ is entirely and exactly (exact fragment matches) contained in $R_i$.

We carried out the following experiment to determine the optimum setting for $S_t$. From the set of simulated error-free Rmaps, we computed the set of overlapping Rmaps for each Rmap. We denote this set as related Rmaps. Then, we used the Valouev et al. aligner to score all pairwise wise alignments between the simulated Rmaps (with errors added) and plot the scores in the form of a histogram, which is shown in Fig. 3. The percentage of related Rmaps with an $S$-score less than 8 is 6.06%. Hence, we choose the setting of $S_t = 8$.

Experiments with our simulated data

The cOMet error correction was run on the simulated *E. coli* data. The corrected Rmaps were then aligned to the error-free Rmaps to determine the number of corrected insertions and deletions. The results of this experiment are shown in Table 3. To determine the quality of error correction, we computed the true positive rate (TPR), which is the ratio between the number of insertion (or deletion) errors that cOMet correctly identified and removed and the number of insertion (deletion) errors, and the FPR, which is the ratio between the number of insertion (or deletion) errors that cOMet incorrectly identified and removed and the total number of fragments not containing an insertion (deletion) error. The TPR is 82.49% and 77.38% with respect to the number of corrected insertions and deletion errors; whereas the FPR is 0.21% and 0.25% with respect to the number of corrected insertions and deletion errors. This demonstrates the high accuracy of the correction made by cOMet. Our method also has high precision. Of the deletion errors corrected, 98.26% are true errors. Similarly, of the insertion errors corrected, 82.19% are true errors.

Additionally, for each corrected Rmap, we computed the alignment $S$-score of both the original Rmap and the corrected Rmap with the error-free Rmap. We found that for 96.5% of the Rmaps, the $S$-scores improved after error correction. In other words, cOMet brought 96.5% Rmaps closer to their error-free state. The mean $S$-score before error correction was 44.91 and it improved by 14.03% to 51.30 after error correction. For 17.5% of the Rmaps (415 Rmaps), the $S$-score improved by more than 10. Last, we mention that the error correction was achieved in 241 central processing unit (CPU) seconds and using 79.54 MB of memory.

To demonstrate the importance of error correction, we assembled the Rmaps before and after error correction using the Valouev et al. assembler [25]. Table 4 summarizes the results of this experiment. We assembled the uncorrected data into five assembled optical maps and the error-corrected data into two assembled optical maps. The N50 statistic of the assembly increased from 1,242 Kbp for the uncorrected data to 3,348 Kbp for the corrected data. Next, we aligned each assembled map to the genome-wide (error-free) optical map using the Valouev et al. aligner in order to locate their positions on the genome and calculate the percentage of the genome that was covered by at least one of the assembled maps. The genome fraction covered by the five assembled maps from the uncorrected Rmaps was 80%, while the genome fraction covered by the two assembled maps from the corrected Rmaps was 82%. Moreover, the assembled maps from the uncorrected data had 47 insertion and deletion errors when aligned to the reference, while the error-corrected data had only 34 such errors. In order to further contextualize these results, we assemble the error-free Rmap dataset and summarize this assembly in Table 4.

Experiments with OMSim data

To present the robustness of our method and its applicability across datasets, we conducted experiments on synthetic data from an optical map-simulating software called OMSim [24]. As described in the Datasets section, we generated eight datasets of synthetic optical maps by varying the insertion and deletion error rates.

In the first experiment, we fixed the FNR at 15% and varied the FPR between 0.5, 1.0, 2.0, and 5.0, respectively. For each of the four datasets, we align each Rmap (using the Valouev et al. aligner) before and after error correction to the reference optical map obtained using the same restriction enzyme and report the percent of Rmaps whose alignment $S$-score increased after error correction and the mean increase in the $S$-score. We note that for each Rmap, the aligner returns the highest-scored alignment, and the score represents how closely the Rmap aligns to the reference genome-wide optical map. Table 5 summarizes the results from this experiment. We observe that the efficiency of error correction improves as the FPR is initially increased. When the FPR reaches a high value of 5, the efficiency of error correction drops. The mean $S$-score improves by more than 9 (~10%) when the FPR is reasonable.

In the second experiment, we first fix the FPR at 1.0 and vary the FNR between 5%, 15%, and 25%, respectively. We then fix the FPR at 2 and vary the FNR between 5%, 15%, and 25%, respectively. We report the same results as in the previous experiment. Table 6 shows the results. Similar to the previous experiment, we find that the efficiency of error correction improves as the FNR increases from 5% to 25%. The error correction improves the quality of a high percentage of Rmaps (>70%) for all values of parameters.

Experiments with real data

Table 7 summarizes the results of running cOMet on the plum and goat datasets. The plum and goat datasets do not contain error-free Rmaps. Therefore, we are restricted to reporting the number of corrections made and the improvement to the $S$-score. In order to compute the $S$-score before and after error correction, we generated an in silico digested genome-wide optical map from the reference genome and aligned both the uncorrected and corrected Rmap to the genome-wide optical map. If it aligned to multiple positions, then we considered the alignment position where the corrected Rmap aligned with greatest $S$-score and considered the difference in the $S$-score when the uncorrected and corrected Rmap aligned to that position. However, we note that this process is error prone because of the fragmented nature of the draft genomes and possible misassemblies present in the genomes. We observed that the $S$-score after error correction improved for 78% of the plum Rmaps and 99% of the goat Rmaps. Figures 4 and 5 show the histograms of the distribution of $S$-scores before and after error correction. For the plum genome, the mean $S$-score improved from 8.60 before error correction to 14.72 after error correction (a 71% improvement in the score), while for the goat genome, it improved from 9.38 before...
Figure 3: Distribution of S-scores of Rmap alignments between related Rmaps and unrelated Rmaps.

Table 4: Assembly results of uncorrected Rmaps, corrected Rmaps, and error-free Rmaps using the Valouev et al. assembler

<table>
<thead>
<tr>
<th>Rmap status</th>
<th>Assembled Map id</th>
<th>Number of fragments</th>
<th>Map length (Kbp)</th>
<th>Alignment location in reference (start-loci, end-loci)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncorrected Rmaps</td>
<td>Assembled Map,0</td>
<td>75</td>
<td>921.41</td>
<td>(246,321)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,1</td>
<td>88</td>
<td>1,242.40</td>
<td>(11,95)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,2</td>
<td>30</td>
<td>531.65</td>
<td>(225,194)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,3</td>
<td>44</td>
<td>759.16</td>
<td>(181,228)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,4</td>
<td>107</td>
<td>1,699.60</td>
<td>(87,194)</td>
</tr>
<tr>
<td>Corrected Rmaps</td>
<td>Assembled Map,0</td>
<td>102</td>
<td>1,397.60</td>
<td>(225,322)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,1</td>
<td>237</td>
<td>3,348</td>
<td>(8,230)</td>
</tr>
<tr>
<td>Error-free Rmaps</td>
<td>Assembled Map,0</td>
<td>60</td>
<td>808.74</td>
<td>(185,239)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,1</td>
<td>91</td>
<td>1,100.5</td>
<td>(241,324)</td>
</tr>
<tr>
<td></td>
<td>Assembled Map,2</td>
<td>104</td>
<td>2,474.4</td>
<td>(19,185)</td>
</tr>
</tbody>
</table>

The Rmaps are simulated from the E. coli genome. Each assembled map is aligned to the reference genome-wide (error-free) optical map using the Valouev et al. aligner. The genome-wide optical map contains 383 fragments.

Table 5: Efficiency of error correction when the FPR is varied and the FNR is fixed at 15%

<table>
<thead>
<tr>
<th>FPR</th>
<th>No. of Rmaps</th>
<th>Percent of Rmaps with improved S-score</th>
<th>Mean S-score before error correction</th>
<th>Mean S-score after error correction</th>
<th>Mean S-score correction improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>129,820</td>
<td>93.42</td>
<td>78.66</td>
<td>91.12</td>
<td>12.46</td>
</tr>
<tr>
<td>1.0</td>
<td>126,133</td>
<td>94.01</td>
<td>76.25</td>
<td>89.44</td>
<td>13.19</td>
</tr>
<tr>
<td>2.0</td>
<td>140,623</td>
<td>92.99</td>
<td>71.93</td>
<td>85.29</td>
<td>13.36</td>
</tr>
<tr>
<td>5.0</td>
<td>130,019</td>
<td>81.35</td>
<td>64.65</td>
<td>71.36</td>
<td>6.71</td>
</tr>
</tbody>
</table>

correction to 16.97 after correction (an 80.92% improvement in the score).

We also measured the genome coverage, i.e., the fraction of the genome covered by at least one Rmap, for both the original Rmaps and the corrected Rmaps as follows. First, we aligned all Rmaps to the genome-wide optical map and then picked the best alignment for each original Rmap and each corrected Rmap. Based on these alignments, we then computed the fraction of the genome covered by at least one original Rmap and the fraction of the genome covered by at least one corrected Rmap. On the goat genome, the genome coverage was 73.08% before correction and it increased to 84.56% after correction. The increase in genome coverage shows that our method is able to correct Rmaps from across the genome. Furthermore, it shows that even...
Table 6: Efficiency of error correction when the FNR is varied

<table>
<thead>
<tr>
<th>FPR</th>
<th>FNR (%)</th>
<th>No. of Rmaps with improved S-score</th>
<th>Mean S-score before error correction</th>
<th>Mean S-score after error correction</th>
<th>Mean S-score improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>5</td>
<td>142,684</td>
<td>87.36</td>
<td>81.32</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>126,133</td>
<td>94.01</td>
<td>76.25</td>
<td>13.19</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>123,252</td>
<td>96.09</td>
<td>70.24</td>
<td>17.20</td>
</tr>
<tr>
<td>2.0</td>
<td>5</td>
<td>148,912</td>
<td>89.23</td>
<td>76.54</td>
<td>7.93</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>140,623</td>
<td>92.99</td>
<td>71.93</td>
<td>13.36</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>130,763</td>
<td>93.02</td>
<td>66.95</td>
<td>14.7</td>
</tr>
</tbody>
</table>

Figure 4: Alignment scores of Rmaps from the plum genome with the reference optical map. Before error correction, the S-score had a mean of 8.6 with a standard deviation of 6.49. After error correction, the mean S-score improved to 14.72 with standard deviation of 6.72.

Table 7: Results on the Rmap data of plum and goat genomes

<table>
<thead>
<tr>
<th>Genome name</th>
<th>Plum</th>
<th>Goat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>7.4 days</td>
<td>105.7 days</td>
</tr>
<tr>
<td>Memory</td>
<td>12.20 GB</td>
<td>113.56 GB</td>
</tr>
<tr>
<td>No. of insertion errors corrected</td>
<td>433,282</td>
<td>2,530,060</td>
</tr>
<tr>
<td>No. of deletion errors corrected</td>
<td>430,329</td>
<td>3,187,023</td>
</tr>
</tbody>
</table>

Peak memory was measured as the maximum resident set size as reported by the operating system. Running time is the user process time, also reported by the operating system.

Next, we ran experiments on the human dataset. Again, since we do not possess the error-free Rmaps corresponding to the raw Rmaps for this dataset, we follow a similar evaluation method as in the previous experiments. We performed our evaluation on an in silico digested human reference genome (GenBank assembly accession: GCA_000001405.15, Genome Reference Consortium Human Build 38) using BspQI, which was the restriction enzyme that was used for generating the Rmap data. cOMet improved the S-score of 74.78% of the Rmaps. The average S-score improved from 85.96 before error correction to 88.65 after error correction.
Mukherjee et al.

Figure 5: Alignment scores of Rmaps from the goat genome with the reference optical map. The mean and standard deviation of the S-scores before error correction were 9.38 and 6.54, respectively. After error correction, the mean S-score improved to 16.97 with a standard deviation of 6.21.

Conclusion

Error correction of high-throughput sequencing data has become an imperative preprocessing step in genome assembly since 2008 when Chaisson and Pevzner showed the dramatic improvement it can have on the quality of the assembly [26, 27, 28]. For example, after error correction, the contig N50 size of an assembly of Rhodobacter sphaeroides improved from 233 bp to 7,793 bp using the same assembler [28]. Due to this inarguable benefit on genome assembly, many methods have been developed for error correction of sequence reads, including BFC [29], Coral [30], EULER [26, 31], and Reptile [32]. Unfortunately, even though there has been a massive effort in error correction of sequence data, there currently does not exist a publicly released method for error correction of Rmap data—a method that would likely improve the quality of genome-wide optical map assemblies and allow such assemblies to be computed with greater efficiency.

Here, we presented cOM\textregistered, an error correction method for Rmap data and demonstrated that it corrects and improves the quality of a high percentage of Rmaps in both simulated and real datasets. As previously discussed, Rmap data are subject to high error rates. In addition to insertion and deletion errors, they contain sizing errors that necessitate the use of a dynamic programming algorithm for pairwise alignment and, subsequently, assembly. By correcting a significant number of errors in Rmap data, cOM\textregistered can make it possible to use faster alignment methods [18, 19, 17] and explore the development of more efficient Rmap assembly algorithms.

Availability of source code

Project name: cOM\textregistered
Project home page: https://github.com/kingufl/cOM\textregistered
Operating system(s): Linux
Programming language: C++
Other requirements: GCC version 5.2.0 or higher
License: GNU General Public License
Research resource identifier: SCR_016276
An archival copy of the code is available via the GigaScience repository GigaDB[33].

Availability of supporting data

The optical mapping data for plum and goat are publicly available and can be accessed from their respective manuscripts [3, 22] and via GigaDB [34]. The human optical map data can also be accessed from its manuscript [23]. The simulated data for E. coli are provided in the github repository along with the python scripts used to generate it, and snapshots of the data and code are also included in GigaDB[33].

Additional material

In Figure S.1 we show the distribution of lengths of Rmaps whose S-score increases after error correction. From the distribution, we can tell that our method is able to error correct Rmaps of all sizes. We also show the distribution of fragment sizes from Rmaps whose score increases after error correction in Figure S.2.

Figure S.1: Distribution of Rmap lengths whose S-score increased after error correction. The Rmaps are simulated from the E. coli K-12 substr. MG 1655 as explained in the text.

Figure S.2: Distribution of fragment sizes of Rmaps whose S-score increased after error correction. The Rmaps are simulated from the E. coli K-12 substr. MG 1655 as explained in the text.

Abbreviations

CPU: central processing unit; FNR: false negative rate; FPR: false positive rate; RAM: random access memory; TPR: true positive rate; CCD: charged coupled device.
Funding

K.M., D.W., M.M., and C.B. were funded by the National Science Foundation (1618814). L.S. was funded by the Academy of Finland (grants 284598 [CoEGR], 308030, and 314170).

References