Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Subject Section

Aligning Optical Maps to De Bruijn Graphs

Kingshuk Mukherjee [1], Bahar Alipanahi [1], Tamer Kahveci [1], Leena Salmela [2] and Christina Boucher [1,]

[1]Department of Computer and Information Science and Engineering, College of Engineering, University of Florida, Gainesville, FL.
[2]Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, Finland.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Optical maps are high resolution restriction maps that give a unique numeric representation to a genome. Used in concert with sequence reads, they provide a useful tool for genome assembly and for discovering structural variations and rearrangements. Although they have been a regular feature of modern genome assembly projects, optical maps have been mainly used in post processing step and not in the genome assembly process itself. Several methods have been proposed for pairwise alignment of single molecule optical maps — called Rmaps, or for aligning optical maps to assembled reads. However, the problem of aligning an Rmap to a graph representing the sequence data of the same genome has not been studied before. Such an alignment provides a mapping between two sets of data: optical maps and sequence data which will facilitate the usage of optical maps in the sequence assembly step itself.
Results: We define the problem of aligning an Rmap to a de Bruijn graph and present the first algorithm for solving this problem which is based on a seed-and-extend approach. We demonstrate that our method is capable of aligning 73% of Rmaps generated from the *E. coli* genome to the de Bruijn graph constructed from short reads generated from the same genome. We validate the alignments and show that our method achieves an accuracy of 99.6%. We also show that our method scales to larger genomes. In particular, we show that 76% of Rmaps can be aligned to the de Bruijn graph in the case of human data.
Availability: The software for aligning optical maps to de Bruijn graph, omGraph is written in C++ and is publicly available under GNU General Public License at https://github.com/kingufl/omGraph
Contact: Kingshuk Mukherjee (kingdpg@ufl.edu)

1 Introduction

Optical mapping is a system for creating an ordered, genome-wide, high-resolution restriction map of a given organism's genome. It was developed by Schwartz et al. (1993), and then later automated in order to produce optical maps at increasingly-high throughput. The optical mapping system works as follows (Samad et al., 1995; Dimalanta et al., 2004): DNA molecules are decoiled and elongated, restriction enzymes are applied to break the DNA into fragments at the loci where the restriction sites occur, the fragments are highlighted with fluorescent dye and digitally photographed under a microscope. The images are analyzed to determine the relative order and size of the fragments (Neely et al., 2011). These ordered lists of fragment sizes are called *restriction maps (Rmaps)*, and thus, correspond to the output of this system. High-throughput technologies automate this process and produce millions of Rmaps simultaneously. Lastly, the Rmaps are typically assembled to genome wide optical maps, which are then used for further analysis.

Ever since the generation of the first Rmap datasets, significant attention has been paid to the development of efficient algorithms for alignment of the data (Valouev et al., 2006a). Now — almost 20 years later — there exists methods to find pairwise alignments between Rmaps (Valouev et al., 2006a), to align in silico digested contigs to a genome wide optical maps (Nagarajan et al., 2008; Muggli et al., 2014), and to find alignments between Rmaps and/or genome wide optical maps (Leung et al., 2017; Mendelowitz et al., 2016; Muggli et al., 2018). With these alignment methods Rmaps can be assembled, and combined with sequence data for other downstream analysis. After assembly and/or alignment, optical mapping data have successfully assisted in the reconstruction of several

genomes (Lin et al., 1999), validated the assembly of large eukaryote genomes (Dong et al., 2013; Ganapathy et al., 2014; Vij et al., 2016; Beier et al., 2017; Daccord et al., 2017; Jarvis et al., 2017), detected structural variations and rearrangements (Teague et al., 2010), and identified mis-assembled regions in draft genomes (Muggli et al., 2015).

One of the most computationally challenging aspects of analyzing optical mapping data is assembly of the Rmap data. Currently, there exists only a single non-proprietary tool for Rmap assembly (Valouev et al., 2006b), which is unable to scale to even moderately-large genomes, such as rice. Bionano Genomics has an assembler, yet its efficiency and scalability has not been documented in the current literature. Nonetheless, it remains clear that the complex errors in the data (see Subsection 3) make assembling the data a difficult problem. A potential alternative to analyzing optical mapping and sequence data is to first align the Rmaps to a graph constructed from the sequence data. This alternative avoids assembly of the Rmap data. One of the most common graphical representation of sequence data is the *de Bruijn graph*, which is most-easily defined using a construction algorithm. Given a set of sequences $S = [s_1, .., s_n]$ and an integer k, a de Bruijn graph $G = (V, E)$ is built for S by creating a directed edge for each unique k-length subsequence (k-mer) in S, labelling the outgoing and incoming nodes with the prefix and suffix of the k-mer, and lastly, gluing all nodes that have the same label. After constructing the de Bruijn graph for a set S, the graph is traversed and the sequences corresponding to the paths are returned. In contrast to Rmap assembly, there has been a plethora of advancements in assembling genomes from sequence data (Zerbino and Birney, 2008; Simpson et al., 2009; Bankevich et al., 2012). The outcome of these advancements is the ability to build the de Bruijn graph efficiently on even relatively-large genomes (Bradnam et al., 2013; Boisvert et al., 2012; Butler et al., 2008).

Identifying alignments of Rmaps to a de Bruijn graph constructed on a set of sequence reads, reference genome, or assembled contigs can then be used for other downstream analysis. Especially when a high quality reference genome is not available, it is advantageous to use a de Bruijn graph rather than a set of contigs for Rmap alignment because the potential connections between the individual contigs present in the graph can be leveraged in the alignment. The most obvious analyses that can be accomplished with this alignment include the detection of mis-assembled contigs (Muggli et al., 2015; Lin et al., 2012), reconstruction of the genomes of the same species (Lin et al., 1999), and identification of structural variants (Teague et al., 2010). Yet, the problem of aligning Rmaps to a de Bruijn graph has not been proposed or solved.

Our Contributions. In this paper, we formally define the problem of aligning an Rmap to a de Bruijn graph and present the first algorithm for solving this problem. Our algorithm is based on a seed-and-extend approach, where paths ("seeds") that align to short segments of the Rmap are found and then extended to create a single path that aligns to the Rmap. We demonstrate that using our method, we are capable of aligning 73% of Rmaps generated from the *E. coli* genome to the de Bruijn graph constructed from short reads generated from the same genome. We validate the alignments and show that our method achieves an accuracy of 99.6%. Similarly, we show our method is capable of scaling to larger genomes. In particular, we show that 76% of Rmaps can be aligned to the de Bruijn graph in the case of human data.

2 Related Work

AGORA (Lin et al., 2012) performs sequence assembly guided by optical maps. It uses optical maps to guide in the sequence assembly by eliminating alternate paths that are inconsistent with the optical map. In contrast to our work, AGORA uses assembled genome wide optical maps. As compared to Rmaps the genome wide optical map are longer, i.e. a single map covers most of the genome, and their error level has been significantly reduced. Thus the alignment algorithm in AGORA and our alignment problem are not directly comparable. Other tools by Shelton et al. (2015) and Pan et al. (2018) make use of optical maps in their pipeline for scaffolding.

The most commonly studied alignment problem of optical maps is the alignment between two single molecule optical maps to detect overlaps. A slight modification of this problem deals with aligning an Rmap to the reference optical map to find its placement in the genome. Valouev et al. (2006a) developed an algorithm that solved both versions of the problem. They computed the best alignment using a dynamic scoring scheme similar to the algorithm by Needleman and Wunsch (1970). Their scoring function is defined as a log likelihood ratio test that takes into account the various errors prevalent in the optical map data.

Another optical mapping alignment method is SOMA (Nagarajan et al., 2008), which aligns contigs from a genome assembler to a genome-wide optical map. SOMA uses a dynamic programming algorithm to perform the alignment but optimizes a different scoring function which imposes a fixed cost penalty on missing and additional cut sites and uses a chi-squared function to penalize for sizing errors.

Over the past few years, new data structures and algorithms have been applied to the optical map alignment problem to create TWIN (Muggli et al., 2015), OMBlast (Leung et al., 2017) and Maligner (Mendelowitz et al., 2016). OMBlast modifies the seed-and-extend approach used in BLAST (Altschul et al., 1990) for finding alignments in optical mapping data. Maligner provides two modes of alignment: an efficient, sensitive dynamic programming implementation that scales to large eukaryotic genomes, and a faster index based implementation for finding alignments with unmatched sites in the reference but not the query. TWIN (Muggli et al., 2015) uses an FM-index for aligning *in silico* digested contigs to a consensus optical map.

3 Background

Strings. Let S be a string $S[1]S[2] \dots S[n]$ of n symbols drawn from the alphabet [`A`, `C`, `G`, `T`, `N`]. The *suffix* of S is any string $S[i]S[i+1]..S[n]$ $1 \leq i \leq n$. Similarly, the *prefix* of S is any string $S[1]S[2]..S[j]$ $1 \leq j \leq n$.

Types of errors in Rmap data. We can view an Rmap $R = [r_1, r_2, \dots, r_n]$ as an ordered list of real numbers, such that each number represents the size of a fragment, i.e., the number of base-pairs between the cut-sites. Next, we define the size of an Rmap R as the number of fragments in R, and denote this as $|R|$. For example, given an enzyme that fragments DNA at the middle position of `AACT`, and the genomic sequence, `TTTTAACTGGGGGGGAACTTTTTTTTAACTTTTT` then the corresponding Rmap will be $R = [6, 11, 11, 6]$.

The main challenge in analyzing Rmap data is its error-profile. 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, an example of an additional cut site would be when the second fragment of R is split into two, e.g., $R' = [6, 5, 6, 11, 6]$, and an example of a missing cut site would be when the last two fragment of R are joined into a single fragment, e.g., $R' = [6, 11, 17]$. Lastly, an example of a sizing error would be if the size of the first fragment is estimated to be 7 rather than 6.

Error models. The sizing errors of an Rmap fragment depends on its length, i.e. the sizing error is a function of the fragment size based on some error model. Valouev et al. (2006a) proposed that the experimental size of an Rmap fragment follows a normal distribution with its true length

as mean and standard deviation proportional to its true length. That is, $o_i \sim N(r_i, \sigma^2 r_i)$ where o_i, r_i and σ are the observed experimental size of a fragment, true size of the fragment and the standard deviation respectively. This model was found to be inconsistent with data acquired from latest generation of optical map platforms, namely the Irys System of Bionano Genomics. Li et al. (2016) proposed a Laplace distribution function to model the sizing error and showed it to be more consistent with the current technology.

De Bruijn graphs. We denote a de Bruijn graph as $G = (V, E)$, where V is the set of nodes and E is the set of (directed) edges. Given a set of sequence reads $\{S_1, S_2, \dots\}$ and an integer k, we construct the de Bruijn graph by creating a directed edge for each unique k-mer in $\{S_1, S_2, \dots\}$, labeling the nodes of that edge as the $(k-1)$-length prefix, and the $(k-1)$-length suffix of that k-mer, and lastly, glue all nodes that have the same label. Figure 2 in the Supplement shows an example of a de Bruijn graph.

Restriction nodes. We recall that a restriction site is a short pre-defined sequence of nucleotides, which is recognized by a restriction enzyme. Throughout this paper, we assume that the length of any restriction site is less than k, where k is the integer used in the de Bruijn graph construction. We note that this is a practical assumption since typically restriction sites are between 6 bp and 8 bp in length and k is greater than 30. We call a node v in a de Bruijn graph G a *restriction node* if the first $|T|$ characters of the $(k-1)$-mer corresponding to v are equal to T. We denote the set of restriction nodes of V as V_T.

Paths and simple paths. We define a *path* p in a de Bruijn graph G as a list of nodes $v_1^p, .., v_n^p$, where $(v_i^p, v_{i+1}^p) \in E$ for all $i = 1, ..n-1$. We allow repeated nodes in a path. We define the length of a path as the number of edges in the path and denote it as $|p|$. We refer to the first and last node of a path as the source and destination node.

We call a path *simple* if it starts at a restriction node, ends at a restriction node, and does not contain any intermediate restriction nodes.

om-gram of an Rmap. Given a non-negative integer om, we define an *om-gram* as a sequence of om successive fragments of an Rmap. We note that $n - om + 1$ om-grams can be extracted from an Rmap of size n. For example, the following 3-grams can be extracted from R : (6, 11, 11) and (11, 11, 6). An om-gram is analogous to k-mer; yet we define a new term in order to avoid ambiguity with the definition of the de Bruijn graph.

4 Definition of Rmap-de Bruijn Graph Alignment

We formally define the Rmap-de Bruijn Graph (Rmap-DBG) alignment problem in this section. The input to the Rmap-DBG alignment problem is a de Bruijn graph $G = (V, E)$, and an Rmap $R = [r_1, .., r_n]$ generated using a restriction enzyme with recognizing restriction site T. The output of the problem is an alignment of R and G, which we define as a function $\varphi : R \rightarrow \{V_T\}$ which maps each r_i to a tuple of nodes in V_T. In other words, φ maps R to a path in G. Implicit in this definition is that $\varphi(r_i)$ can be equal to $\emptyset$, indicating that either end of r_i is a wrong additional cut-site, and that $\varphi(r_i)$ can contain more than two nodes in V_T, indicating that r_i contains a missing cut-site.

Next, we define the *induced Rmap* of R as $R' = [r_{\pi_1}, ..., r_{\pi_y}]$ where all r_i such that $\varphi(r_i) = \emptyset$ have been merged to a neighboring fragment r_j such that $\varphi(r_j) \neq \emptyset$. In addition, we denote a_i and m_i as the number of additional or missing cut-sites corresponding to $r_{\pi_i} \in R'$. We note that a_i equals the number of fragments merged into the i:th fragment of R' and $m_i = |\varphi(r_{\pi_i})| - 2$.

Therefore, the alignment function $\varphi()$ yields a sequence of tuples of restriction nodes $\{\varphi(r_{\pi_1}), .., \varphi(r_{\pi_y})\}$. these nodes. Let the set of paths in G that pass through each of the nodes in $\varphi(r_{\pi_k})$ be P_k. Therefore, for

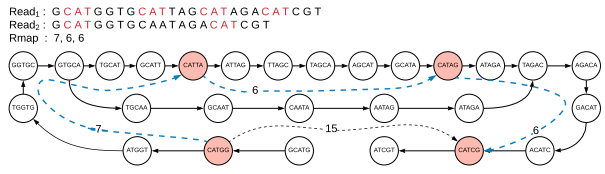


Fig. 1. Illustration of optical maps and de Bruijn graph alignment. The read sequence is in silico digested using an enzyme which nicks at restriction site CAT. First, all restriction nodes (starting with CAT) are located (colored in red), then for every restriction node as the source of a single path, using the dynamic programming algorithm, the destination restriction node(s) and length of the path(s) is found. The simple paths are shown as dotted connections between restriction nodes and the optimal path according to the de Bruijn graph is shown in blue.

each fragment of the induced Rmap (i.e. when $1 \leq k \leq y$), we define the score of an alignment between a path $p \in P_k$ and fragment r_{π_i} be proportional to the difference in the length of p and the size of fragment r_{π_i}. Further, we add penalties to the score for additional and missing cut sites in r_{π_i}. Formally the score $S(p, r_{\pi_k})$ is defined as:

$$S(p, r_{\pi_k}) = \big||p| - r_{\pi_k}\big| + \eta_a a_k + \eta_m m_k \tag{1}$$

where η_a and η_m are penalties for an additional and missing cut site, respectively. Also, in order to preserve the continuity of alignment, for $2 \leq k \leq y$ the first node in p has to be the same as the final node of $\varphi(r_{\pi_{k-1}})$ (i.e. the final node of the path corresponding to the previous fragment, $r_{\pi_{k-1}}$).

Then it follows that the score of an alignment $\varphi(R)$ is:

$$S(\varphi(R)) = \sum_{k=1}^{y} \min_{p \in P_k} S(p, r_{\pi_k})$$

.

Lastly, we define the optimal alignment of R and G as:

$$\arg\min_{\varphi}\{S(\varphi(R))\}$$

.

5 Methods

We present a solution for the Rmap-DBG alignment that follows a seed-and-extend paradigm. The input to the algorithm is a set of Rmaps $\{R_1, .., R_m\}$ and a de Bruijn graph $G = (V, E)$. The output is a path in G for each Rmap in $\{R_1, .., R_m\}$. The algorithm has four main steps: preprocessing, seeding, extending, and optimization. The seeding step finds partial alignments for a Rmap by finding paths in the graph for one or more of the om-grams in the Rmap. The extending step then aims to join these paths and find a single path corresponding to the alignment of the Rmap. Since it is possible that more than one path is discovered, the path that maximizes the alignment score (see Section 4) is returned. Algorithm 1 in the Supplement gives an overview of our approach.

5.1 Preprocessing

Error Correction of Rmaps. We error correct Rmaps $\{R_1, .., R_m\}$ using cOMet (Mukherjee et al., 2018), and extract all om-grams for a given value of om.

Find Restriction Nodes. We find and store all restriction nodes in the de Bruijn graph. Again, we denote the set of restriction nodes as V_T. We note that for the remainder of this section, the explanation of the alignment algorithm is restricted to the alignment of a single Rmap in $\{R_1, .., R_m\}$. We denote this Rmap as $R = [r_1, .., r_{|R|}]$.

Find and Store Simple Paths. Lastly, we find and store all simple paths in G that have length at most D using a dynamic programming algorithm which is similar to the gap filling algorithm by Salmela et al. (2016) and the algorithm for the exact path length problem by Nykänen and Ukkonen (2002).

For each restriction node v_s, we define a $|V| \times D$ binary matrix A_{v_s} where $\mathsf{A}_{v_s}(u,\ell) = 1$ if there exists a path from v_s to node u of length ℓ, and $\mathsf{A}_{v_s}(u,\ell) = 0$ otherwise. We first initialize A_{v_s} as follows: $\mathsf{A}_{v_s}(v_s,0) = 1$, $\mathsf{A}_{v_s}(u,0) = 0$ for all $u \in V - \{v_s\}$, $\mathsf{A}_{v_s}(u,1) = 1$ for all u such that there exists an edge between v_s and u, and lastly, $\mathsf{A}_{v_s}(u,1) = 0$ otherwise. Then we compute $\mathsf{A}_{v_s}(u,\ell)$ for all $u \in V$ and $2 \leq \ell \leq D$ based on the following recurrence:

$$\mathsf{A}(u,\ell) = \bigvee_{\{w \in V - V_T : (w,u) \in E\}} \mathsf{A}_{v_s}(w, \ell - 1).$$

In addition to A_{v_s}, we store an integer vector B_{v_s} of length $|V|$ for each restriction vertex v_s, where $\mathsf{B}_{v_s}[u]$ stores the length of the longest path from v_s to a node u in G. To compute B_{v_s}, we first initialize $\mathsf{B}_{v_s}[u] = 0$ for all $u \in V$ and then update $\mathsf{B}_{v_s}[u] = \ell$ when $\mathsf{A}_{v_s}(u,\ell) = 1$. After computing A_{v_s} and B_{v_s}, we use these matrices to identify and store all simple paths starting at v_s as follows. We iterate through each column of A_{v_s} and find the nodes reachable from v_s using a path of length ℓ (i.e. $\mathsf{A}_{v_s}(u,\ell) = 1$) which also satisfy the condition $|j - \mathsf{B}_{v_s}[u]| > 500$. We note that this second condition is required since the set of nodes reachable from v_s can grow extremely large if there exists directed cycles of length ≤ 500 bp in G. If u is in V_t then we store v_s, u, and ℓ in a list sorted by length, which we denote as L.

5.2 Seeding Step

Given a de Bruijn graph G, integer se, and an om-gram $[r_1, .., r_{om}]$, we define a *seed* of $[r_1, .., r_{om}]$ to be a path containing $om + 1$ restriction nodes, where the path in G between the x-th pair of restriction nodes has length at most $r_x \pm se$. We note that a seed of an om-gram is a simple path representing an alignment that can contain sizing error but no added or missing cut-sites, and se is the parameter that gives an upper bound on the sizing error. In this step, we aim to find a seed for the om-grams in R but note it is unlikely that there exists a seed for all om-grams due to the prevalence of added and/or missing cut-sites.

For finding seeds of an om-gram $[r_1, .., r_{om}]$ we initiate an om-level breadth first search in L, which was computed in the previous step. In the first iteration of this algorithm, we find all simple paths that have length in the range $[r_1 - se, r_1 + se]$. Next,we find all simple paths that have length in the range $[r_2 - se, r_2 + se]$ and whose source node is the destination node of the simple paths found in the previous iteration. We repeat this step om times, and store a seed for each sequence of om simple paths whose lengths satisfy the bounds on the sizing error. For each seed, we store the source and destination nodes of each om simple path. Lastly, we consider each om-gram in R for which there exists at least one seed; if there exists more than one seed then we choose the seed that minimizes the sizing error, breaking ties arbitrarily.

5.3 Extending Step

We assume R has a seed for two or more of its non-overlapping om-grams since otherwise we halt the algorithm and return that we cannot find an alignment for R. The following extending and optimizing steps are repeated for each pair of consecutive seeded om-grams. We let $[r_i, \ldots, r_{i+om-1}]$ and $[r_j, \ldots, r_{j+om-1}]$ be two such consecutive om-grams, and $\varphi([r_i, \ldots, r_{i+om-1}])$ and $\varphi([r_j, \ldots, r_{j+om-1}])$ be the respective seeds found in the previous step. It follows that $R^e = [r_{i+om}, \ldots, r_{j-1}]$ is the portion of R between that is unaligned. In this step, we find all sequences of simple paths $\{sp_1, \ldots, sp_n\}$ where

$$\varphi([r_i, \ldots, r_{i+om-1}]) \cup sp_1 \cup .. \cup sp_n \cup \varphi([r_j, \ldots, r_{j+om-1}]$$

is a path in G. We note that we restrict interest to sequences of simple paths where: (1) the total length is bounded by $|t \pm se|$, where t be the sum of all the fragment lengths in R^e, i.e. $t = r_{i+om} + .. + r_{j-1}$; and (2) the number of simple paths is at most $2|R^e|$[1]. The first constraint accounts for possible sizing error, and the latter constraint implies the fragments of R^e can be aligned to one or more simple paths in G, which enables the consideration of added or missing cut-sites.

Next, we describe an iterative algorithm for finding all sequences of simple paths that satisfy these constraints. Our algorithm can be seen as a version of breadth first search with added constraints. We let v_s to be the last node of $\varphi([r_i, \ldots, r_{i+om-1}])$ and v_d be the first node of $\varphi([r_j, \ldots, r_{j+om-1}])$. Hence, the first step of this algorithm is to initialize the set of all sequences of such simple paths, denoted as $\mathcal{P}$, to be the empty set, and initialize the set of source nodes, denoted as V_s, to be equal to $\{v_s\}$. At each iteration of the algorithm, we find all simple paths that start with a node v that is in V_s such that $v \neq v_d$. For each such simple path sp, we determine if the addition to the sequence of simple paths in $\mathcal{P}$ that end at v is such that the total length is at most $t + se$. If this is true then we add the corresponding set of sequences of simple paths to $\mathcal{P}$ and update V_s to contain the destination node of sp. After $2|R^e|$ iterations, we consider each sequence in $\mathcal{P}$ and eliminate any sequence where the destination node is not equal to v_d and the length is less than $t - se$. Figure 3 in the Supplement illustrates this algorithm.

Upon termination of this algorithm, $\mathcal{P}$ contains all possible sequences of simple paths that connects $[r_i, \ldots, r_{i+om-1}]$ and $[r_j, \ldots, r_{j+om-1}]$.

5.4 Optimizing Step

Lastly, we choose an optimal sequence of extending paths for each pair of consecutive seeded om-grams. To do this we use a dynamic programming algorithm — similar to Valouev et al. (2006a) and Nagarajan et al. (2008) — which minimizes the alignment score. For each $P \in \mathcal{P}$, we compute the optimal score of aligning P with $R^e = [r_1, .., r_{|R^e|}]$. For this alignment we use a DP matrix C, whose dimensions are $|P|$ by $|R^e|$. The block $\mathsf{C}[x][y]$ contains the optimal score of aligning the first x simple paths of P with the first y fragments of R^e. The process of filling up C is explained in details in the Supplement section 1.1. After completing the DP matrix, the optimal alignment score is found at $\mathsf{C}[|P|][|R^e|]$.

After computing the optimal score for each $P \in \mathcal{P}$ we return the set of simple paths that achieves the best score overall.

5.5 Computational Complexity

In this section we discuss the computational complexity of aligning one Rmap containing $|R|$ fragments to the de Bruijn graph with $|V|$ nodes and $|E|$ edges and $|V_T|$ restriction nodes. Further discussion on the complexity analysis is included in Supplement section 1.2.

For each source restriction node, finding all simple paths, for given value of D is solved in time $O(|E|D \log|V_T|)$ using the dynamic programming formulation described in Section 5.2. Therefore, the total time for finding simple paths is $O(|V_T||E|D \log|V_T|)$. This time complexity is pseudopolynomial in D. However, assuming $D = O(|E|)$ the complexity is polynomial in the input size. After finding simple paths, the complexity of finding seeds is given by $O(om^2 \log(n_{sp}))$ where om is the value of om-gram used and n_{sp} denotes the number of simple paths

[1] The upper bound of $2|R^e|$ is a practical assumption. Considering the average digestion rate of an Rmap fragment is 0.8 Li et al. (2016), the probability of exceeding this bound is less than 0.016% when $|R^e| \geq 3$.

found. This comes from the fact that for each om-gram we do an om-level search in the simple path table which is sorted according to path paths.

The extension step has two parts. The complexity of finding sets of simple paths which joins seeds of the Rmap is $O(n_{sp}\log(n_{sp}))$ for each pair of seeds, since at each simple path we look up the next one in $\log(n_{sp})$ due to the binary search in sorted list of simple paths, and there are at most n_{sp} simple paths. The second step of extension is evaluating the cost of aligning each connecting path against the unaligned fragments of the Rmap. For each connecting path containing n simple paths and aligned to m Rmap fragments, the complexity of alignment is $O(mn)$. Now, n is bounded by $2m$ and m is bounded by the total number of Rmap fragments. Therefore the complexity of alignment is $O(|R|^2)$. The overall complexity of our method is dominated by finding simple paths step, hence the overall complexity is $O(|V_T||E|D\log|V_T|)$.

6 Experiments

6.1 Datasets

We ran experiments on the following two datasets of optical maps: (1) a simulated Rmap dataset generated using *E. coli* K-12 substr. MG 1655, and (2) an Rmap dataset generated for the human genome using the Bionano platform. In the Supplement section 1.3, we explain each dataset in more detail.

All experiments were ran on Intel E5-2698v3 processors with 192 GB of RAM running 64-bit Linux. The wall time was determined by running the alignment in parallel on 500 CPUs. The CPU time and wall time are reported in hours (h), minutes (m), and seconds (s). The peak memory usage is reported in gigabytes (GB) and megabytes (MB).

6.2 Experiments on Simulated Data

We constructed the de Bruijn graph by error correcting the sequence reads using the error correction module in SPAdes (Bankevich et al., 2012), extracting all k-mers from the resulting reads using the KMC tool (Deorowicz et al., 2014), and constructing the graph from the resulting k-mers using the method of Bowe et al. (2012).

6.2.1 Evaluation of the Effect of the Maximum Length

The value of D puts an upper bound on the maximum fragment size that can be aligned to the de Bruijn graph. Table 4 in the Supplement demonstrates the impact of D on the CPU time, wall time, memory usage, and number of simple paths found in the seeding step. Figures 6 and 7 (in the Supplement) illustrate the distribution of fragment sizes in the *E. coli* and human optical map datasets, respectively. As seen in the figure, when $D = 70,000$ then over 99% of the *E. coli* fragments have size at most D.

Higher values of D lead to an overall increase in the CPU time, wall time and memory usage. Choosing D to be equal to the size of the largest fragment, is inefficient since the largest fragment in an optical map dataset is usually much larger than the value of D which covers 99% of the fragments. For instance, $D = 70,000$ covers 99.22% of the Rmap fragments and the running time is 4.11 hrs whereas $D = 100,000$ covers 100% of the Rmap fragments but the running time increases by 88.80% to 7.76 hrs. Therefore we set $D = 70,000$ for the *E. coli* dataset.

6.2.2 Evaluation of the Effect of Sizing Error

We considered various values of the maximum sizing error se and determined the percentage of om-grams in which at least one seed (path) was identified and also noted the running times for each value of se. We considered five values for se, 100, 250, 500, 750 and 1000 where $k = 64$, $D = 50,000$ and om varied from 3 to 6. Table 2 illustrates the results of these experiments. We see that as se increases, a greater percentage of the seeds were found for each om-gram with the highest performance witnessed when $om = 3$. We also consider that increasing se increases the running time since with higher se, more seeds are discovered (many of which are erroneous) which results in more processing in the extension step. The running times (in hours) for the different se settings for $om = 3$ are as follows: 2.7, 3.12, 4.18, 6.28 and 10.21 for se, 100, 250, 500, 750 and 1000 respectively. We also note that while a higher percentage of seeded om-grams is more desirable than a very low percentage, a small increase in the number of seeded om-grams do not necessarily translate to better overall alignment results. Therefore, we set $se = 500$ since then seeds were identified for close to 72% of the om-grams. For higher values of se there is a negligible increase in the percentage of seeded om-grams but they incur a significance increase in the running time. For $se = 750$ and $se = 1000$, the running times increase by 50.24% and 144.25% respectively compared to $se = 500$

	k=31	**k=43**	**k=55**	**k=63**
Percentage of seeded 3-grams	100%	96%	76%	72%
% of Rmaps with ≥ 2 seeded 3-grams	100%	100%	86%	81%
Mean Rmap overlay	100%	94%	92%	90%
Percentage of seeded 4-grams	100%	80%	36%	40%
% of Rmaps with ≥ 2 seeded 4-grams	100%	100%	83%	81%
Mean Rmap overlay	100%	93%	81%	80%
Percentage of seeded 5-grams	100%	69%	48%	26%
% of Rmaps with ≥ 2 seeded 5-grams	100%	100%	80%	80%
Mean Rmap overlay	100%	93%	70%	73%

Table 1. The performance of the seeding step when om and k were varied. Rmap overlay is the percent of an Rmap that is contained in between the two furthest seeded om-grams of that Rmap. We describe it in details in Section 6.2.3. We define the mean Rmap overlay as the average Rmap overlay of all Rmaps in the dataset.

om	$se = 100$	$se = 250$	$se = 500$	$se = 750$	$se = 1000$
3	20.35%	46.98%	71.94%	73.27%	73.84%
4	9.48%	24.87%	39.07%	43.54%	44.63%
5	4.91%	15.87%	24.61%	25.72%	25.81%
6	2.83%	10.98%	17.64%	19.43%	20.01%

Table 2. The percentage of seeded om-grams for different values of om and se. In this experiment, k and D were kept constant at 64 and 70,000, respectively.

6.2.3 Evaluation of the Effect of k and om

The selection of om-gram size is based on the frequency of additional cut site and missing cut site errors in the Rmap data. For an om-gram seed to be discovered, a necessary criteria is that the om consecutive fragments of the Rmaps that do not contain an additional or missing cut-site. We performed experiments on the following values of k-mers : 31, 43, 55 and 63. Table 1 shows the results of seeding step on these values using om-gram sizes 3, 4 and 5.

For an Rmap, we define the *Rmap overlay* as the maximum number of Rmap fragments which can be aligned with the seed and extend method, i.e. the number of Rmap fragments between the two furthest seeded om-grams plus $(2 \times om)$ (to account for the fragments of the seeded om-grams). This value is expressed as a percentage of the total number of fragments of the Rmap. For example consider Figure 5 in the Supplement. The 3-grams $\{13, 4, 6\}$ and $\{8, 17, 4\}$ are seeded, therefore the Rmap overlay is $(6 + 2 \times 3)/15$ which equals to 80%. In Table 1, we report the percent of seeded om-grams, the percent of Rmaps with at least 2 seeded om-grams and the mean Rmap overlay for each value of k.

For $k = 31$, the de Bruijn graph is very dense, hence a large number of simple paths are found — which results in extensive seeding. Seeds

	No. of Aligned Rmaps	Cumulative % of Aligned Rmaps	CPU Time	Peak Memory	Wall time
$k = 63$	803 (32.05%)	32.05%	5.89 h	1.3 GB	24.21 m
$k = 55$	680 (27.14%)	59.19%	15.60 h	1.87 GB	68.49 m
$k = 43$	241 (9.6%)	68.79%	189.21 h	2.22 GB	103.04 m
$k = 31$	101 (4.03%)	72.82%	296.52 h	2.56 GB	126.27 m
$k = 63$	511,613	64.5%	426.11 h	10.86 GB	43.68 h
$k = 55$	92,011	76.2%	412.64 h	14.70 GB	41.91 h

Table 3. The alignment results, the CPU time, and wall time with default parameter settings for **Top:** the simulated Rmaps and **Bottom :** human optical map data. The de Bruijn graphs are built using different values of k, i.e., 63, 55, 43 and 31. We first aligned all Rmaps to the de Bruijn graph constructed for $k = 63$. Then all Rmaps that did not align were aligned to the de Bruijn graph built on $k = 55$. In case of simulated data, we repeated this process for $k = 43$ and $k = 31$. We define the cumulative % of aligned Rmaps as the percentage of Rmaps that aligned to one of the de Bruijn graphs.

are discovered for every om-gram — even for ones with additional or missing cut sites (which are obviously erroneous and misleading). This is not desirable and ultimately leads to poor alignment and longer run time. Hence, the figures for $k = 31$ are not very informative with respect to choosing an optimal value for om. Considering the remaining results, we find $om = 3$ gives the best result on seeding since it achieves the highest percentage of seeded om-grams and mean Rmap overlay across all graphs. With om-gram sizes 4 and 5, the percentage of seeded om-grams drops significantly in higher order de Bruijn graphs.

6.2.4 Performance with Default Settings

Based on the experiments above, we determined the following default values for the input parameters: $se = 500$ and $om = 3$. In our dataset, the maximum sized fragment is 91.35 kbp long. However, we fix $D = 70,000$ which covers 99% of fragment sizes (as explained in Section 6.2.1). We constructed the de Bruijn graph for the following values of k: 31, 43, 55, and 63. We first aligned all Rmaps to the graph constructed for $k = 63$, and for those that did not align we then attempted to align them to the graph constructed for $k = 55$. We continue this procedure, lowering the value of k at each iteration. We deem any Rmap as unaligned if it did not align to one of these graphs.

Table 3 summarizes the results of these experiments. We aligned 803 (32.05%) of the Rmaps to the de Bruijn graph constructed with $k = 63$. Another 680 (27.14%), 241 (9.6%) and 101 (4.03%) Rmaps were aligned to de Bruijn graph constructed with $k = 55$, $k = 43$ and $k = 31$ respectively. Therefore, we aligned 72.82% of the Rmaps in total. We could not align the remaining 27.18% of Rmaps to any of the de Bruijn graphs. We performed the following experiment to investigate the relative quality of the Rmaps which were aligned versus those which were not. First, we performed *in silico* digestion of the reference genome using the enzyme RsrII which produces the reference optical map of the genome. Then we use the dynamic aligner from Valouev et al. (2006a), to align each Rmap in the dataset with the reference optical map. Each alignment produced an alignment score called the S-score which denotes the quality of alignment. Since the reference optical map is error-free, the S-score gives a measure of the quality of the Rmap. That is, the S-score is higher if the Rmap has fewer errors. The S-score is proportional to the number of fragments in the Rmaps. Therefore for each Rmap, we *normalize* the S-score by dividing by the number of Rmap fragments. Figure 8 in the Supplement show the distribution of the normalized S-scores. We see from the graph that the Rmaps which could not be aligned have a lower distribution of S-scores. The mean and standard deviation of the normalized S-score for the aligned Rmaps is 4.2 and 0.71 respectively whereas for the unaligned Rmaps it is 2.9 and 0.56, respectively.

6.3 Validation of Alignments

To validate our alignments, we simulated the Rmap data and stored the location in the reference genome from where the Rmap originated. In particular, we kept the start and end positions of the Rmap in the reference genome of each simulated Rmap. Next, we constructed the de Bruijn graph on the sequence reads and aligned the Rmaps to the resulting graph using the default parameters. Then for each Rmap, we considered the path in the de Bruijn graph to which the Rmap aligned, and constructed the corresponding contig by traversing the path and constructing the respective (DNA) sequence. We then aligned the resulting contig to the reference genome of *E. coli* K-12 substr. MG 1655 using BLAT (Kent, 2002). The start and end positions of this alignment in the reference genome are compared with the known start and end positions of the Rmap in the same reference genome. This comparison was completed for all Rmaps. Using our method, we found that 99.6 % of the Rmap alignments (1,818 out of the 1,825 total number of alignments) corresponded to the known alignments, meaning the location of the Rmap in the genome intersected with the location of the sequence corresponding to the aligned nodes in the de Bruijn graph. This demonstrates the accuracy of our method in finding correct alignments.

6.4 Branching resolution by alignment

We perform and report on the following experiment to find out how effective an optical map - de Bruijn graph alignment is for resolving branching during traversal of the de Bruijn graph for sequence assembly. For each simple path found in each de Bruijn graph, we evaluate if the path contains any branching along its way i.e. if at least one of its nodes has an out degree greater than one. The percentage of branching simple-paths in the graphs built on k= 63, 55, 43 and 31 are 86.3%, 89.6%, 91.3% and 92.7% respectively. Since any successful alignment contains a minimum of two seeds i.e. $\geq 2 \times om$ number of simple paths, all optical map - de Bruijn graph alignments contain one or more branching nodes and therefore can be used to resolve them.

6.5 Performance on Human Data

For this experiment, we built de Bruijn graphs on k-mer sizes 63 and 55 and used the default parameters. The maximum sized fragment in this dataset is 240 kbp but based on the distribution of fragment sizes, we fix $D = 70,000$ which covers 99.9% of fragment sizes. Figure 7 in the Supplement show the distribution of fragment sizes of this dataset. Since the de Bruijn graph is built from the reference genome, the graph is connected even with higher values of k-mer size. Furthermore for a complex and large genome such as the human genome, the de Bruijn graph will get tangled for small values of k. Therefore, we align the Rmaps to the graphs built on $k = 63$ and 55 and do not use lower order de Bruijn graphs. Table 3 summarizes the results of these experiments.

In order to contextualize these results, we align all the Rmaps to an optical map generated from the human reference genome. To accomplish this, we first in silico digest the human reference genome (GenBank assembly accession: GCA_000001405.15) using BspQI, and align each Rmap using the method of Valouev et al. (2006a) with default parameters. We found that 98.4% of the Rmaps aligned to the simulated optical map. Figure 9 in

the Supplement shows the distribution of alignment scores of these alignments. These numbers validate the percentage of Rmaps that align to the de Bruijn graph.

7 Conclusion

We define and solve a new problem: aligning optical maps to de Bruijn graphs constructed on sequence data of the same genome. We described a seed-and-extend algorithm for solving the problem of aligning optical maps to de Bruijn graphs, and aligned 72.82% of Rmaps simulated from the *E. coli* genome to the de Bruijn graph constructed from short reads generated from the same genome. For each aligned Rmap, we validate the correctness of the alignment by comparing the location of the Rmap in the genome with the location of the aligned sequence in the same genome. For 99.6% of the aligned Rmaps, the locations intersected with each other which shows the high accuracy of our method. We also show our method is capable of scaling to larger genomes. In particular, we show that 76.2% Rmaps can be aligned to the de Bruijn graph in the case of human data.

Funding

KM, BA, LS, and CB were funded by the National Science Foundation (1618814) and LS was funded by Academy of Finland (grants 308030, and 314170).

References

Altschul, S. et al. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

Bankevich, A. et al. (2012). SPAdes: a new Genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477.

Beier, S. et al. (2017). Construction of a map-based reference genome sequence for barley, hordeum vulgare l. *Scientific Data*, 4:170044–170044.

Boisvert, S., Raymond, F., Godzaridis, E., Laviolette, F., and Corbeil, J. (2012). Ray Meta: Scalable De Novo Metagenome Assembly and Profiling. *Genome Biol.*, 13(12):R122+.

Bowe, A., Onodera, T., Sadakane, K., and Shibuya, T. (2012). Succinct de bruijn graphs. In *Proc of WABI*, pages 225–235.

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

Butler, J. et al. (2008). ALLPATHS: De novo Assembly of Whole-Genome Shotgun Microreads. *Genome Research*, 18(5):810–820.

Daccord, N. et al. (2017). High-quality de novo assembly of the apple genome and methylome dynamics of early fruit development. *Nature Genetics*, 49:1099–1106.

Deorowicz, S., Kokot, M., Grabowski, S., and Debudaj-Grabysz, A. (2014). KMC 2: Fast and resource-frugal k-mer counting. *CoRR*.

Dimalanta, E. et al. (2004). A microfluidic system for large dna molecule arrays. *Analytical Chemistry*, 76(18):5293–5301.

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

Ganapathy, G. et al. (2014). *De novo* high-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience*, 3(1):1–9.

Jarvis, D. E. et al. (2017). The genome of chenopodium quinoa. *Nature*, 542:307 EP –.

Kent, J. (2002). BLAT–The BLAST-Like Alignment Tool. *Genome Research*, 12(4):656–664.

Leung, A. et al. (2017). Omblast: alignment tool for optical mapping using a seed-and-extend approach. *Bioinformatics*, 33(3):311–319.

Li, M. et al. (2016). Towards a more accurate error model for bionano optical maps. In *Proc of ISBRA*, pages 67–79.

Lin, H. et al. (2012). Agora: assembly guided by optical restriction alignment. *BMC Bioinformatics*, 13(1):189.

Lin, J. et al. (1999). Whole-genome shotgun optical mapping of deinococcus radiodurans. *Science*, 285(5433):1558–1562.

Mendelowitz, L. et al. (2016). Maligner: a fast ordered restriction map aligner. *Bioinformatics*, 32(7):1016–1022.

Muggli, M., Puglisi, S., and Boucher, C. (2014). Efficient indexed alignment of contigs to optical maps. In *Proc of WABI*, pages 68–81.

Muggli, M., Puglisi, S., Ronen, R., and Boucher, C. (2015). Misassembly detection using paired-end sequence reads and optical mapping data. *Bioinformatics*, 31(12):i80–i88.

Muggli, M. D., Puglisi, S. J., and Boucher, C. (2018). A Succinct Solution to Rmap Alignment. In *WABI 2018*, volume 113, pages 12:1–12:16.

Mukherjee, K., Washimkar, D., Muggli, M., Salmela, L., and Boucher, C. (2018). Error correcting optical mapping data. *GigaScience*, 7(6):giy061.

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

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

Neely, R. K., Deen, J., and Hofkens, J. (2011). Optical mapping of DNA: single-molecule-based methods for mapping genome. *Biopolymers*, 95(5):298–311.

Nykänen, M. and Ukkonen, E. (2002). The exact path length problem. *J. Algorithms*, 42(1):41–53.

Pan, W. et al. (2018). Novo & stitch: accurate reconciliation of genome assemblies via optical maps. *Bioinformatics*, 34(13):i43–i51.

Salmela, L., Sahlin, K., Mäkinen, V., and Tomescu, A. (2016). Gap filling as exact path length problem. *J. Comp. Biol.*, 23(5):347–361.

Samad, A., Huff, E., Cai, W., and Schwartz, D. (1995). Optical mapping: a novel, single-molecule approach to genomic analysis. *Genome Research*, 5:1–4.

Schwartz, D. et al. (1993). Ordered restriction maps of *saccharomyces cerevisiae* chromosomes constructed by optical mapping. *Science*, 262:110–114.

Shelton, J. M. et al. (2015). Tools and pipelines for bionano data: molecule assembly pipeline and fasta super scaffolding tool. *BMC Genomics*, 16(1):734.

Shi, L. et al. (2016). Long-read sequencing and de novo assembly of a chinese genome. *Nature Communications*, 7.

Simpson, J. et al. (2009). ABySS: A Parallel Assembler for Short Read Sequence Data. *Genome Research*, 19(6):1117–1123.

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

Valouev, A. et al. (2006a). Alignment of optical maps. *Journal of Computational Biology*, 13(2):442–462.

Valouev, A., Schwartz, D., Zhou, S., and Waterman, M. (2006b). An algorithm for assembly of ordered restriction maps from single dna molecules. *Proceedings of the National Academy of Sciences*, 103(43):15770–15775.

Vij, S. et al. (2016). Chromosomal-level assembly of the asian seabass genome using long sequence reads and multi-layered scaffolding. *PLoS Genet*, 12(4):e1005954–e1005954.

Zerbino, D. and Birney, E. (2008). Velvet: Algorithms for De Novo Short Read Assembly using De Bruijn Graphs. *Genome Research*, 18(5):821–829.

1 Supplement

1.1 Optimizing step

In the optimizing step, for each $P \in \mathcal{P}$, such that $P = \{sp_1, .., sp_{|P|}\}$, we compute the optimal score of aligning P with $R^e = [r_1, .., r_{|R^e|}]$ as follows. We let C be a $|P|$ by $|R^e|$ matrix, where $\mathsf{C}[x][y]$ is the optimal score of aligning the first x simple paths of P with the first y fragments of R^e. We initialize $\mathsf{C}[0][y]$ and $\mathsf{C}[x][0]$ to be equal to ∞ for all $x = 1, .., |P|$ and $y = 1, .., |R^e|$, and $\mathsf{C}[0][0] = 0$. Then, we let $\mathsf{C}[x][y]$ to be equal to the minimum of:

- $\mathsf{C}[x-1][y-1] + ||sp_x| - r_y|$ (in this particular case we do not have any added or missing cut sites),
- $\mathsf{C}[x-2][y-1] + ||sp_{x-1}| + |sp_x| - r_y| + \eta_a$ or $\mathsf{C}[x-3][y-1] + ||sp_{x-2}| + |sp_{x-1}| + |sp_x| - r_y| + 2\eta_a$ (in this particular case we have one or two added cut-sites, respectively),
- $\mathsf{C}[x-1][y-2] + ||sp_x| - (r_{y-1} + r_y)| + \eta_m$ or $\mathsf{C}[x-1][y-3] + ||sp_x| - (r_{y-2} + r_{y-1} + r_y)| + 2\eta_m$, (in this particular case we have one or two missed cut sites, respectively),
- $\mathsf{C}[x-2][y-2] + ||sp_{x-1}| + |sp_x| - (r_{y-1} + r_y)| + \eta_a + \eta_m$ or $\mathsf{C}[x-3][y-2] + ||sp_{x-2}| + |sp_{x-1}| + |sp_x| - (r_{y-1} + r_y)| + 2\eta_a + \eta_m$ (in this particular case we have one additional cut-site and one or two missed cut-sites, respectively),
- $\mathsf{C}[x-2][y-3] + ||sp_{x-1}| + |sp_x| - (r_{y-2} + r_{y-1} + r_y)| + \eta_m + 2\eta_a$ or $\mathsf{C}[x-3][y-3] + ||sp_{x-2}| + |sp_{x-1}| + |sp_x| - (r_{y-2} + r_{y-1} + r_y)| + 2\eta_m + 2\eta_a$ (in this particular case we have two additional cut-sites and one or two missed cut-sites, respectively),

for all $0 \leq x \leq |P|$ and $0 \leq y \leq |R^e|$. We note that η_a and η_m are defined in section 4. The optimal alignment score for P and R^e is found at $\mathsf{C}[|P|][|R^e|]$. In addition to the scoring matrix C, we also store a matrix that allows us to obtain the actual alignment of P to R^e.

1.2 More on complexity

We note that finding simple paths between restriction nodes closely resembles the gap filling problem (Salmela et al., 2016) and the exact path length problem (Nykänen and Ukkonen, 2002). Both of these problems have been shown to be NP-complete. However, the NP-completeness proofs for these problems are for a more general case of weighted graphs and the weights are essential in keeping the reduction polynomial. These proofs are thus not directly applicable to our problem because de Bruijn graphs are not weighted. Furthermore, algorithms pseudopolynomial in the path length, which corresponds to our maximum distance D, exist for both of these problems. Therefore if the path length is assumed to be polynomial in the size of the graph, then the gap filling problem and the exact path length problem in fact have polynomial time solutions.

1.3 Datasets

We simulated the Rmap data from the *E. coli* reference genome as follows. First, we made 200x copies of the reference genome, and then selected loci uniformly at random for each of these copies—these loci form the ends of single molecules that would undergo *in silico* digestion. These loci represent the start and end locations of the simulated Rmaps and are later used to validate the alignments. We discarded molecules that were smaller than 150 Kbp, and identified the restriction sites for the RsrII enzyme in the remaining molecules. We note that we used this error-free Rmap data for validation. Lastly, we added errors into the Rmaps according to the error model given in Li et al. (2016). We refer the reader to Section 3 for a description of this model. This simulation resulted in 2,505 Rmaps, containing 7,485 missing cut-sites and 554 additional cut-site. We built the de Bruijn graph dataset consisting of approximately 27 million paired-end 100 bp reads from *E. coli* (substr. K-12) (ERA000206, EMBL-EBI Sequence Read Archive).

The second dataset consists of 793,199 Rmaps generated from the Irys Bionano platform for the human genome using the enzyme BspQI. We obtained this dataset from the *de novo* assembly of a Chinese genome by Shi et al. (2016). We built the de Bruijn graph from the human reference genome (GenBank assembly accession: GCA_000001405.15, Genome Reference Consortium Human Build 38).

D	**Rmap fragments covered**	**CPU time**	**Memory**	**Wall time**	**No. of simple paths found**
5,000	35.51%	12.41 m	449 MB	12 s	338
10,000	57.70%	16.85 m	450 MB	30 s	1,214
25,000	87.88%	49.33 m	510 MB	3.53 m	8,152
50,000	98.17%	2.61 h	629 MB	8.66 m	29,709
70,000	99.22%	4.11 h	755 MB	13.29 m	53,199
100,000	100%	7.76 h	959 MB	21.38 m	89,852

Table 4. Impact of D on finding and storing simple paths. The graph is built with $k = 63$ on $E.coli$ short read data, and consists of 12,478,516 nodes and 12,560,111 edges. There are 777 restriction nodes. The percent of Rmap fragments from the simulated optical map dataset whose size is less than or equal to D is referred to as Rmap fragments covered.

Algorithm 1 Seed-and-extend Rmap-DBG alignment

% Input: a set of Rmaps $\{R_1, .., R_m\}$ and a de Bruijn graph $G = (V, E)$.
% Output: a path in G for each Rmap in $\{R_1, .., R_m\}$.
Preprocessing Step:
Error correct the Rmaps $\{R_1, .., R_m\}$.
Find and store the restriction nodes V_T in G.
Initialize L to be an empty list.
For each v_s in V_T **do**
1. Find all simple paths starting at v_s, and ending at v_t, where $v_t \in V_T$.
2. Store each simple path in L.
For each R in $\{R_1, .., R_m\}$ **do**
Seeding Step:
For each om-gram o in R **do**
1. Find and store each seed for o;
2. If more than one seed exists, store the one that minimizes the sizing error.
Extending Step:
If R has less than two seeds **then**
Return: No alignment for R.
Else
For each pair of consecutive seeds:**do**
1. Use L to find and store sequences of simple paths that create a single path connecting the seeds found for R;
2. Store each sequence of simple paths in $\mathcal{P}$
3. Add the sequence of P that optimizes the alignment score to the final alignment.
Return final alignment of R

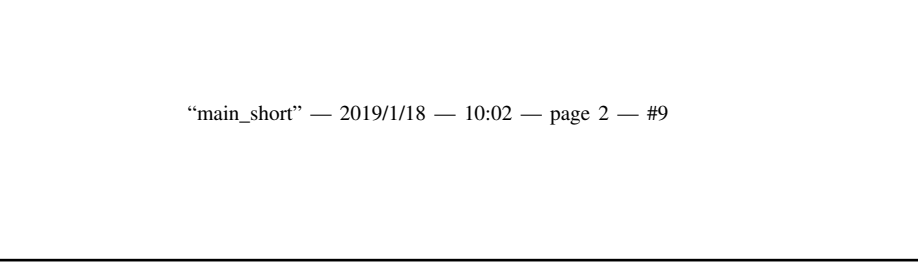


Fig. 2. Illustration of a de Bruijn Graph build from two sequences S_1 and S_2 with $k = 5$. (b) Set of all distinctive k-mers from S_1 and S_2. (c) Each k-mer becomes an edge in the graph which connects two nodes whose labels are the $(k-1)$-length prefix and the $(k-1)$-length suffix of the edge label. The nodes with same labels have the same color. (d) The completed de Bruijn graph after merging of nodes with the same label.

	No. of Aligned Rmaps	**Cumulative % of Aligned Rmaps**	**CPU Time**	**Peak Memory**	**Wall time**
$k = 63$	506,485	63.8%	429.29 h	10.95 GB	44.03 h
$k = 55$	93,724	75.6%	418.77 h	14.81 GB	42.75 h

Table 5. The alignment results, the CPU time, and wall time for uncorrected human optical map data with default parameter settings (Table 3 presented the results for error corrected human optical maps). The de Bruijn graphs are built using two different values of k, i.e., 63 and 55. Again, we aligned all Rmaps to the de Bruijn graph constructed for $k = 63$ and then aligned the remaining Rmaps (ones which did not align) to the de Bruijn graph built on $k = 55$. We define the cumulative % of aligned Rmaps as the percentage of Rmaps that aligned to one of the graphs currently considered.

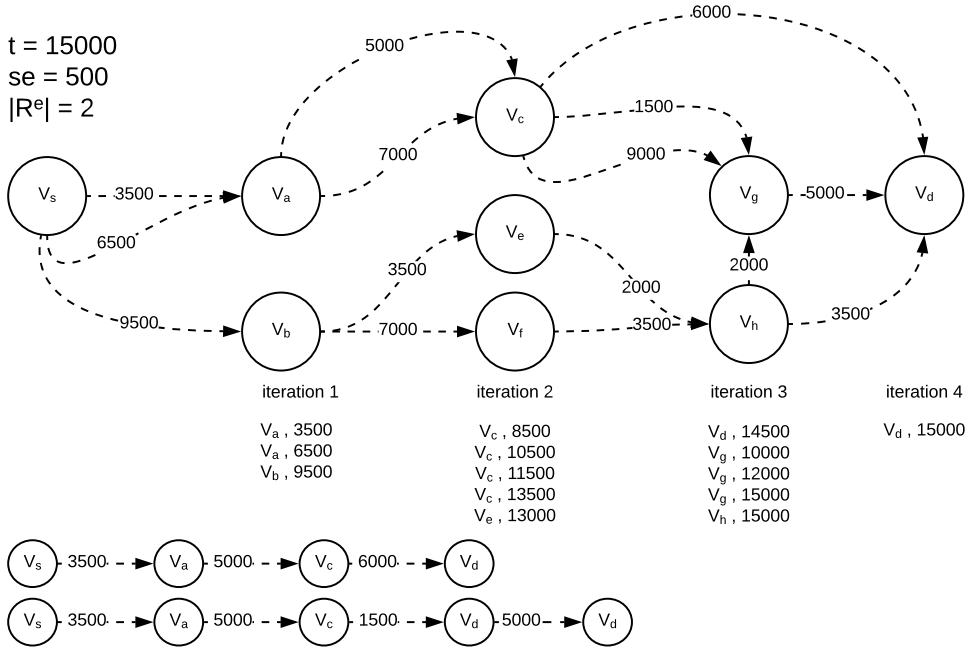
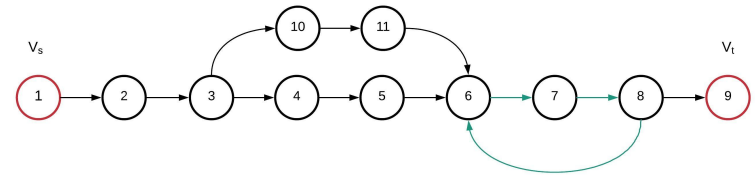


Fig. 3. Illustration of the extension algorithm. The dotted arrows represent simple paths in the graph. The extension step finds paths that align to the segment of an Rmap in between two seeded om-grams. t denotes the total length of the segment of the Rmap, $|R^e|$ represents the number of fragments in the Rmap segment and se represents the sizing error allowance. Nodes v_s and v_d are the source and destination nodes of the extension. We find paths whose total length is $t \pm se$ and has at most $2|R^e|$ number of simple paths. The final extension paths which satisfy the conditions are shown below.



	0		1		2		3		4		5		6		7		8	
V1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V2	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
V3	0	0	0	0	1	2	0	2	0	2	0	2	0	2	0	2	0	2
V4	0	0	0	0	0	0	1	3	0	3	0	3	0	3	0	3	0	3
V5	0	0	0	0	0	0	0	0	1	4	0	4	0	4	0	4	0	4
V6	0	0	0	0	0	0	0	0	0	0	1	5	0	5	0	5	0	5
V7	0	0	0	0	0	0	0	0	0	0	0	0	1	6	0	6	0	6
V8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	7	0	7
V9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8
V10	0	0	0	0	0	0	1	3	0	3	0	3	0	3	0	3	0	3
V11	0	0	0	0	0	0	0	0	1	4	0	4	0	4	0	4	0	4

Fig. 4. Top: Example of a cycle in a de Bruijn graph — illustrated in green. The restriction nodes, v_s and v_t are shown in red. Without any refactoring of cycles, there are two simple paths of length 8 from v_s to v_t and an infinite number of additional simple paths of lengths 11,14,17 …and so on (because of the cycle). However, after refactoring, we get a single simple path of length 8. **Bottom:** Illustration of finding simple paths using the binary matrix A_{v_s} (the white cells of the table) and integer vector B_{v_s} (shown in gray). Note that at each iteration, B_{v_s} is updated and the previous states are not stored.

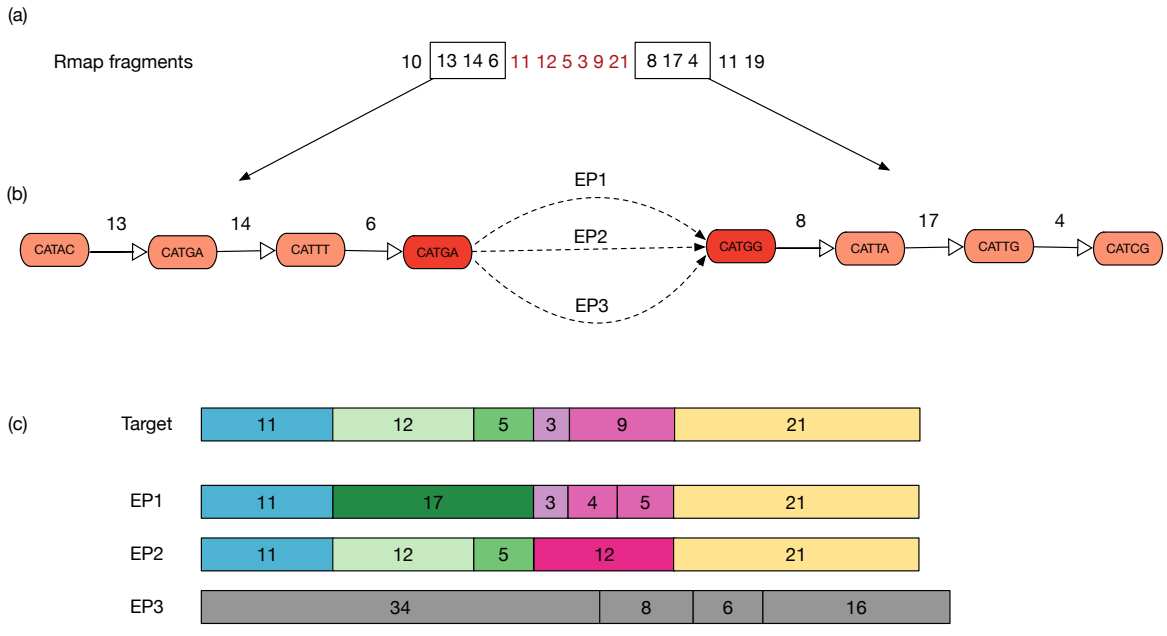


Fig. 5. An illustration of the extension and optimization steps. The read sequence is in silico digested using the enzyme `CAT`. (a) Two seeded 3-grams are shown in boxes, which will be connected by running the extension algorithms on red fragments. (b) Representation of two seeds and three extension paths between them: EP1, EP2 and EP3. All nodes are restriction nodes (starting with `CAT`), and the number on each arrow shows the path length between two restriction nodes. The nodes `CATGA` and `CATGG` are the source and destination of each extension path respectively which are colored as dark red. (c) The red fragments in (a) are now considered as the target, which all extension paths will be aligned to. In this graph, EP1 is aligned to the target, with one additional (17) and one missing (4,5) cut sites. EP2 is aligned to the target with one additional (12) cut site, and EP3 does not align to target (also EP3 is longer than EP1 and EP2).

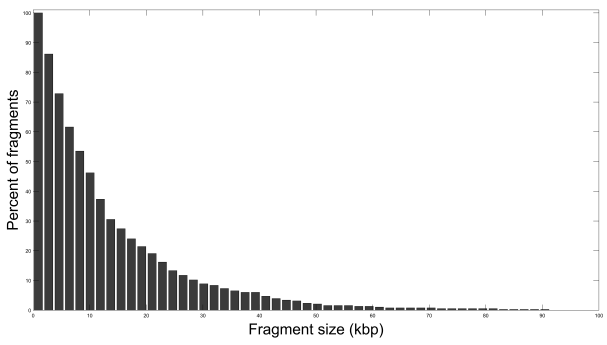


Fig. 6. The distribution of fragment sizes of E. coli when digested with restriction enzyme RsrII.

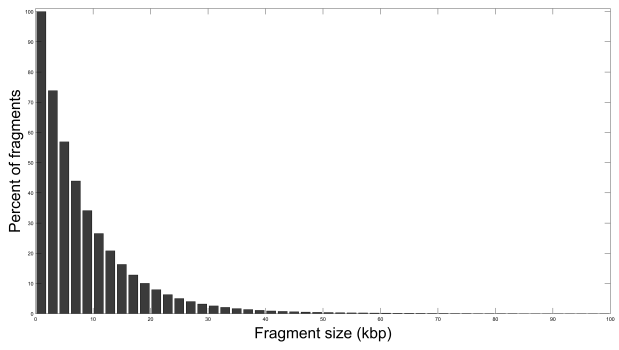


Fig. 7. The distribution of fragment sizes of human optical map data when digested with restriction enzyme BspQI.

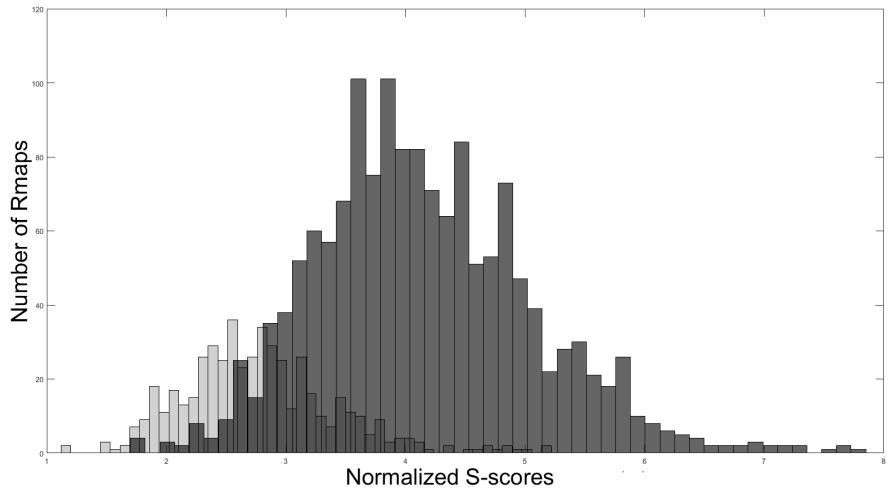


Fig. 8. The distribution of S-scores of the E. coli Rmaps. To construct this graph we performed the following experiment. We aligned each of the Rmaps to the in silico digested reference genome to create a single optical map, and then aligned each of the Rmaps to this optical map. The light bars depict the distribution of S-scores for Rmaps that were not aligned to the de Bruijn graph, while the dark bars depicts the distribution of S-scores for Rmaps that were aligned to the de Bruijn graph using our method. The mean and standard deviation of the normalized S-score for the aligned Rmaps is 4.2 and 0.71, respectively; whereas, the mean and standard deviation for the unaligned Rmaps is 2.9 and 0.56, respectively.

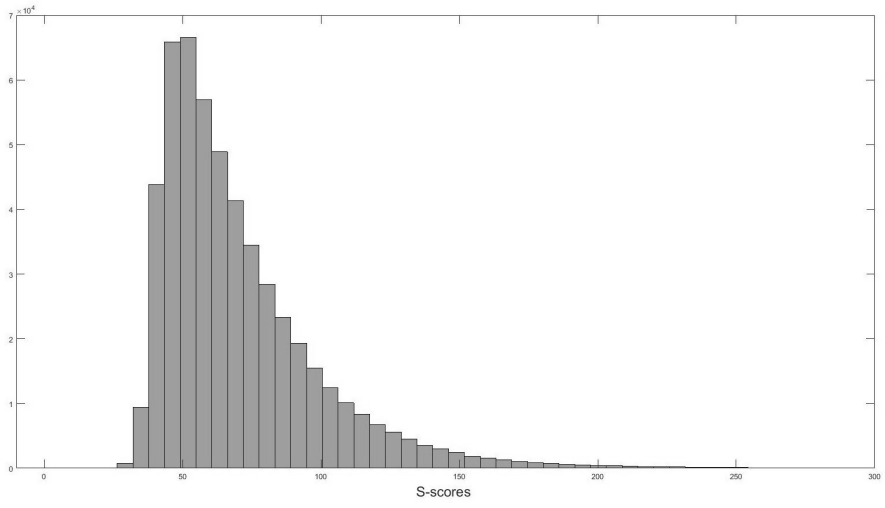


Fig. 9. Distribution of S-scores of real human Rmaps when aligned to the human reference optical map. Each Rmap was aligned to the reference using the aligner by Valouev et al. (2006a).