OXFORD UNIVERSITY PRESS | Bioinformatics

Fast and Accurate Alignment of Single Molecule Maps

Journal:	*Bioinformatics*
Manuscript ID	BIOINF-2016-0526
Category:	Original Paper
Date Submitted by the Author:	20-Apr-2016
Complete List of Authors:	Muggli, Martin; Colorado State University, Computer Science Puglisi, Simon; Yrkeshogskolan Sydvast - Helsingfors, Computer Science Boucher, Christina; Colorado State University, Computer Science
Keywords:	HitSeq, Optical Maps, Alignment, Rmap, Restriction Map

BIOINFORMATICS

Vol. 00 no. 00 2016
Pages 1–10

Fast and Accurate Alignment of Single Molecule Maps

Martin D. Muggli [1,*], Simon J. Puglisi [2] and Christina Boucher [1*]

[1]Department of Computer Science,Colorado State University, Fort Collins, CO
[2]Department of Computer Science,University of Helsinki, Finland

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: An optical map is an ordered genome-wide high-resolution restriction map that indicates the positions of one or more short nucleotide sequences. Since optical maps are derived independently of short sequence reads, they are used in *de novo* genome assembly for validating a draft genome (Nature 2013), finding structural variation (Nature Methods 2015), and detecting mis-assembled regions within draft genomes (ISMB 2015). Single molecule maps, referred to as *Rmaps*, is the raw optical mapping data used to construct the genome-wide optical maps which are used for aiding in genome assembly. Currently, there exists very few computational methods to find alignments between the Rmaps—a task that is the first step in assembling the Rmap data into a genome-wide optical maps. and is challenging due to various Rmap specific errors and their frequency.

Results: We present DOPPELGANGER, the first index-based, fully error-tolerant alignment method for optical mapping data. All prior alignment methods use dynamic programming, which is computationally expensive, or have limited error tolerance. We demonstrate a 20x speedup on the plum genome and demonstrate on the Ecoli genome the validity of our alignments. Thus, DOPPELGANGER is the only non-proprietary method that is capable of performing pairwise Rmap alignment for large eukaryote organisms in reasonable time. Lastly, we conclude with other applications of DOPPELGANGER, such as the alignment of long reads with high error rate (e.g. PacBio reads) to a genome-wide optical map.

Availability: The DOPPELGANGER alignment method is available for download at `https://github.com/mmuggli/doppelganger`.

Contact: muggli@CS.ColoState.EDU

1 INTRODUCTION

Even with significantly high coverage and various insert sizes, genome assembly and structural variation detection are tenuous computational processes when only short read sequence data is used due to the repetitive nature of genomes. In 2011, Alkan et al. [1] studied the genome assemblies of a Han Chinese individual and a Yoruban individual generated from short read sequence data, and they found that these assemblies were 16% shorter than the reference genome. Furthermore, they found that 420.2 Mbp of common repeats and 99.1% of validated duplicated sequences were missing from the genome. In fact, over 2,377 coding exons were completely missing from these draft genomes. Several years later, Ilie et al. [13] and Muggli et al. [18] detected that these misassembled regions were just as numerous even after genome assemblers became more advanced. Thus, there is active interest in generating more diverse types of data—to be used alone or in concert with short read data—in order to overcome the limitations of short read data and produce higher quality genome assemblies.

Genome-wide optical maps, which are ordered high-resolution restriction maps that indicate the positions of occurrence of one or more short nucleotide sequences, are one type of data that can aid the detection of misassembled region. The system that produces the raw optical mapping data works as follows [3, 7]: an ensemble of DNA molecules adhered to a charged glass plate are elongated by fluid flow. An enzyme is then used to cleave them into fragments at loci where the enzyme's recognition sequence occurs. Next, the remaining fragments are highlighted with fluorescent dye and digitally photographed under a microscope. Finally, these images are analyzed to estimate the fragment sizes, producing a molecular map. Since the fragments stay relatively stationary during the aforementioned process, the images capture their relative order and size [22]. Multiple copies of the genome undergo this process, producing an optical map for every molecule. These single molecule maps are referred to as *Rmaps*. Rmaps can be assembled into a genome-wide optical map as a consensus of Rmaps. A genome-wide optical map provides the approximate number of bases between occurrences of the enzyme recognition sequence within each chromosome of the genome [2].

More formally, we denote an Rmap with x fragments as $\ell = \{\ell_1, \ell_2, \ldots, \ell_x\}$, where ℓ_i is the length of the ith fragment in base pairs. Maps as a sequence of fragment sizes can be converted to a sequence of cut site locations by means of a prefix-sum. We denote the converted data as follows: $L = \{L_0 < L_1 < \cdots < L_n\}$, where $\ell_i = L_i - L_{i-1}$ for $i = 1, \ldots, n$, and L_0 and L_n are defined by the original molecule as a segment of the whole genome by shearing.

Optical mapping data is being generated at increasingly high throughput and decreasing cost due to continued technological improvements. For example, in 2015 BioNano Genomics released the Irys System, which requires one week and $1,000 USD to produce the Rmap data for an average size eukaryote genome. Whereas, it required $100,000 and six months in 2009[1]. However, even prior to 2009, optical mapping data showed to be valuable for assembling the genome of a variety of species, including several prokaryote species [24, 31, 32], *Oryza sativa* (rice) [30], maize [33], and mouse [6]. Newer technology has bee used to generate data for goat [8], *Melopsittacus Undulatus* (budgerigar) [12], and *Amborella trichopoda* [5].

*to whom correspondence should be addressed

[1] http://www.bionanogenomics.com/press-releases/bionano-genomics-launches-irys-a-novel-platform-for-complex-human-genome-analysis/

Unfortunately, even with these technology advancements and demonstrated utility, the development of computational tools that analyze the resulting data has not advanced as readily or quickly as the technology advancements. At present, there does not exist an efficient, non-proprietary methods for finding alignments between Rmaps, which is critical because it is the first step in assembling these data into a genome-wide optical map. Current methods for accomplishing this task rely on dynamic programming and thus, are unable to scale to large genomes—as described by Valouev et al. [27, 28] when they state "...their [Anantharaman et al.] algorithm had deficiencies in terms of scalability to large genomes. Consequently, application of this algorithm to genome assemblies more complex than bacteria required additional extensive ad hoc approaches. As such, there is a definite need for new algorithms that are specifically designed for handling many computational issues inherent to the assembly of large genomes, such as plant and mammalian." While the methods of Valouev et al. [27, 28] may have been motivated by inefficiencies of the method previously described by Anantharaman et al., their own results still showed there is room for further improvement; Their method required 100,000 CPU hours to compute the alignments for rice, which is a relatively small plant genome.

Applying index-based data structures, such as the Burrows-Wheeler Transform [4], to the problem of short read alignment has proven to greatly increase the space and time efficiency of the alignment task. Both BWA [15] and Bowtie [14] are examples of BWT-based short read aligners. This successful use suggests the introduction of index-based data structures may lead to an efficient Rmap alignment method; however, development of such a method is conceptually more challenging to the higher error rate. Moreover, the error profile make the alignment more akin to the problem of aligning long (e.g. PacBio) reads. The most prevalent challenges in applying index-based data structures to Rmap alignment are: (1) the fragment sizes $(\ell_1, \ell_2, \ldots, \ell_x)$ are inexact such that nearly every correspondence in an alignment is a substitution, (2) the alphabet consists of all unique fragment sizes across the raw data set and thus, is extremely large in size (over 16,000 symbols for the goat genome), and (3) the existence of insertions and deletions within the data which require one-to-many substitutions. The first two challenges lead to the observation that the FM-index standard backward search algorithm that short read aligners use cannot be used for detecting alignments between the Rmap data. The third challenge leads to the observation that a more complex index-based data structure is needed to create an aligner that is robust for insertions and deletions. Nonetheless, even with these challenges, we developed the first index-based Rmap alignment program that is capable of finding all pairwise alignments in large eukaryote organisms.

Our Contribution. We present a fast, error-tolerant method for aligning raw optical mapping data. We first abstract the problem to that of approximate-path matching in a directed acyclic graph (DAG). Our method, which we refer to as DOPPELGANGER, indexes a set of Rmaps represented as a DAG, using a suitably modified form of the *generalized compressed suffix array (GCSA)*, a variant of the FM-index developed by Siren et al. [26]. The principle novel insight of our work is that while GCSA is able to efficiently match all similar paths concurrently, it was designed for indexing variations observed in a collection of sequences; all other variations found in a query sequence are handled in query processing. In contrast, our work introduces speculative variations based on the error profile of our data into the indexed data. Further, we demonstrate that challenges posed by the inexact fragment sizes and the alphabet size can be overcome in this context with a wavelet tree [21] as we demonstrated in the less error tolerant work of Muggli et al. [19]. Hence, the novelty of our approach is the formulation of error-tolerant Rmap alignment as automaton path matching, and the application of modern index-based data structures (e.g. the GCSA with the wavelet tree)—an algorithmic paradigm that could serve useful in other contexts, such as finding all alignments between long (PacBio) reads.

We demonstrate that DOPPELGANGER is the only non-proprietary method capable of finding all alignments between Rmaps generated for eukaryote organisms in a reasonable amount of time. In particular, we show that the method of Valouev et al. requires over 40 days of CPU time to find all pairwise alignments for the plum genome but DOPPELGANGER requires only two days. Lastly, we verify our approach on simulated *E. coli* Rmap data by showing that DOPPELGANGER achieves similar sensitivity and specificity to Valouev et al. and with more permissive alignment acceptance criteria 90% of Rmaps with known overlapping genomic origin for simulated *E. coli*. The Rmap data is available with the download of our method.

Related Work. There are several existing optical map alignment methods. The majority of the existing approaches can be divided into: *fit* aligners that align small optical maps (such as Rmaps or *in silico* digested contigs) to a genome-wide optical map, and *overlap* aligners that align Rmaps to each other. The former aligners are frequently used to validate or scaffold assembled contigs. TWIN [19] is a fit aligner that uses index-based data structures. After the release of TWIN, Mendelowitz et al. [17] presented malignerIX, which is another index-based aligner that aligns *in silico* digested contigs to a genome-wide optical map. They demonstrate that the efficiency of malignerIX is comparable to TWIN but that it produces more accurate alignments. Both malignerIX and TWIN are orders of magnitude faster than dynamic programming aligners but are not suitable for aligning Rmap data due to the error profile of this raw data.

Gentig [2], the software developed by Valouev et al. [27, 28], SOMA [20] and malignerDP [17] are capable of aligning Rmap data. All methods largely use dynamic programming. SOMA [20] is, in fact, a method to scaffold short-read assemblies using a genome-wide optical map; however, SOMA provides a method for scaffolding in addition to their $O(n^2m^2)$-time dynamic programming algorithm, the latter part of this package would be capable of aligning Rmap data. Gentig [2], and software developed by Valouev et al. [27, 28] also use dynamic programming to address the closely related task of finding alignments between optical maps. Gentig is not available for download. In 2015, malignerDP [17] was developed to improve upon the dynamic programming approach of SOMA by a.) bounding the number of consecutive missed cut sites (as does the aligner by Valouev et al.) and b.) evaluate alignment significance using all non-overlapping alignments discoverable from a dynamic programming matrix in place of SOMA's expensive permutation test. They compare their method against SOMA and TWIN and demonstrate it significantly improves on the efficiency of

SOMA. At the present moment, there does not exist any index-based methods to align Rmap data.

2 BACKGROUND

2.1 Basic Definitions and Notation

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

2.2 Optical Mapping

From a computer science viewpoint, restriction mapping (by optical or other means) can be seen as a process that takes in two sequences: a genome $\mathsf{A}[1,n]$ and a restriction enzyme's recognition sequence $\mathsf{B}[1,b]$, and produces an array (sequence) of integers C, the *genome restriction map*, which we define as follows. First define the array of integers $\mathsf{C}[1,m]$ where $\mathsf{C}[i] = j$ if and only if $\mathsf{A}[j..j+b] = \mathsf{B}$ is the ith occurrence of B in A. Then $\mathsf{R}[i] = (\mathsf{C}[i] - \mathsf{C}[i-1])$, with $\mathsf{R}[1] = \mathsf{C}[1] - 1$. In words, R contains the distance between occurrences of B in A. For example, if we let $\mathsf{B} = \mathit{act}$ and

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
A	*a*	*t*	*a*	*c*	*t*	*t*	*a*	*c*	*t*	*g*	*g*	*a*	*c*	*t*	*a*	*c*	*t*	*a*	*a*	*a*	*c*	*t*

then we would have $\mathsf{C} = 3, 7, 12, 15, 20$ and $\mathsf{R} = 2, 4, 5, 3, 5$.

In reality, R is a consensus sequence formed from millions of erroneous Rmap sequences. The optical mapping system produces millions of Rmaps for a single genome. It is performed on many cells of the organism (not a single cell) and for each cell there are thousands of Rmaps (each at least 250 Kbp in length in publicly available data). The Rmaps are then assembled to produce a genome-wide optical map. Like the final R sequence, each Rmap is an array of lengths — or fragment sizes — between occurrences of B in A.

There are three types of errors that an Rmap (and hence with lower magnitude and frequency, also the consensus map) can contain: (1) missing and false cuts, which are caused by an enzyme not cleaving at a specific site, or by random breaks in the DNA molecule, respectively; (2) missing fragments that are caused by *desorption*, where small ($<$ 1 Kbp) fragments are lost and so not detected by the imaging system; and (3) inaccuracy in the fragment size due to varying fluorescent dye adhesion to the DNA and other limitations of the imaging process. Continuing again with the example above where $\mathsf{R} = 2, 4, 5, 3, 5$ is the error-free Rmap: an example of an Rmap with the first type of error could be $\mathsf{R}' = 6, 5, 3, 5$ (the first cut site is missing so the fragment sizes 2, and 4 are summed to become 6 in R'); an example of a Rmap with the second type of error would be $\mathsf{R}'' = 2, 4, 3, 5$ (the third fragment is missing); and lastly, the third type of error could be illustrated by $\mathsf{R}''' = 2, 4, 7, 3, 5$ (the size of the third fragment is inaccurately given). In the optical mapping system, there is a 20% probability that a fragment is missed and a 0.15% probability of a false break per Kbp, i.e., error type (1) occurs in a fragment. Popular restiction enzymes in optical mapping experiments recognize a 6 bp sequence giving an expected cutting density of 1 per 4096 bp. At this cutting density, false breaks are less common than missing restriction sites (approx. $0.25 * .2 = .05$ for missing sites vs. 0.0015 for false sites per bp). The inaccuracy of the fragment sizes, i.e, error type (3), follows a normal distribution with mean and variance assumed to be 0 bp and $\ell\sigma^2$ ($\sigma = .58$ kbp), respectively [27].

2.3 Rank and Select

Two basic operations used in almost every succinct and compressed data structure are *rank* and *select*. Given a sequence (string) $S[1,n]$ over an alphabet $\Sigma = \{1,\ldots,\sigma\}$, a character $c \in \Sigma$, and integers i,j, $\mathsf{rank}_c(S,i)$ is the number of times that c appears in $S[1,i]$, and $\mathsf{select}_c(S,j)$ is the position of the j-th occurrence of c in S.

2.4 Suffix Arrays, BWT and Backward Search

The suffix array [16] SA_X (we drop subscripts when they are clear from the context) of a sequence X is an array $\mathsf{SA}[1..n]$ which contains a permutation of the integers $[1..n]$ such that $\mathsf{X}[\mathsf{SA}[1]..n] < \mathsf{X}[\mathsf{SA}[2]..n] < \cdots < \mathsf{X}[\mathsf{SA}[n]..n]$. In other words, $\mathsf{SA}[j] = i$ iff $\mathsf{X}[i..n]$ is the j^{th} suffix of X in lexicographic order.

For a sequence Y, the Y-interval in the suffix array SA_X is the interval $\mathsf{SA}[s..e]$ that contains all suffixes having Y as a prefix. The Y-interval is a representation of the occurrences of Y in X. For a character c and a sequence Y, the computation of $c\mathsf{Y}$-interval from Y-interval is called a *left extension*.

The Burrows-Wheeler Transform $\mathsf{BWT}[1..n]$ is a permutation of X such that $\mathsf{BWT}[i] = \mathsf{X}[\mathsf{SA}[i]-1]$ if $\mathsf{SA}[i] > 1$ and \$ otherwise [4]. We also define $\mathsf{LF}[i] = j$ iff $\mathsf{SA}[j] = \mathsf{SA}[i] - 1$, except when $\mathsf{SA}[i] = 1$, in which case $\mathsf{LF}[i] = I$, where $\mathsf{SA}[I] = n$.

Ferragina and Manzini [9] linked BWT and SA in the following way. Let $\mathsf{C}[c]$, for symbol c, be the number of symbols in X lexicographically smaller than c. The function $\mathsf{rank}(\mathsf{X},c,i)$, for sequence X, symbol c, and integer i, returns the number of occurrences of c in $\mathsf{X}[1..i]$. It is well known that $\mathsf{LF}[i] = \mathsf{C}[\mathsf{BWT}[i]]+\mathsf{rank}(\mathsf{BWT},\mathsf{BWT}[i],i)$. Furthermore, we can compute the left extension using C and rank. If $\mathsf{SA}[s..e]$ is the Y-interval, then $\mathsf{SA}[\mathsf{C}[c] + \mathsf{rank}(\mathsf{BWT},c,s), \mathsf{C}[c] + \mathsf{rank}(\mathsf{BWT},c,e)]$ is the $c\mathsf{Y}$-interval. This is called *backward search* [9], and a data structure supporting it is called an *FM-index*.

To support the rank queries needed for backward search, a data structure called a *wavelet tree* [11, 21] is commonly used. It occupies $n\log\sigma + o(n\log\sigma)$ bits of space and supports rank queries in $O(\log\sigma)$ time. Wavelet trees also support a variety of more complex queries on the underlying string efficiently (see, e.g. [10]). One wavelet tree query we will make use of in this paper is *range alphabet*, $\mathsf{ralpha}(i,j)$, which returns the set X of distinct symbols occurring in $S[i,j]$. Wavelet trees support ralpha queries in $O(|X|\log\sigma)$ time.

3 METHODS

In this section we give our problem formulation, the algorithm behind DOPPELGANGER, and some important implementation details. The three main insights that enable our index-based aligner for Rmap data is the abstraction of the alignment problem to a finite automaton, the use of the GCSA for storing and querying the automaton, and the modification of the backward search algorithm to use the wavelet tree in specific ways important to characteristics of RMap data. Casting the alignment problem as path

matching in a finite automaton allows the first two errors (missing and false cut sites, and missing fragments) to be modelled. The GCSA is an index-based data structure for storing and querying finite automaton. The third type of error (inaccuracy in the fragment sizes) is taken into account by the use of the wavelet tree in the backward search algorithm. For simplicity we first describe standard backward search and then show how it can be extended.

3.1 The Rmap Alignment Problem

The problem of aligning Rmap data involves comparing one Rmap (the *query*) against the set of all other Rmaps in the dataset (the *target*). We will denote the query Rmap as R_q and the target database as $\mathsf{R}_1 \ldots \mathsf{R}_n$, where each R_i is a sequence of fragment sizes, i.e, $\mathsf{R}_i = [f_{i1}, .., f_{im_i}]$. An alignment between two Rmaps is a relationship between them comprising groups of zero or more consecutive fragments in one Rmap which correspond to groups of zero or more consecutive fragments in the other. For example, a typical alignment between $\mathsf{R}_i = [4, 5, 10, 9, 3]$ and $\mathsf{R}_j = [10, 9, 11]$ would be $\{[4, 5], [10]\}, \{[10], [9]\}, \{[9], [11]\}, \{[3], []\}$. A group may contain more than one fragment (e.g. $[4, 5]$) when the restriction site delimiting the fragments is absent in the corresponding group of the other Rmap (e.g $[10]$). This can occur under two scenarios: 1) when there is a false restriction site in one Rmap, for example if a random break in the DNA was responsible for creating fragments of measured size 4 and 5 out of an original fragment of length 9 in the first group of the preceeding example or 2) when there is a missing restriction site in the other, for example if the DNA fragment measured to be of length 10 in the first group contained an enzyme recognition sequence but was not digested. Since these scenarios complement each other and we cannot tell from only two Rmaps which occurred, for the purpose of our remaining discussion it will be sufficient to consider only the scenario of missed (undigested) restriction sites, given that they arrise in both target and query Rmaps.

During the alignment search process, we try various assignments of fragments to groups. Since groups need to agree well in size, it is useful to have a concept that ignores the number of constituent fragments. Thus, we introduce the term *compound fragment* to refer to a hypothetical fragment of the total length of a putative group. We refer to a *proper* compound fragment if it is specifically composed of multiple fragments. In our previous example, the group $[4, 5]$ whose sum is 9 is a proper compound fragment and would pair with the compound fragment 10 in R_j.

3.2 Finite Automaton

Continuing with the example in Section 2, we need to be able to align $\mathsf{R}' = 7, 6, 3, 4$ to $\mathsf{R} = 2, 4, 5, 3, 5$ and vice versa. In order to accomplish this it will be useful to cast the Rmap alignment problem to matching paths in a finite automaton. A finite automaton is a directed, labeled graph that defines a *language*, or a specific set of sequences composed of vertex labels. A sequence is recognized by an automaton if there exists a path *matching* that spells that sequence in the automaton. We represent the target Rmaps as an automaton and the query as a path in this context. The intuition is as follows: we can construct an automaton where vertices represent compound fragments (groups of fragments), and the language comprises all the legal assignments of fragments to groups. That is, every fragment of an Rmap is assigned to exactly one group.

The automaton for our target Rmaps can be constructed as follows. First concatenate the $\mathsf{R}_1 \ldots \mathsf{R}_n$ together into a single sequence with each Rmap separated by a special symbol which will not match any query symbol. Let R^* denote this concatenated sequence. Hence, $\mathsf{R}^* = [f_{11}, .., f_{1m_1}, \ldots, f_{n1}, .., f_{nm_n}]$. Then, construct an initial finite automaton $\mathsf{A} = (V, E)$ for R^* by creating a set of vertices $v_1^i..v_m^i$, one vertex labeled with each fragment length and edges connecting them for each Rmap R_i. Additionally, introduce to A a *starting vertex* v_1 labeled with # and a *final vertex* v_f labeled with the character \$. All other vertices in A will be labeled with integral values. This initial set of vertices and edges is called the *backbone*. The backbone by it self would only be sufficient for finding alignments with no missing cut sites in the query. Figure 1(a) illustrates the construction of A for a single Rmap. The backbone of this automaton is $\#, 2, 3, 4, 5, 6, \$$. Next, extra vertices ("skip vertices") and extra edges are added to A to allow for the automaton to accept all valid queries.

Skip Vertices and Skip Edges We introduce additional vertices labeled with compound fragments to allow missing cut sites (first type of error) to be taken into account in querying the target Rmaps. We refer to these as *skip vertices* as they provide alternative path segments which skip past two or more backbone vertices. Thus, we add a *skip vertex* to A for every $o + 1$ length run of consecutive vertices in the backbone where $1 < o < order$ and *order* is the maximum number of consecutive missed cut sites to be accommodated. First order skip vertices are each labeled with the sum of two consecutive backbone vertices. Second order skip vertices are each labeled with the sum of three consecutive backbone vertices. The vertex labeled with 7 connecting 2 and 5 in 1(a) is an example of a skip vertex. Likewise, $5, 9, 11$ are other skip vertices.

Finally, we add *skip edges* which provide paths around vertices with small labels in the backbone. These allow a query to match paths that exclude such small fragments whose presence may not be conserved consistently across Rmaps. Hence, the addition of skip edges allow for desorption (the second type of error) to be taken into account in querying the target Rmaps.

3.3 The Rmap Alignment Score

Alignments are found by incrementally extending candidate partial alignments (paths in the automaton) to longer partial alignments by choosing one of several compatible extension matches (adjacent vertices to the end of a path in the automaton). To perform this search efficiently, we bound the partial search by a scoring model that scores the size agreement of the matched compound fragments, and the frequency of putative missing cut sites.

Size Agreement We use the cumulative distribution functions for the Chi-square (Chi-square CDF) statistic to asses the size agreement. These assume the fragment size errors are independent, normally distributed events. For each pair of matched compound fragments in a partial alignment, we take the average between the two as the assumed true length of the DNA that was measured. Then we compute the expected standard deviation using this mean. Each compound fragment deviates from the assumed true value by half the distance between them. These two deviation values contribute two degrees of freedom to the Chi-square calculation.

Cut Site Error Frequency We use the binomial cumulative distribution (Binomial CDF) statistic to asses the probability of the number of cut site errors in a partial alignment. The assumption under this model is that the missing cut site errors are independent, Bernoulli processes events. We account for all the putatively conserved cut sites on the boundaries and those delimiting compound fragments in both partially aligned Rmaps plus twice the number of missed sites as Bernoulli trials.

Thus, given a set of Rmaps and input parameters ρ_L and ρ_U, we produce the set of all Rmap alignments that have a chi-square CDF statistic less than ρ_U and a binomial CDF statistic less than ρ_L. Both of these are subject to the additional constraint of a maximum consecutive missed restriction site run between aligned sites of δ and a minimum aligned site set cardinality of 10.

3.4 Generalized Compressed Suffix Array

We construct an index for the automaton using the GCSA [26], which allows efficient storage and path querying. The GCSA can be seen as a generalization of the FM-index—which as previously mentioned, only supports indexing and querying strings. Nonetheless, there are some parallels between the two data structures and we will explain the GCSA by drawing on the definition of the (more commonly known) FM-index. As stated in Section 2, the FM-index is based on the deep relationship between the SA and the BWT data structures of the input string X. The BWT of an input string is formed by sorting all characters of the string by

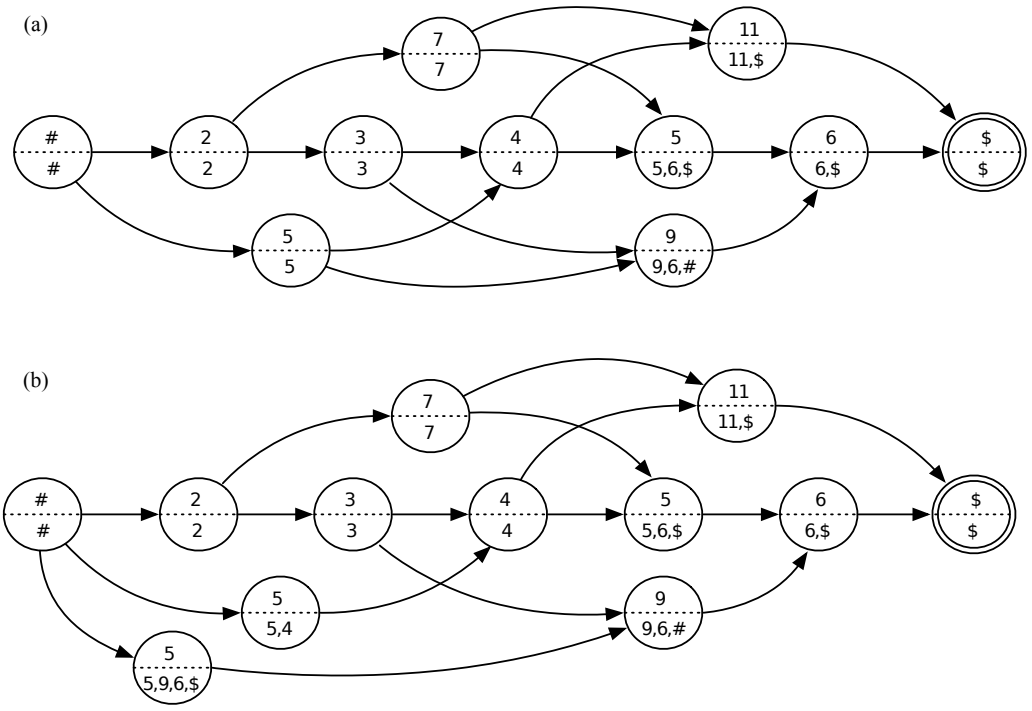

Fig. 1. Figure 1(a) shows an example automaton. The top half of vertices contains the label, which models a fragment size in Kbp. The common prefixes of all suffixes spelled from a vertex is written in the bottom half. Note that there is no ordering of vertices such that all their corresponding suffixes are in lexicographic order. The leftmost vertex labelled with "5" spells suffixes beginning "5,4,..." as well as the suffix "5,9,6,$" while the rightmost 5 spells the suffix "5,6,$". Figure 1(b) shows gives the prefix sorted automaton corresponding to the one in (a). The leftmost vertex 5 has been duplicated and the outgoing edges of the previous version have been divided between the new replacement instances. This also divides the suffixes spellable from the prior version. Now the three 5 vertices can be ordered based on their common prefixes as ["5,4,...","5,6,$", "5,9,6, $"].

	$	2	3	4	5,4	5,6,$	5,9,6,$	6,$	7	9,6,$	11,$	#
BWT	6,11	#	2	3,5	#	4,7	#	5,9	2	3,5	4,7	$
M	1	10	10	10	1	1	1	1	10	1	1	100
F	10	1	1	10	1	10	1	10	1	10	10	1

Table 1. Table listing the three arrays storing the automaton in memory: BWT, M, and F.

the lexicographic order of the suffix immediately following each character. The main properties the FM-index exploits in order to perform queries efficiently are a) $\mathsf{BWT}[i] = \mathsf{X}[\mathsf{SA}[i]-1]$; and b) given that $\mathsf{SA}[i] = j$, and $\mathsf{C}[c]$ gives the position of the first suffix in SA prefixed with character c, then using small auxiliary data structures we can quickly determine $k = \mathsf{C}[\mathsf{BWT}[i]] + \mathsf{rank}(\mathsf{BWT}, \mathsf{BWT}[i], i)$, such that $\mathsf{SA}[k] = j - 1$. The first of these properties is simply the definition of the BWT. The second is because the symbols of X occur in the same order in both the single character prefixes in the suffix array and in the BWT; Given a set of sorted suffixes, prepending the same character onto each suffixes does not change their order. Thus, if we consider all the suffixes in a range of SA which are preceded by the same symbol c, that subset will appear in the same relative order in (another part of) SA: as a contiguous subinterval of the interval that contains all the suffixes beginning with c. Thus by knowing where a symbol's run begins in the SA and the rank of an instance of that symbol, we can identify the SA position beginning with that instance from its position in BWT. Auxiliary data structures storing this information plus the BWT thus constitute a kind of *compressed suffix array*. The FM-index LF() function computes the SA position from the BWT position using these auxiliary data structures.

In order to generalize the FM-index to automata (instead of strings), we need to efficiently store the vertices and edges in a manner such that the FM-index properties still hold, allowing the GCSA to support queries efficiently. An FM-index's compressed suffix array for a string S encodes a relationship between each suffix S and its left extension. Hence, this suffix array can be generalized to edges in a graph that represent a relationship between vertices. We note that the traditional compressed suffix array for a given string is a special case of this generalization where the vertices are labeled with the string's symbols in a contiguous non-branching path. Now, we will describe this generalization in detail by describing how to sort all vertices and edges in an automaton using BWT and two bit arrays.

Vertex Representation In order to index the vertices we first construct an unique string, which we refer to as the *common prefix*, and associate each vertex with respect to that string. This association will allow the vertices to be sorted by their common prefixes and from then on identified by their rank in this sorted order. To construct the common prefix string, it is important

to note that the automaton defines a language comprising a set of strings. Thus, every path through the automaton corresponds to one of these strings. If we consider all of the paths that include a particular vertex, we see that the vertex will also be the initial symbol of a suffix for each path. Those suffixes will have a common prefix–even if it is just the node label itself–which is the common prefix string.

These common prefixes may not be sufficient to sort vertices as necessary. In some cases, two vertices may share a property that all paths reachable from one are lexicographically smaller than all paths reachable from the other. A vertex that shares this property with every other vertex is prefix sorted. An automaton where every vertex is prefix sorted is a prefix sorted automaton. A non-prefix sorted automaton can be made prefix sorted through a process of duplicating vertices and their incoming edges but dividing their outgoing edges between the new instances. See Figure 1 for an example of a non-prefix sorted automaton and a prefix sorted automaton. The common prefix of these vertices will then be longer, and with sufficient repetitions of this process, a non-prefix sorted automaton will become prefix sorted. Thus, in order to ensure the vertices can be identified by their rank in sorted order, we construct the automaton so that it is prefix sorted.

Edge Representation Next, we describe the edge representation. Each vertex has a set of one or more preceding vertices and therefore, a set of predecessor labels in the automaton. These predecessor label sets are concatenated to form the BWT array. The order of concatenation among these sets is based on the aforementioned lexicographic rank of the successor of each set. Each element of BWT then denotes an edge in the automaton. Another array of bits, F, marks a '1' for the first element of BWT corresponding to a vertex and a '0' for all subsequent elements. Thus, the predecessor labels, and hence the associated edges, for a vertex with rank r can be found in $\mathsf{BWT}[\mathsf{select}(r)..\mathsf{select}(r+1)]$. Another array, M, captures the out degree of each vertex and allows the set of vertex ranks associated with a BWT interval to be found using rank() queries.

3.5 Exact Matching: GCSA Backward Search

Exact matching with the GCSA is similar to the standard FM-index backward search algorithm. As outlined in Section 2, the FM-index backward search proceeds by finding a succession of lexicographic ranges that progressively match longer and longer suffixes of the query string, starting from the rightmost symbol of the query. Hence, the search maintains two items — a lexicographic range and an index into the query string — and the property that the path prefix associated with the lexicographic range is equal to the suffix of the query marked by the query index. Initially, the query index is at the rightmost symbol and the interval is the entire array since every path prefix matches an empty suffix. The search continues using GCSA's backward search step function, which is supplied with the next symbol from the query string (i.e. fragment size in R_q) and the current interval, and returns a new interval. The query index is advanced leftward after each backward search step. In theory, since the current interval corresponds to a consecutive range in the BWT, the backward search could use use select() queries on a bit vector F to determine all the edges adjacent to a given vertex and then two FM-index LF() queries are applied to the limits of the current interval to obtain the new interval. GCSA's implementation uses one succinct bit vector per alphabet symbol to encode which symbols precede a given vertex instead of F. Finally, this new interval, which corresponds to a set of edges, is mapped back to a set of vertices using rank() on the M bit vector.

3.6 Inexact Matching: GCSA Backward Search Using a Wavelet Tree

We made the following modifications in order to adapt the backward search algorithm: (1) we used a wavelet tree to allow efficient retrieval of substitution candidates; (2) modified the search process to join various lengths of consecutive query fragments into compound fragments so as to match fragments in R^* missing the interposing restriction site; and (3) introduced backtracking, both to try both substitution candidates and combinations of compound fragment. We now discuss each of these modifications in more detail.

First, in order to accommodate possible errors in the fragment sizes, we determine a set, D, of candidate match fragment sizes that are similar in size to the next fragment of R_q to be matched in the query. These candidates are drawn from the interval of the BWT (predecessor labels) corresponding to the interval currently active in our backward search (matched labels) using the wavelet tree algorithm of Gagie et al. [10]. This requires introducing bit array F into the actual implementation. (Recall that this active interval, when applied to a lexicographic range, represents the suffixes whose prefixes are the matched portion of the query, while the same interval applied to the BWT represents possible extension symbols). To accommodate possible restriction sites that are present in the query Rmap but absent in target Rmaps, we generate compound fragments; we try combining single, pairs and triples of consecutive query fragments by adding their sizes and query the wavelet tree for substitutions of these compound fragments. Note that summing multiple consecutive fragments is complementary to the skip vertices in the target automaton. It allows accommodation of missed restriction sites in the target, just as the skip vertices in the target accommodate missed restriction sites in the query.

Similarly, small fragments may be subject to desorption in either the query or the target. One could simply prune all the fragments below a certain threshold size; however, due to the sizing error, one cannot discard based on the true fragment size and one would certainly not achieve consistent discarding across all Rmaps (e.g. a small fragment in two maps covering the same genomic region may be above the threshold in one map and below in another, leading to inconsistent discarding). Therefore, to find all alignments, some mechanism to allow inconsistent inclusion of small fragments must be included. We use skip edges to allow paths around fragments found in the target set that might be missing from a query Rmap. Normally, one might need to include skipping over small fragments in not just the target, but in the query as well. This would however result in further branching and slow down the search. As an alternative, we use separate thresholds for the query and the target so that we can ensure that if a given genomic fragment only is retained in one sequence, it will be retained in the target.

Lastly, since there may be multiple match candidates in the BWT interval of R^* for a compound fragment drawn from R_q, we extend the backward search with backtracking so that each candidate size returned to the search algorithm from the wavelet tree is evaluated, i.e., for a given compound fragment size f generated from R_q, every possible candidate fragment size, f', that can be found in R^* in the range $f-t \ldots f+t$ and in the interval $s \ldots e$ (of the BWT of R^*) for some tolerance t is used as a substitute in the backward search. Each of these candidates is then checked to ensure that the longer partial alignment formed by a left extension would still satisfy alignment criteria. This requires that the alignment criteria be *monotone*, that is, that the prefixes of a credible long alignment are also credible alignments. Since the chi-square and binomial CDF statistics are p-values, using them as alignment criteria satisfy this requirement. When this criteria is still satisfied after including a candidate symbol, it is used as the extension symbol in the backward search. This neighborhood is chosen to be within a radius of six standard deviations about the query symbol, since each of the query and target fragments may be three standard deviations away from the actual fragment size in the genome.

3.7 Practical Considerations

Pruning Queries One side effect of summing consecutive fragments in both the search algorithm and the target data structure is that several successive search steps with agreeing fragment sizes will also have agreeing sums of those successive fragments. In this scenario, proceeding deeper in the search space will result in wasted effort. To reduce this risk, we maintain a table of scores obtained when reaching a particular lexicographic range and query cursor pair. We only proceed with the search past this point when either the

point has never been reached before, or has only been reached before with inferior scores.

Wavelet Tree Cutoff The wavelet tree allows efficiently finding the set of vertex labels that are predecessors of the vertices in the current match interval intersected with the set of vertex labels that would be compatible with the next compound fragment to be matched in the query. However, when the match interval is sufficiently small (< 750) it is faster to scan the vertices in BWT directly.

Quantization The alphabet of fragment sizes can be large considering all the measured fragments from multiple copies of the genome. This can cause an extremely large branching factor for the initial symbol and first few extensions in the search. To improve the efficiency of the search, the fragment sizes are initially quantized, thus reducing the size of the effective alphabet and the number of substitution candidates under consideration at each point in the search. Quantization also increases the number of identical path segments across the indexed graph which allows a greater amount of candidate matches to be evaluated in parallel because they all fall into the same BWT interval during the search. This does, however, introduce some quantization error into the fragment sizes, but the bin size is chosen to keep this small in comparison to the sizing error.

4 DISCUSSION

We evaluated the performance of DOPPELGANGER against the only publicly available competing method, Valouev et al. [27], on simulated Rmap data for *E. coli* and real Rmap data for plum. Our experiments measured runtime, peak memory usage, and the number of alignments found by both DOPPELGANGER and the software of Valouev et al., for each dataset. Peak memory was measured as the maximum resident set size as reported by the operating system. Runtime was the user process time, also reported by the operating system. All alignment experiments were performed on Intel x86-64 Xeon workstations with sufficient RAM to avoid paging (16 GB), running 64-bit Linux. Query reads were distributed among these workstations using the IPython Parallel package; however, timing and memory results were tabulated from values reported within each of the applications. The parallel infrastructure was only used for experimental convenience. Both applications are single threaded, single machine applications.

DOPPELGANGER was configured to run on plum using 100 bp bins for quantization, a minimum match length parameter of 15 compound fragments, a χ^2 CDF threshold of 0.05, a binomial CDF threshold of 0.45 and two orders of skip nodes.

The software of Valouev et al. was run with default parameters except for reducing the maximum compound fragment length (their δ parameter) from 6 fragments to 3. We observed that the software of Valouev et al. rarely included alignments containing more than two missed restriction sites in a compound fragment, hence we compare both tools with this compound fragment size limit.

As previously mentioned, the method of Valouev et al. is the only freely available method for aligning Rmap data. Gentig [2] and BACop [33] were not available for download so could not be tested. Although SOMA [20] could potentially align Rmap data, it implements a dynamic programming algorithm that is similar in spirit to that of Valouev et al., but was not intended for this purpose; it was created instead for aligning *in silico* digested contigs to a genome-wide optical map. For this reason, we did not compare against SOMA [20]. We note that previous work [19] demonstrates that SOMA is significantly slower than Valouev et al. on larger genomes. Similarly, TWIN [19] was created for the same purpose as SOMA, and furthermore, assumes a consensus optical map with no missing restriction sites, so we did not compare against it either. We tried to run MalignerDP on the plum data but the process stopped making visible progress for over a day after 300 hours of runtime.

Species	Genome Size	No. of Reads	Depth
Plum	280 Mbp	140,000	135x
E. coli	4.639 Mbp	272	34x

Table 2. Description of the datasets used for our experiments. The genome of plum was only recently assembled and thus, the genome size is an estimation based on the assembly [29].

4.1 Performance on Simulated E.coli Rmap Data

To verify the correctness of our method, we simulated a read set from the *E. coli* reference genome. This simulation was performed as follows: we started with 1,400 copies of the genome, and then generated 40 uniformly distributed random loci within each of 1,400 copies of the genome which are taken to be breakpoints. These breakpoints form the ends of single molecules that would undergo digestion. Molecules smaller than 250 Kbp were discarded. The cleavage sites for the XhoI enzyme were then identified within each of these simulated molecules. We removed 20% of these at random from each simulated molecule to model partial digestion. Finally, normally distributed noise was added to each fragment with a standard deviation of .58 kb per 1 kb of the fragment. Because the origin loci of all of these simulated molecules was known within the reference genome, we could identify all pairs of simulated molecules that had more than 16 conserved cleavage sites after removing the 20%. These pairs with overlapping regions of origin become the ground truth data for testing our method and that of Valouev et al. This method of simulation was based on the *E. coli* statistics given by Valouev et al. [28] and giving a molecule length distribution as observed in publicly available Rmap data from OpGen, Inc. We note that while the Rmap length filtering in combination with the unproven molecule breakpoint simulation approach only leaves 272 Rmaps, alignment methods are still challenged with finding the 4,305 overlapping pairs from the $272^2/2$ possible pairs from this dataset.

Our simulation data based alignment results are summarized in Table 3 and described as follows. The software of Valouev et al. required 155 seconds and 4.0 MB of memory to find 699 of the 4,305 (16%) ground truth alignments. DOPPELGANGER, when configured with a χ^2 CDF threshold of .05 and a binomial CDF threshold of .45 ran for 14 seconds and occupied 19.0 MB of memory to find 707 of the 4,305 (27%) ground truth alignments. With this configuration, both tools favor high specificity over sensitivity. The differences in their alignment scoring models appears as a small agreement in their alignment sets. However, both sets consist of high quality alignments and we have no grounds for evaluating one as being better than the other.

MalignerDP is capable of handling unmatched sites in either target or query sequence, though is designed for fit alignments of queries against a genome wide target map instead of a collection of Rmaps. Accordingly, we relaxed some of its default settings with the

additional arguments "–ref-max-misses 2 –query-miss-penalty 3.0 –query-max-miss-rate 0.50", since both source and target are equally noisy in this context. It found 1,150 of the 4,305 (27%) ground truth alignments in 33 seconds and consumed 5.7 MB of memory in the process.

DOPPELGANGER uses χ^2 and binomial CDF thresholds to prune the backtracking search when deciding whether to extend alignments to progressively longer alignments. More permissive match criteria, using higher thresholds, allows more Rmaps to be reached in the search and thus considered aligned, but it also results in less aggressive pruning in the search, thus lengthening runtime. When DOPPELGANGER was configured with a CDF threshold of .5 and a binomial CDF threshold of .7, it ran for 585 seconds and found 3,859 of the 4,305 (90%) ground truth alignments.

In interpreting alignment quality statistics, it is easy to see that Rmaps with overlapping origin should be found to have a high quality alignment. However, various types of repeats can also introduce high quality alignments between Rmaps which do not have overlapping origins. These are not faults of the alignment method but intrinsic to the data itself and, to some degree, the choice of enzyme. These repeats in the data occur because of (1) genomic repeats; (2) whole genome restriction maps may have repeats beyond those in the sequence data as any distinguishing variations in the DNA sequence between restriction sites will not affect the map; and (3) sizing error. To expand upon the last point, experimentally derived optical maps are not precise to the base pair resolution and thus, disparate regions in an error-free restriction map may be indistinguishable at the precision of optical experimental measurement techniques.

While Sarkar et al. [25] have done some work to evaluate the quality of alignment metrics, the field is still in its infancy, and there is not yet a gold standard of discrimination ability. We wish to emphasize that our contribution is not focused on alignment quality; we have adopted a simple but capable model and our contribution is primarily that of effectively overcoming the unique challenges in adapting index based methods to optical mapping data. Indeed, our method and software can be easily fit with any alignment scoring method that effectively evaluates candidate partial alignments while full alignment are incrementally extended from them.

4.2 Performance on Plum Rmap Data

The Beijing Forestry University, Beijing Genome Institute, Beijing Lin Fu Ke Yuan Flowers Co., Ltd, and other institutes assembled the first genomic sequence of plum (*Prunus mume*). The plum cultivar sequenced is known as *mei*. This sequencing and assembly is of significant agricultural importance because it pioneers an initial understanding of the evolution and genetic modification of fruit trees [29]. For this species, several diverse datasets were generated, including short read data with various insert lengths and optical mapping data generated from OpGen Inc. The optical map is available in the GigaScience repository. The data was assembled using Genome-Builder™ software developed at OpGen, which itself employs the use of Gentig. The amount of CPU time was not reported and Genome-Builder™ and Gentig are not publicly available so we could not compare our methods against these software tools. We used the Rmap data from June, 2011. This data was generated using the BamHI restriction enzyme and consists of 139,281 rmaps.

Our results on this plum dataset are summarized in Table 4 and described as follows. DOPPELGANGER, configured with a χ^2 CDF threshold of .05, took 48 hours of CPU time to search for alignments across the plum Rmap data, representing a 15x speedup of the 1,001 hours taken by the Valouev et al. software. The largest alignment job used a peak of 7.6 GB of RAM. Building the GCSA data structure required 11 minutes and required 13 GB of RAM. Loading the GCSA structure from disk before alignment required 50 seconds.

Method	Time	Memory	Alignments
DOPPELGANGER	48 hours	7.6 GB	22,029,938
Valouev et al.	1,001 hours	61 MB	6,387

Table 4. Performance on Plum.

Our results from *E. coli* simulation demonstrate that our method has comparable capability in finding alignments to the software of Valouev et al. Additionally, these results show that for these same parameter settings our method yields significant speedup on eukaryote scale Rmap data.

5 CONCLUSION

We demonstrate that DOPPELGANGER is capable of finding the alignment between any pair of Rmaps in the June plum data set in less than 67 CPU hours whereas the dynamic programming method of Valouev et al. [27] requires 1001 hours of CPU time. This latter method computes the optimal alignment between all prefixes of every pair of Rmaps, while our method succeeds in avoiding this exhaustive computation by using an index data structure to narrow the search to consider a smaller set of plausible alignments only. Hence, although, the dynamic programming methods achieve practical running time on small genomes, they are unlikely to scale to large genomes. As previously mentioned, the first step in building a genome wide consensus map from the Rmap data is to compute the pairwise alignment among Rmaps, and this is the primary motivation for the development of DOPPELGANGER. This pairwise alignment step is the main computational bottleneck in the consensus map building tool of Valouev et al., and DOPPELGANGER could easily be substituted for their dynamic programming alignment method, to obtain an efficient means to compute the consensus map for large genomes, such as plum.

Taking a broader view, DOPPELGANGER is simply an error-tolerant index-based alignment program that could have additional purposes beyond Rmap data. For example, one interesting application of this work would be to align *in silico* digested long (Pac Bio) reads to a genome wide consensus optical map, or to the Rmaps. This complements the work of Pendleton et al. [23], which scaffolds Pac Bio data using BioNano Irys data. Hence, our alignment method could be used to assemble or preprocess Pac Bio reads that have large errors and should undergo error correction, or to isolate the regions in the reads that should undergo error correction. The PacBio RS technology generates extremely long reads ($\geq$1,000 bp), but with high single-pass error rates (15% error rate), and correction of these reads is widely regarded as

Method	Time	Memory	Alignments	Fraction of Ground Truth
DOPPELGANGER ($\chi^2 < .05$, Binom. $< .45$)	14 s.	19.0 MB	923	707 / 4,305 (16%)
DOPPELGANGER ($\chi^2 < .5$, Binom. $< .7$)	585 s.	18.3 MB	8,858	3,859 / 4,305 (90%)
Valouev et al.	155 s.	4.0 MB	742	699 / 4,305 (16%)
MalignerDP	33 s.	5.7 MB	2,134	1,150 / 4,305 (27%)

Table 3. Performance on simulated *E. coli* dataset.

computationally expensive. This proposed filtering step could make the error correction process more efficient.

REFERENCES

[1]C. Alkan, S. Sajjadian, and E.E. Eichler. Limitations of next-generation genome sequence assembly. *Nature Methods*, 8(1):61–65, 2010.

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

[3]C. Aston and D.C. Schwartz. *Optical Mapping in Genomic Analysis*. John Wiley and Sons, Ltd, 2006.

[4]M. Burrows and D.J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[5]S. Chamala, A.S. Chanderbali, J.P. Der, T. Lan, B. Walts, V.A. Albert, C.W. de Pamphilis, J. Leebens-Mack, S. Rounsley, S.C. Schuster, R.A. Wing, N. Xiao, R. Moore, P.S. Soltis, and D.E. Soltis. Assembly and validation of the genome of the nonmodel basal angiosperm *amborella*. *Science*, 342(6165):1516–1517, 2013.

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

[7]E.T. Dimalanta, A. Lim, R. Runnheim, C. Lamers, C. Churas, D.K. Forrest, J.J. de Pablo, M.D. Graham, S.N. Coppersmith, S. Goldstein, and D.C. Schwartz. A microfluidic system for large DNA molecule arrays. *Analytical Chemistry*, 76(18):5293–5301, 2004.

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

[9]P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

[10]T. Gagie, G. Navarro, and S. J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theoretical Computer Science*, 426-427:25–41, 2012.

[11]R. Grossi, A. Gupta, and J.S. Vitter. High-order entropy-compressed text indexes. In *Proc. of SODA*, pages 841–850, 2003.

[12]J. T. Howard, S. Koren, A. Phillippy, S. Zhou, D. Schwartz, M. Schatz, R. Aboukhalil, J. M. Ward, J. Li, B. Li, O. Fedrigo, L. Bukovnik, T. Wang, G. Wray, I. Rasolonjatovo, R. Winer, J. R. Knight, W. Warren, G. Zhang, and E. D. Jarvis. *De Novo* high-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience Database*, 3:11, 2014.

[13]L. Ilie, B. Haider, M. Molnar, and R Solis-Oba. SAGE: String-overlap Assembly of GEnomes. *BMC Bioinformatics*, 15(302), 2014.

[14]B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[15]H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–60, 2009.

[16]U. Manber and G. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Scientific Computing*, 22(5):935–948, 1993.

[17]L. Mendelowitz, D. Schwartz, and M. Pop. . *To appear in Journal of Bioinformatics*.

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

[19]M.D. Muggli, S.J. Puglisi, and C. Boucher. Efficient indexed alignment of contigs to optical maps. In *Proc. of WABI*, pages 68–81, 2014.

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

[21]G. Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

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

[23]M. Pendleton et al. Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nature Methods*, 12:780–786, 2015.

[24]S. Reslewic, S. Zhou, M. Place, Y. Zhang, A. Briska, S. Goldstein, C. Churas, R. Runnheim, D. Forrest, A. Lim, A. Lapidus, C.S. Han, G.P. Roberts, and D.C. Schwartz. Whole-genome shotgun optical mapping of *Rhodospirillum Rubrum*. *Applied Environmental Microbiology*, 71(9):5511–5522, 2005.

[25]D. Sarkar et al. Statistical significance of optical map alignments. *Journal of Computational Biology*, 19(5):478–492, 2012.

[26]J. Sirén, N. Välimäki, and V. Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

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

[28]A. Valouev, D.C. Schwartz, S. Zhou, and M.S. Waterman. An algorithm for assembly of ordered restriction maps from single DNA molecules. *Proceedings of the National Academy of*

Sciences, 103(43):15770–15775, 2006.

[29]Q. Zhang et al. The genome of *Prunus mume*. *Nature Communications*, 3:1318, 2012.

[30]S. Zhou, M.C. Bechner, M. Place, C.P. Churas, L. Pape, S.A. Leong, R. Runnheim, D.K. Forrest, S. Goldstein, and M. Livny. Validation of rice genome sequence by optical mapping. *BMC Genomics*, 8(1):278, 2007.

[31]S. Zhou, W. Deng, T.S. Anantharaman, A. Lim, E.T. Dimalanta, J. Wang, T. Wu, T. Chunhong, R. Creighton, A. Kile, E. Kvikstad, M. Bechner, G. Yen, A. Garic-Stankovic, J. Severin, D. Forrest, R. Runnheim, C. Churas, C. Lamers, N.T. Perna, V. Burland, F.R. Blattner, B. Mishra, and D.C. Schwartz. A whole-genome shotgun optical map of *Yersinia pestis* strain KIM. *Applied and Environmental Microbiology*, 68(12):6321–6331, 2002.

[32]S. Zhou, A. Kile, E. Kvikstad, M. Bechner, J. Severin, D. Forrest, R. Runnheim, C. Churas, T.S. Anantharaman, P. Myler, C. Vogt, A. Ivens, K. Stuart, and D.C. Schwartz. Shotgun optical mapping of the entire *Leishmania major* Friedlin genome. *Molecular and Biochemical Parasitology*, 138(1):97–106, 2004.

[33]S. Zhou, F. Wei, J. Nguyen, M. Bechner, K. Potamousis, S. Goldstein, L. Pape, M.R. Mehan, C. Churas, S. Pasternak, D.K. Forrest, R. Wise, D. Ware, R.A. Wing, M.S. Waterman, M. Livny, and D.C. Schwartz. A single molecule scaffold for the maize genome. *PLoS Genetics*, 5(11):e1000711, 2009.