A Succinct Solution to RMap Alignment

Martin D. Muggli
Department of Computer Science, Colorado State University
martin.muggli@colostate.edu

Simon J. Puglisi
Department of Computer Science, University of Helsinki, Finland

Christina Boucher
Department of Computer and Information Science and Engineering, University of Florida

Abstract

We present KOHDISTA, which is an index-based algorithm for finding pairwise alignments between single molecule maps (*Rmaps*). The novelty of our approach is the formulation of the alignment problem as automaton path matching, and the application of modern index-based data structures. In particular, we combine the use of the Generalized Compressed Suffix Array (GCSA) index with the wavelet tree in order to build KOHDISTA. We validate the approach on *E. coli* data, and show KOHDISTA is at least 3x faster than existing methods for finding quality pairwise alignments in plum Rmap data. Lastly, we demonstrate KOHDISTA is the only non-proprietary method that is capable of finding high quality pairwise Rmap alignments for large eukaryote organisms in reasonable time. KOHDISTA is available at `https://github.com/mmuggli/KOHDISTA/`.

2012 ACM Subject Classification WABI 2018

Keywords and phrases Optical mapping, index based data structures, FM-index, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI 2018..

1 Introduction

There is a current resurgence in generating 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. Data from an optical mapping system [1, 4] is one such example and has itself become more practical with falling costs of high-throughput methods. For example, the current BioNano Genomics Irys System 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]. These technological advances and the demonstrated utility of optical mapping in genome assembly [16, 21, 22, 3, 5] have driven several recent tool development efforts [20, 10, 13].

Genome-wide optical maps are ordered high-resolution restriction maps that give the position of occurrence of restriction cut sites corresponding to one or more restriction enzymes. These genome-wide optical maps are assembled using an overlap-layout-consensus approach using raw optical map data, which are referred to as *Rmaps*. Hence, Rmaps are akin to reads in genome sequencing. To date, however, there is no efficient, non-proprietary method for finding pairwise alignments between Rmaps, which is the first step in assembling genome-wide maps.

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

Several existing methods are superficially applicable to Rmap pairwise alignments but all programs either struggle to scale to even moderate size genomes or require significant further adaptation to the problem. Several methods exhaustively evaluate all pairs of Rmaps using dynamic programming. One of these is the method of Valouev *et al.* [19], which is capable of solving the problem exactly but requires over 100,000 CPU hours to compute the alignments for rice [18]. The others are SOMA [15] and MalignerDP [13] which are designed only for semi-global alignments instead of overlap alignments, which are required for assembly.

Other methods reduce the number of map pairs to be individually considered by initially finding seed matches and then extending them through more intensive work. These include OMBlast [10], OPTIMA [20], and MalignerIX [13]. These, along with MalignerDP, were designed for a related alignment problem of aligning consensus data but cannot consistently find high quality Rmap pairwise alignments in reasonable time as we show later. This is unsurprising since these methods were designed for either already assembled optical maps or *in silico* digested sequence data for one of their inputs, both having a lower error rate than Rmap data.

Our contributions. In this paper, we present a fast, error-tolerant method for performing pairwise Rmap alignment that makes use of a novel FM-index based data structure. Although the FM-index can naturally be applied to short read alignment [11, 9], it is non-trivial to apply it to Rmap alignment. The difficulty arises from: (1) the abundance of missing or false cut sites, (2) the fragment sizes require inexact fragment-fragment matches (e.g. 1,547 bp and 1,503 bp represent the same fragment), (3) the Rmap sequence alphabet consists of all unique fragment sizes and is so extremely large (e.g., over 16,000 symbols for the goat genome). The second two challenges render inefficient the standard FM-index backward search algorithm, which excels at exact matching over small alphabets. The first (and most-notable) challenge requires a more complex index-based data structure be used to create an aligner that is robust for insertion and deletion of cut sites. To overcome the mismatch cut site challenge while still accommodating the other two, we develop KOHDISTA, an index-based Rmap alignment program that is capable of finding all pairwise alignments in large eukaryote organisms.

We first abstract the problem to that of approximate-path matching in a directed acyclic graph (DAG). The KOHDISTA method then indexes a set of Rmaps represented as a DAG, using a modified form of the *generalized compressed suffix array (GCSA)*, which is a variant of the FM-index developed by Siren *et al.* [17]. The principle 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. In contrast, our work indexes variations that are instead speculative, based on the Rmap error profile. Lastly, we demonstrate that challenges posed by the inexact fragment sizes and alphabet size can be overcome, specifically in the context of the GCSA, via careful use of a wavelet tree [7, 14].

We verify our approach on simulated *E. coli* Rmap data by showing that KOHDISTA achieves similar sensitivity and specificity to Valouev *et al.*, and with more permissive alignment acceptance criteria 90% of Rmap pairs simulated from overlapping genomic regions. We also show the utility of our approach on larger eukaryote genomes by demonstrating that existing published methods require more than 151 hours of CPU time to find all pairwise alignments in the plum Rmap data; whereas, KOHDISTA requires 31 hours. Thus, we present the first fully-indexed method capable of finding all match patterns in the pairwise Rmap alignment problem.

2 Background

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

Overview of 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 B be `act` and A = `atacttactggactactaaact` 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 an organism 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).

Frequency of Errors. In the optical mapping system, there is a 20% probability that a cut site 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 [19].

Suffix Arrays, BWT and Backward Search The suffix array [12] 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 [2]. 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 [6] 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* [6], and a data structure supporting it is called an *FM-index*.

To support rank queries in backward search, a data structure called a *wavelet tree* (see [7]) can be 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. [7]). One such query we will use in this paper is to return the set X of distinct symbols occurring in $S[i, j]$, which takes $O(|X| \log \sigma)$ time.

3 The Pairwise Rmap Alignment Problem

Given a genome $\mathsf{A}[1, n]$ and a restriction enzyme's recognition sequence $\mathsf{B}[1, b]$, the optical mapping system produces Rmaps, which are arrays of lengths—or fragment sizes—between occurrences of B in A. The background section provides details on the optical mapping process. Producing Rmap data is an error prone process. Thus, there are three types of errors can occur: (1) missing and false cuts that delimit fragments; (2) missing fragments; and (3) inaccuracy in the fragment sizes. For example, let $\mathsf{R} = 2, 4, 5, 3, 5$ be an error-free Rmap, then 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)

The pairwise Rmap alignment problem aims to align one Rmap (the *query*) R_q against the set of all other Rmaps in the dataset (the *target*). We denote 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 relation between them comprising groups of zero or more consecutive fragment sizes in one Rmap associated with groups of zero or more consecutive fragments in the other. For example, given $\mathsf{R}_i = [4, 5, 10, 9, 3]$ and $\mathsf{R}_j = [10, 9, 11]$ one possible alignment is $\{[4, 5], [10]\}, \{[10], [9]\}, \{[9], [11]\}, \{[3], []\}$. 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 if there is a false restriction site in one Rmap, or there is a missing restriction site in the other. Since we cannot tell from only two Rmaps which of these scenarios occurred, for the purpose of our remaining discussion it will be sufficient to consider only the scenario of missed (undigested) restriction sites.

4 Methods

We now describe the algorithm behind KOHDISTA. Three main insights enable our index-based aligner for Rmap data: 1) abstraction of the alignment problem to a finite automaton; 2) use of the GCSA for storing and querying the automaton; and 3) modification of backward search to use a wavelet tree in specific ways to account for the Rmap error profile.

4.1 Finite Automaton

Continuing with the example in the background section, we want to align $\mathsf{R}' = 6, 5, 3, 5$ to $\mathsf{R}''' = 2, 4, 7, 3, 5$ and vice versa. To accomplish this we cast the Rmap alignment problem to that of 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 it contains a matching path: a consecutive sequence of vertex labels equal to the sequence. We represent the target Rmaps as an automaton and the query as a path in this context.

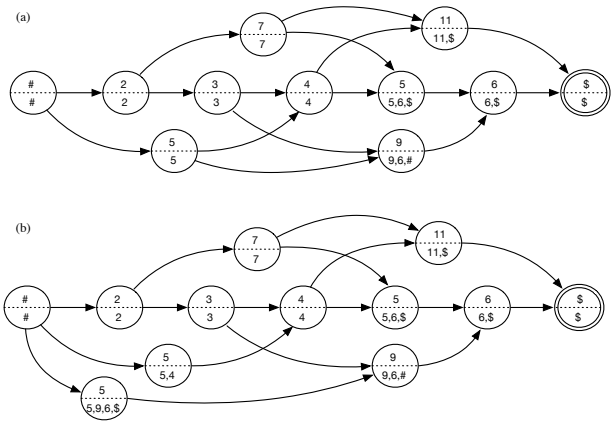
The automaton for our target Rmaps can be constructed as follows. First concatenate the $\mathsf{R}_1 \ldots \mathsf{R}_n$ together into a single sequence with each Rmap separated by a special symbol which will not match any query symbol. Let R^* denote this concatenated sequence. Hence, $\mathsf{R}^* = [f_{11}, .., f_{1m_1}, \ldots, f_{n1}, .., f_{nm_n}]$. 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.Also, 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 are labeled with integral values. This initial set of vertices and edges is called the *backbone*. The backbone by itself is only sufficient for finding alignments with no missing cut sites in the query. 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. Figure 1a(a) illustrates the construction of A for a single Rmap.

4.1.1 Skip Vertices and Skip Edges

We introduce extra 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 1a(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 with a missing fragment to still match.[2] Hence, the addition of skip edges allow for desorption (the second type of error) to be taken into account in querying the target Rmaps.

[2] Different smallness thresholds for query and target bias toward this scenario, avoiding backtracking in the search.



(a) An example automaton. The top half of vertices contains the label, which models a fragment size in Kbp. The common prefixes of all suffixes spellable 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,$" while the rightmost 5 spells the suffix "5,6,$". (b) shows 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, $"].

	BWT	M	F
$	6	1	1
	11		0
2	#	1	1
		0	
3	2	1	1
		0	
4	3	1	1
	5	0	0
5,4	#	1	1
5,6,$	4	1	1
	7		0
5,9,6,$	#	1	1
6,$	5	1	1
	9		0
7	2	1	1
		0	
9,6,$	3	1	1
	5		0
11,$	4	1	1
	7		0
#	$	1	1
		0	
		0	

(b) Table listing the three arrays storing the automaton in memory: BWT, M, and F. Each row in the table delimits its elements associated with a particular vertex.

Figure 1 Example automata and corresponding memory representation.

4.2 Generalized Compressed Suffix Array

We index the automaton with the GCSA [17] for efficient storage and path querying. The GCSA is a generalization of the FM-index for automata and we will explain the GCSA by drawing on the definition of the (more widely known) FM-index. As stated in the background section, 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 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 suffix 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. A rank data structure over the BWT thus constitutes a sufficient compressed index of the suffix array needed for traversal.

To generalize the FM-index to automata (from 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. The compressed suffix array for a string is a special case where the vertices are labeled with the string's symbols in a non-branching path.

4.2.1 Prefix-sorted Automata

Just as backward search for strings is linked to suffix sorting, backward searching in the BWT of the automaton requires us to be able to sort the vertices (and a special set of the paths) of the automaton in a particular way. In [17] this property is called *prefix-sortedness*. Let $A = (V, E)$ be a finite automaton, let $v_{|V|}$ denote its terminal vertex, and let $v \in V$ be a vertex. We say v is *prefix-sorted* by prefix $p(v)$ if the labels of all paths from v to $v_{|V|}$ share a common prefix $p(v)$, and no path from any other vertex $u \neq v$ to $v_{|V|}$ has $p(v)$ as a prefix of its label. Automaton A is prefix-sorted if all vertices are prefix-sorted. See Figure 1a for an example of a non-prefix sorted automaton and 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 [17]).

Clearly the prefixes $p(v)$ allow us to sort the vertices of a prefix-sorted automaton into lexicographical order. Moreover, if we consider the list of outgoing edges (u, v), sorted by pairs $(p(u), p(v))$, they are also sorted by the sequences $\ell(u)p(v)$, where $\ell(u)$ denotes the label of vertex u. This (dual sortedness) property allows backward searching to work over the list of vertex labels (sorted by $p(v)$) in the same way that is does for the symbols of a string ordered by their following suffixes in normal backward search for strings.

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. The sets are concatenated in the order defined by the above mentioned lexicographic ordering of the vertices. Each element in 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 in that set. Thus, the predecessor labels, and hence the associated edges, for a vertex with rank r are $\mathsf{BWT}[\mathsf{select}(r)..\mathsf{select}(r+1)]$. Another array, M, stores the out degree of each vertex and allows the set of vertex ranks associated with a BWT interval to be found using rank() queries.

4.3 Exact Matching: GCSA Backward Search

Exact matching with the GCSA is similar to the standard FM-index backward search algorithm. As outlined in the background section, 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. 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 range is $[1..n]$ since every path prefix matches the empty suffix. The search continues using GCSA's backward search step function, which takes as parameters the next symbol (to the left) in the query (i.e. fragment size in R_q) and the current range, and returns a new range. The query index is advanced leftward after each backward search step. In theory, since the current range corresponds to a consecutive range in the BWT, the backward search could 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 range to obtain the new one. 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 range, which corresponds to a set of edges, is mapped back to a set of vertices using rank() on the M bit vector.

4.4 Inexact Matching: GCSA Backward Search Using a Wavelet Tree

We modified GCSA backward search in the following ways: (1) we used a wavelet tree to allow efficient retrieval of substitution candidates; (2) we modified the search process to combine consecutive query fragments into compound fragments so as to match fragments in R^* missing the interposing restriction site; and (3) we introduced backtracking, in order to both try size substitution candidates as well as various combinations of compound fragments. These modifications are further detailed below.

First, in order to accommodate possible errors in fragment size, we determine a set, D, of candidate fragment sizes that are similar to the next fragment of R_q to be matched in the query. These candidates are determined by enumerating the distinct symbols in the currently active backward-search range of the BWT[3] using the wavelet tree algorithm of Gagie *et al.* [7]. This method was proposed by Muggli *et al.* [14] for use with an FM-index but was not directly applicable to the originally proposed implementation of GCSA. This is because some of GCSA's theoretical constructs (i.e. F) were substituted in implementation for efficiency reasons. In order to apply the aforementioned wavelet tree method, we thus resurrect the previously theoretical only bit array F (which we encode succinctly) as well as symbol array BWT (which we encoded with a wavelet tree) into KOHDISTA using the SDSL-Lite library by Gog *et al.* [8].

To accommodate possible restriction sites that are present in the query Rmap but absent in target Rmaps, we generate compound fragments (i.e. new symbols) by summing pairs and triples of consecutive query fragment size and then querying the wavelet tree for substitutions of these compound fragments. This summing of multiple consecutive fragments is complementary to the skip vertices in the target automaton and accommodates missed restriction sites in the target, just as the skip vertices accommodate missed sites in the query.

Lastly, since there may be multiple match candidates in the BWT interval of R^* for a compound fragment generated from R_q and multiple compound fragments generated at a given position in R_q, we employ the common practice of adding backtracking to backward search (as is done, for example in the works of Li *et al.* and Langmead *et al.*). This is 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.

[3] Recall that this active range, when applied to a lexicographic range, represents the suffixes whose prefixes are the matched portion of the query, while the same range of the BWT contains possible extension symbols.

5 Results and Discussion

We evaluated KOHDISTA against the other available optical map alignment software. Our experiments measured runtime, peak memory, and alignment quality on simulated *E. coli* Rmaps and experimentally generated plum Rmaps. All experiments were performed on Intel Xeon computers with $\geq$ 16 GB RAM running 64-bit Linux.

5.1 Performance on Simulated E.coli Rmap Data

To verify the correctness of our method, we simulated a read set from a 4.6 Mbp *E. coli* reference genome as follows: we started with 1,400 copies of the genome, and then generated 40 random loci within each. These loci form the ends of molecules that would undergo digestion. Molecules smaller than 250 Kbp were discarded leaving 272 molecules with a combined length equating to 35x coverage depth. 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. Simulated molecule pairs having 16 common conserved digestion sites become the "ground truth"[4] data for testing our method with the others. Although a molecule would align to itself, these are not included in the ground truth set. This method of simulation was based on the *E. coli* statistics given by Valouev *et al.* [18] and resulting in a molecule length distribution as observed in publicly available Rmap data from OpGen, Inc.

Most of the tools were designed for less noisy data but in theory could address all the data error types required. For tools with tunable parameters, we tried aligning the *E. coli* Rmaps with combinations of parameters for each method related to its alignment score thresholds and error model parameters. We used parameterization giving results similar to those for the default parameters of Valouev *et al.*'s method to the extent such parameters did not significantly increasing each tool's runtime. These same parameterization were used in the next section on Plum data.

Even with tuning, we were unable to obtain pairwise alignments on *E. coli* for two methods. We found OPTIMA only produced self alignments with its recommended overlap protocol and report its resource use in Table 1. For MalignerIX, even when we relaxed the parameters to account for the greater sizing error and mismatch cut site frequency, it was also only able to find self alignments. This is expected as by design it only allows missing sites in one sequence in order to run faster. Thus no further testing was performed with MalignerIX or OPTIMA. We did not test SOMA as earlier investigation indicate it would not scale to larger genomes [14].

Results on *E. coli* are presented in Table 1. KOHDISTA 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 to be considered aligned, but it also results in less aggressive pruning in the search, thus lengthening runtime. As an example, note that when KOHDISTA was configured with a much relaxed CDF threshold of .5 and a binomial CDF threshold of .7, it found 3,925 of the 4,305 (91%) ground truth alignments, but slowed down considerably. This illustrates the index and algorithm's capability in handling all error types.

[4] Due to repeats in the restriction map, and apparent repeats at the resolution attainable through optical measurement, some alignments beyond these are expected.

Method	Time	Memory	Align-ments	Recall	Precision
KOHDISTA	20 s.	19.0 MB	907	702 / 4,305 (16%)	702 / 907 (77%)
KOHDISTA (lax)	373 s.	18.3 MB	8,545	3,925 / 4,305 (91%)	3,925 / 8,545 (46%)
Valouev *et al.*	148 s.	4.0 MB	742	699 / 4,305 (16%)	699 / 742 (94%)
MalignerDP	47 s.	6.0 MB	1,959	1,296 / 4,305 (30%)	1,296 / 1959 (66%)
OMBlast	116 s.	2,078 MB	1,008	806 / 4,305 (19%)	806 / 1008 (80%)
RefAligner	31 s.	81.2 MB	992	958 / 4,305 (22%)	948 / 992 (97%)
MalignerIX	4 s.	6.0 MB	0	0 / 4,305 (0%)	0 / 0 (N/A)
OPTIMA	455 s.	10,756.5 MB	0	0 / 4,305 (0%)	0 / 0 (N/A)

Table 1 Performance on simulated *E. coli* dataset. KOHDISTA (lax) demonstrates that our indexing and search method is capable of finding the majority of ground truth alignments when the search is pruned to the more relaxed thresholds of $\chi^2 < .02$, Binom. $< .5$.

Method	Time	Memory	Alignments
KOHDISTA	31 hours	7.4 GB	16,109,151
Valouev *et al.*	678 hours	60 MB	6,387
MalignerDP	214 hours	784 MB	568,744
OMBlast	151 hours	12.3 GB	424,730
RefAligner	90 hours	374 MB	10,039

Table 2 Performance on Plum.

5.2 Performance on Plum Rmap Data

The Beijing Forestry University and other institutes assembled the first plum (*Prunus mume*) genome using short reads and optical mapping data from OpGen Inc. We test the various available alignment methods on the 139,281 plum Rmaps from June 2011 available in the GigaScience repository. These Rmaps were created with the BamHI enzyme and have a coverage depth of 135x of the 280 Mbp genome. For the plum dataset, we ran all the methods which approach the statistical performance of the Valouev *et al.* method when measured on *E. coli*. Thus, we omitted MalignerIX and OPTIMA because they had 0% recall and precision on *E. coli*. Our results on this plum dataset are summarized in Table 2.

KOHDISTA was the fastest and achieved obtained the greatest number of alignments than the competing methods. When configured with a χ^2 CDF threshold of .02, it took 31 hours of CPU time to find all the pairwise alignments in the plum Rmap data. This represents a 21x speed-up over the 678 hours taken by the exhaustive Valouev *et al.* software. The other non-proprietary methods, MalignerDP and OMBlast, took 214 hours and 151 hours, respectively. These results represent a 6.9x and 4.8x speed-up over MalignerDP and OMBlast. All methods used less than 13 GB of RAM and thus, were considered practical from a memory perspective.

To measure the quality of the alignments, we scored each pairwise alignments using the scoring scheme of Valouev *et al.* and present histograms of these alignment scores in Figure 2. For comparison, we also scored and present the histogram for random pairs of Rmaps. Both Valouev *et al.* produces very few but high-scoring alignments and although it could theoretically be altered to produce a larger number of alignments, the running time makes this prospect impractical (678 hours). Although KOHDISTA and RefAligner produce high-quality alignments, RefAligner produced very few alignments (10,039) and required almost 5x more time to do so. OMBlast and Maligner required significantly more time and produced significantly lower quality alignments.

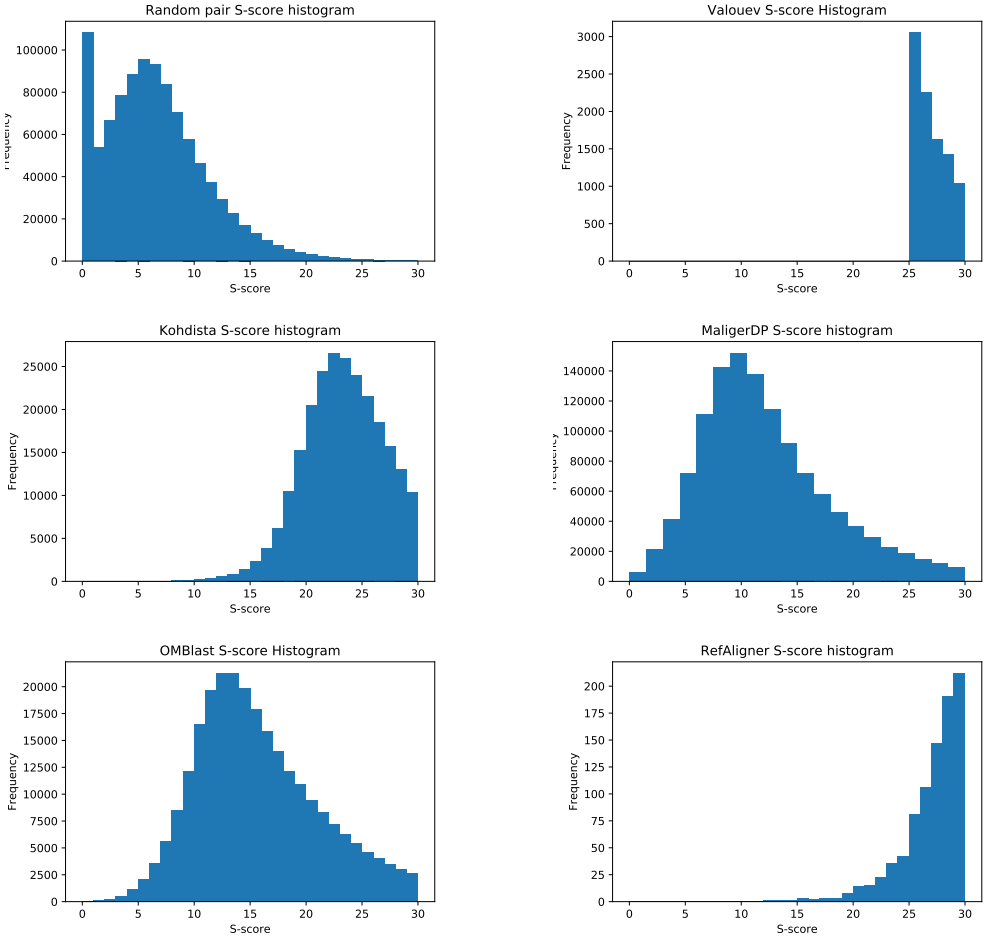


Figure 2 All alignments found on Plum were realigned using Valouev *et al.*'s dynamic programming method. Their method finds the optimal alignment using a function balancing size agreement and cut site agreement known as an s-score. (a) The s-score distribution for random pairs. (b) The Valouev *et al.* software considers any pair with an s-score > 25 to be aligned. (c) KOHDISTA alignments tend to have significantly higher s-scores than random. (d) MalignerDP alignments tend to have slightly higher s-scores than random. (e) OMBlast alignments tend to have higher s-scores than random. (f) BioNano's commercial RefAligner method alignments tends to have a significantly higher s-scores than random.

6 Conclusion

In this paper, we demonstrate how finding pairwise alignments in Rmap data can be modelled approximate-path matching in a directed acyclic graph, and combining the GCSA with the wavelet tree results in a index-based data structure for solving this problem. We implement this method and present results comparing KOHDISTA with competing methods. By demonstrating results on both simulated *E. coli* Rmap data and real plum Rmaps, we show that KOHDISTA is capable of detecting high scoring alignments in efficient time. In particular, KOHDISTA detected the largest number of alignments in 31 hours. RefAligner, a proprietary method, very few high scoring alignments (10,039) and requires almost 5x more time to do so. OMBlast and Maligner required significantly more time and produced

significantly lower quality alignments. Valouev *et al.* produced high scoring alignments but required more than 21x time to do.

References

1. C. Aston and D.C. Schwartz. *Optical Mapping in Genomic Analysis*. Wiley, 2006.
2. M. Burrows and D.J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
3. S. Chamala et al. Assembly and validation of the genome of the nonmodel basal angiosperm *amborella*. *Science*, 342(6165):1516–1517, 2013.
4. 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.
5. 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.
6. P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.
7. 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.
8. Simon Gog et al. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.
9. B. Langmead et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.
10. Alden King-Yung Leung et al. Omblast: alignment tool for optical mapping using a seed-and-extend approach. *Bioinformatics*, page btw620, 2016.
11. H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–60, 2009.
12. 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.
13. L. M. Mendelowitz et al. Maligner: a fast ordered restriction map aligner. *Bioinformatics*, 32(7):1016–1022, 2016.
14. 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.
15. 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.
16. S. Reslewic et al. Whole-genome shotgun optical mapping of *Rhodospirillum Rubrum*. *Applied Environmental Microbiology*, 71(9):5511–5522, 2005.
17. J. Sirén et al. Indexing graphs for path queries with applications in genome research. *IEEE/ACM TCBB*, 11(2):375–388, 2014.
18. A. Valouev et al. An algorithm for assembly of ordered restriction maps from single DNA molecules. *Proc Natl Acad Sci*, 103(43):15770–15775, 2006.
19. A. Valouev et al. Alignment of optical maps. *J. Comp Bio*, 13(2):442–462, 2006.
20. Davide Verzotto et al. Optima: Sensitive and accurate whole-genome alignment of error-prone genomic maps by combinatorial indexing and technology-agnostic statistical analysis. *GigaScience*, 5(1):2, 2016.
21. S. Zhou et al. A whole-genome shotgun optical map of *Yersinia pestis* strain KIM. *Applied and Environmental Microbiology*, 68(12):6321–6331, 2002.
22. S. Zhou et al. Shotgun optical mapping of the entire *Leishmania major* Friedlin genome. *Molecular and Biochemical Parasitology*, 138(1):97–106, 2004.

A Practical Indexing Considerations

A.1 Pruning the Search

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 prune the search by computing the χ^2 and binomial CDF statistics of the partial matches and use thresholds to ensure reasonable size agreement of the matched compound fragments, and the frequency of putative missing cut sites.

A.1.1 Size Agreement.

We use the Chi-square CDF statistic to assess size agreement. This assumes the fragment size errors are independent, normally distributed events. For each pair of matched compound fragments in a partial alignment, we take the mean between of the two as the assumed true length and 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. Thus each deviation is normalized by dividing by the expected standard deviation, these are squared, and summed across all compound fragments to generate the χ^2 statistic. We use the standard χ^2 CDF function to compute the area under the curve of the probability mass function up to this χ^2 statistic, which gives the probability two Rmap segments from common genomic origin would have a χ^2 statistic no more extreme than observed. This probability is compared to KOHDISTA's chi-squared-cdf-thresh and if smaller, the candidate compound fragment is assumed to be a reasonable match and the search continues.

A.1.2 Cut Site Error Frequency.

We use the Binomial CDF statistic to assess the probability of the number of cut site errors in a partial alignment. This assumes 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 the number of Bernoulli trials. We use the standard binomial CDF function to compute the sum of the probability density function up to the number of non-conserved cut sites in a candidate match. Like the size agreement calculation above, this gives the probability two Rmaps of common genomic origin would have the number of non-conserved sites seen or fewer in the candidate partial alignment under consideration. This is compared to the binom-cdf-thresh to decide whether to consider extensions to the given candidate partial alignment. 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 16.

A.1.2.1 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.

A.1.2.2 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.

A.1.2.3 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.

A.1.2.4 Example Traversal

A partial search for a query Rmap [3 kb, 7 kb, 6 kb] in Figure 1a and Table **??** given an error model with a constant 1 kb sizing error would proceed with steps: 1. Start with the semi-open interval matching the empty string [0..12). 2. A wavelet tree query on BWT would indicate the set of symbols {5, 6, 7} is the intersection of two sets: 1.) The set of symbols that would all be valid left extensions of the (currently empty) match string and 2.) The set of size appropriate symbols that match our next query symbol (i.e. 6 kb, working from the right end of our query) in light of the expected sizing error (i.e. 6kb +/- 1 kb). 3. We would then do a GCSA backward search step on the first value in the set (5) which would yield the new interval [4..7). This new interval denotes only nodes where each node's common prefix is compatible with the spelling of our current backward traversal path through the automaton (i.e. our short path of just [5] does not contradict any path spellable from any of the three nodes denoted in the match interval). 4. A wavelet tree query on the BWT for this interval for values 7 kb +/- 1 kb would return the set of symbols 7. 5. Another backward search step would yield the new interval [8..9). At this point our traversal path would be [7, 5] (denoted as a left extension of a forward path that we are building by traversing the graph backward). The common prefix of each node (only one node here) in our match interval (i.e. [7 kb]) is compatible with the path [7, 5]. This process would continue until backward search returns no match interval or our scoring model indicates our repeatedly left extended path has grown too divergent from our query. At this point backtracking would occur to find other matches (e.g. at some point we would backward search using the value 6 kb instead of the 5 kb obtained in step 2.)

A.1.2.5 Parameters Used

We tried OPTIMA with both "p-value" and "score" scoring and the allMaps option and report the higher sensitivity "score" setting. We followed the OPTIMA-Overlap protocol of splitting Rmaps into k-mers, each containing 12 fragments as suggested in [20]. For OMBlast, we adjusted parameters maxclusteritem, match, fpp, fnp, meas, minclusterscore, and minconf. For MalignerDP, we adjusted parameters max-misses, miss-penalty, sd-rate, min-sd, and max-miss-rate and additionally filtered the results by alignment score. Though unpublished, for comparison we also include the proprietary RefAligner software from BioNano. For RefAligner we adjusted parameters FP, FN, sd, sf, A, and S. For KOHDISTA, we adjusted parameters chi-squared-cdf-thresh and binom-cdf-thresh. For Valouev, we adjusted score_thresh and t_score_thresh variables in the source. In Table 1 we report statistical and computational performance for each method.

OMBlast was configured with parameters meas=3000, minconf=0.09, minmatch=15 and the rest left at defaults. RefAligner was run with parameters FP=0.15, sd=0.6, sf=0.2, sr=0.0, se=0.0, A=15, S=22 and the rest left at deafults. MalignerDP was configured with parameters ref-max-misses=2, query-miss-penalty=3, query-max-miss-rate=0.5, min-sd=1500, and the rest left at defaults.

The software of Valouev *et al.* was run with default parameters except we reduced the maximum compound fragment length (their δ parameter) from 6 fragments to 3. We observed the software of Valouev *et al.* rarely included alignments containing more than two missed restriction sites in a compound fragment.

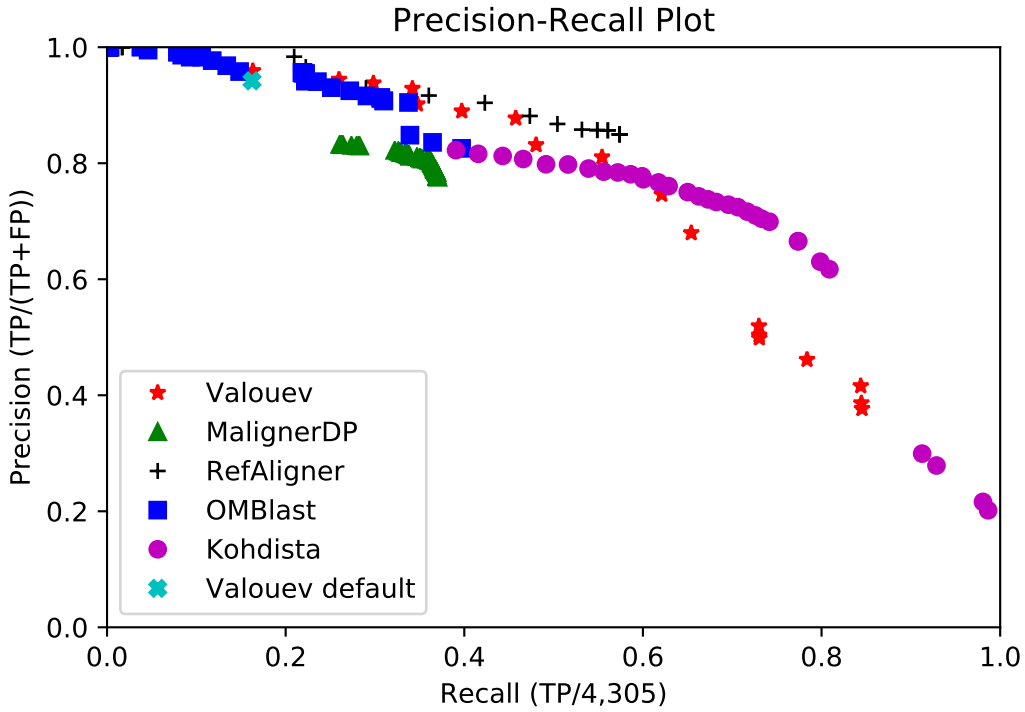


Figure 3 Precision-Recall plot of successful methods on *simulated E. coli*