Internal Report CS Bioinformatics Track 16-03 October 2016

Leiden University

Computer Science

Bioinformatics Track

Improving Sequence Alignment through Population Graph Inference.

Tom Mokveld

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgments

I would like to thank my supervisors Marcel Reinders and Jasper Linthorst for their continued support, guidance, and advice during my time working on this project.

Contents

1 Introduction

Over the past twenty years high throughput sequencing technology has greatly advanced our understanding of genetics [1, 2]. This has lead to insights in areas such as gene functionality, expression, regulation, and structural information of the genome [3–5]. The availability of reference genomes, constructed by projects like the Human Genome Project [6–8], have played a key role in this success.

To this day, the importance of these reference genomes has not diminished. The main purpose of a reference genome in this setting is that it serves as a target for read alignment; the mapping of fragmented genome sequences (reads) against it. It provides the coordinate system by which reads can be related to the reference in order to identify genetic variants and put them into the context of genes and other functional elements.

In principle, sequence variation can also be addressed by the comparison of whole-genome assemblies. However, the construction of high quality *de novo* assemblies is costly and therefore often infeasible in practice (especially for genomes as large and complex as the human genome). For this reason, reference based approaches are often the only practical solution for detecting sequence variations.

The process of read alignment is not without difficulties, as errors introduced by different sequencing technologies introduce ambiguity in the placement of reads. In downstream variant detection this may result in incorrectly calling or altogether missing variations. Further complications can also be attributed to the simplicity of using a single reference genome. If we consider the variation seen in a population, every new instance of a studied genome may contain sequence that is not encoded or highly diverged with respect to the reference genome. This leads to difficulties resolving variants in these regions [9].

One can account for these difficulties by incorporating previously observed sequence variations into the reference. Such a representation may be referred to as a multi-genome. It is expected that the alignment of reads to a multi-genome would result in more accurate read placement and therefore improved variant detection.

1.1 Related work

With the availability of population variation annotations, a number of attempts have been made to include this information to guide the process of alignment and variant calling. For example, these annotations are available from a number of databases, which includes the Genome Reference Consortium (GRC). They not only host the latest build of the human reference genome but also track alternate loci; long sequences (~54 Kb) that provide an alternate representation of a locus found in an otherwise haploid assembly. These loci are mapped onto the assembly with matching flanking sequences, whereas all sequence between these flanks diverge. Such alternate loci may serve as additional alignment targets for any mapping tool that is aware and capable of using such information, e.g. BWA [10]. Utilization of these loci can lead to more accurate mappings, effectively improving variation detection.

Similar to the GRC, there are also databases that track common and smaller variants, such as dbSNP and SNPedia [11, 12]. The application of variation information from databases has been used to aid in the detection of variants. These methodologies require an initial alignment through conventional means, and subsequently matching mismatches with respect to the reference onto one of the variation databases, identifying the common variants [13]. However, this approach may still bias the results towards the reference that was used. Furthermore, any errors generated by the read alignment may be propagated to the succeeding step of variation detection, further limiting the applicability [14].

One of the first applications to represent variations in a multi-genome representation with the purpose of read alignment was GenomeMapper [15]. In their approach a fixed length hash-based index is build from one reference genome and a list of known variants obtained from other sampled genomes, where positions that are not explicitly annotated are assumed to be the same between genomes. The index maps subsequences of sequences that are approximately 10 nucleotides long, relative to their positions on the genomes. Alignment follows from sampling these subsequences in the reads, after which the index is scanned to identify exact matches to acquire seeds. These seeds are subsequently extended to obtain potential target sequences, and finally banded Needleman-Wunsch dynamic programming is used to align the sampled read against these targets, where the best scoring alignment is returned. At the time only a solution was presented for alignment against the small A. *thaliana* genome, and further scalability of the method has not been shown, which can partially be attributed to the high memory footprint.

The first multi-genome alignment implementation shown to be scalable up to human genome scale was BWBBLE [14]. They build a multi-genome representation from an initial reference genome and a supplied list of variations that describe edits upon this reference. In their notation, single nucleotide polymorphisms (SNPs) are generalized into separate characters using the IUPAC (16 letter encoding) alphabet, allowing any SNP to be encoded by a single character. Other types of variations such as indels are encoded as alternatives sequences in separate branching contigs alongside the reference. Although this encoding offers more flexibility and attempts to linearize as much sequence as possible for which most search algorithms are optimized. The most significant drawback also stems from this encoding, which increases the alphabet size 4-fold. This becomes apparent in the alignment itself for which a modification of Burrows-Wheeler transform (BWT) read alignment is used [16], generalized to support IUPAC encodings. Given this alphabet, the number of suffix array interval computations that have to be performed when backward searching becomes far higher, and leads BWBBLE to be on the order of a hundred times slower than traditional single reference BWT aligners. The discrepancy in speed makes it seem that adaptation of the method is questionable. Yet, the authors of BWBBLE do show that while the number of actual novel variants in the multi-genome increases linearly when including more variants from samples, this growth eventually subsides. Suggesting that at one point a break-point is reached where the use of BWBBLE would be more advantageous than using an aligner on each individual genome.

A more recent and still developing methodology is VG [17]. VG models a population variant aware reference system as a directed graph with sequence annotations. In this graph one reference genome forms the backbone while a supplied list of variants are encoded as edits upon this reference. Aside of handling the alignment of short reads against such a graph, they also support graph construction, variant calling, and general modification of the graph itself. Core components that form the driving force behind VG include: XG index, generalized compressed suffix array (GCSA2) index, graph to directed acyclic graph (DAG) conversion, and partial order alignment (POA).

The standard data representation of graphs in VG is optimized to allow efficient runtime for editing and transformation operations on the graph. To do this effectively a dynamic version of the graph is required by using a hash structure, which is not optimized for memory overhead. Consequently, loading the entire graph into memory is not feasible, hence another index is required to reflect a static and succinct representation of the dynamic graph. For that they introduce the XG index, which is based upon the Succinct Data Structures Library [18].

In the early development of VG, alignment relied on indexing the graph using GCSA [19–21]. This is a generalization of the extended Burrows-Wheeler transform (for labeled trees), extended for labeled DAGs, allowing for the indexing of arbitrary length paths. More recent work by the authors of GCSA has further generalized this indexing for de Bruijn graphs (GCSA2) [22]. A caveat being that rather than indexing all full length paths, only paths up to 128 bp long are indexed. This begins with paths that are $\leq$ 16 bp long, where these paths may be extended through a maximum of three doubling steps to generate a de Bruijn graph up to an order of 128, in which path queries of up to this length are supported in the input graph. Limiting the path length to 128 bp was necessary given the extension to allow for cyclicity in the graph which significantly increases the number of possible paths. However even in DAGs limiting the path length becomes a necessity to constrain the number of possible paths, otherwise leading to a combinatorial explosion of paths in complex graphs.

To allow for local alignment of reads onto graph structures VG uses POA. POA is a generalization of traditional pairwise sequence alignment methods based on Smith-Waterman (S&W) dynamic programming with support for graph alignments [23, 24]. In S&W deriving the best trace relies on considering all possible inbound positions in the recursion that determines the score for a new cell. In the current setting this implies that rather than considering nucleotides in a sequence which always have a single predecessor, vertices in graphs can have multiple and should therefore all be evaluated. Alignment with POA is only possible if the graph is linearly orderable, which can only be done in DAGs. This to ensure that all the cells of predecessors have been computed beforehand.

Sequence query matching in VG is driven by finding maximal exact matches (MEMs). These are matches between a query and a reference that cannot be extended in either direction along the query while still matching some sequence in the reference system. Super-maximal exact matches (SMEMs) are MEMs that are not contained by any other MEM. A seed-and-extend approach is used to perform local alignments, in which SMEMs are used for seeding. Given a set of sequencing reads, the first step is to derive SMEMs for a query relative to the GCSA2 index. Highly abundant SMEMs are filtered out by counting the number of occurrences in the index, avoiding those that have thousands of hits without extracting the specific hits from the index. SMEMs are consequently clustered by using an approximate distance metric on the graph. i.e. if the vertex id space is locally ordered, then vertices with nearby ids are likely to lie closer together in the graph. From each cluster a subgraph is extracted that includes a small neighborhood around it from the static XG index. These subgraphs serve as targets for the local alignment of a sampled read using POA. As was mentioned, POA is only possible if the subgraphs are DAGs, meaning that prior to the actual alignment the subgraphs are first transformed from cyclic graphs into DAGs by duplicating vertices and edges until the subgraph is fully unrolled.

Paired end reads may also be handled by using an approximate locality metric based on the vertex ids. For very long reads POA becomes prohibitive, to resolve this the reads are broken apart into bands of a fixed length with overlap between successive bands. These bands are aligned independently after which overlaps are trimmed from the alignments and the results are concatenated.

1.2 Contribution of this work

In this work we present a solution that exploits the additional information that is offered in multi-genome reference representations, while using established alignment methods that are available for single reference strings. Given a DAG representation of a multi-genome, the population graph, we reconstruct a path through this graph, which is best supported by the sampled reads. This path then represents a personalized linear reference genome, which can then ultimately be used to call variations.

As opposed to building a reference graph from a single backbone genome and a set of known variants, graphs are constructed based on assembled genomes using REVEAL [25, 26]. This way, graphs, retain information on haplotypes. Consequently, genomes that were used to build the graph may all be reconstructed by walking vertices that represent these genomes. This is not possible when building graphs based on a backbone genome and variants alone, offering the opportunity to exploit this information for alignment.

In the discussed alignment methods, one recurring step is the indexing of the reference system. Indexing a graph is more complicated than indexing a single string. This is due to the fact that the number of possible paths increases exponentially with length of paths. For this reasons arbitrary length indexing such as through the use of FM-indices and the BWT are not an option for graphs that are not trivially sized. Therefore, to make graph indexing tractable, it is essential to limit the length of paths. Determining the optimal length for these paths is dependent on the resolution that is required to capture enough overlap between vertices to be able to place reads accurately. Note that the maximal feasible path length to be indexed can vary between input graphs and is influenced by the number of variants encoded in the graph as well as the length and density of these variants.

A schematic overview of the procedure from input DAG to personalized genome is given in Figure 1. The approach used to index arbitrary length paths in DAGs relies on a process where overlap is captured between neighboring vertices by assigning subsequences. Such assignment is in some cases not possible without introducing ambiguity in the index, requiring local graph modifications. Novel in this indexing is that rather than capturing all paths, it is possible to restrict paths to remain in haplotype blocks. This is computationally favorable as it reduces the number of possible paths, while enabling longer paths to be indexed. This approach is also biologically supported since it retains the paths that were actually observed while omitting unlikely combinations of subsequences observed in the input genomes.

The index that is obtained captures all paths of a given length and may consequently be used as a target for read alignment. During this mapping we determine where a read may be best placed, and a read can either be contained in a single vertex or span multiple vertices. The alignment onto this index can then be projected back onto the input DAG, obtaining a read coverage distribution over the vertices and edges in it. This distribution is then used to guide the process of genome inference, and the generated genome can then be used as a target for read alignment.

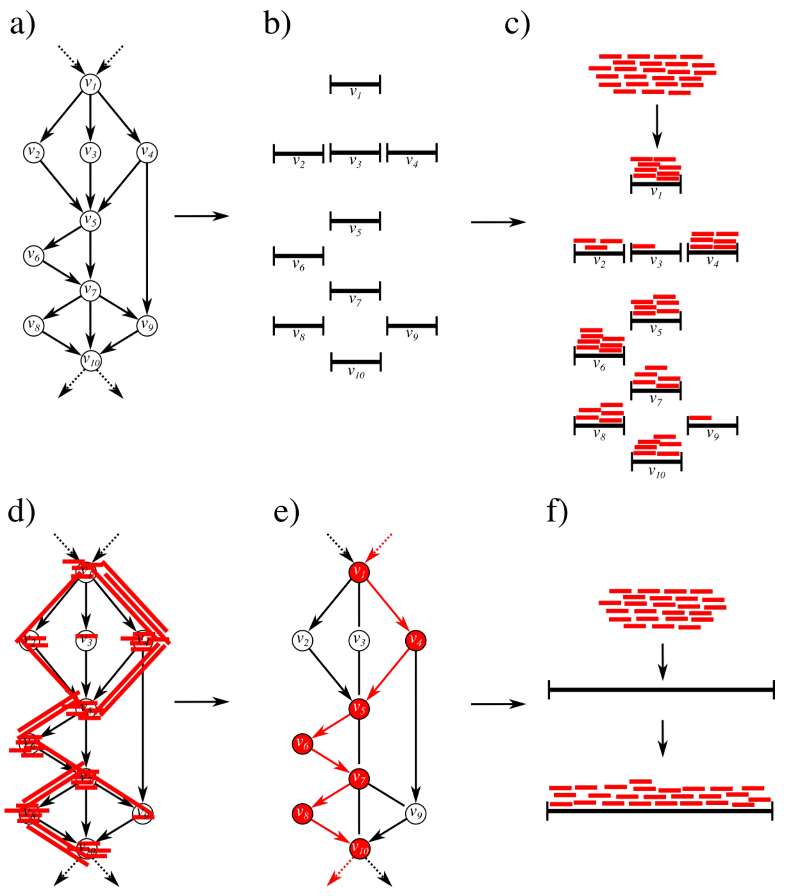


Figure 1: **a)**: An input DAG which represents a multi-genome or population graph. **b)**: Indexing of the DAG transforms it into an edgeless graph, a representation in which all paths up to a given length are captured. **c)**: Alignment of reads onto the edgeless graph, wherein mappings of reads can be contained in a single vertex or traverse a path of vertices. **d)**: The read alignment can be projected back onto the input DAG, yielding a coverage distribution of the reads over the graph. **e)**: Given the coverage distribution of reads on the graph, the most covered path can be inferred. **f)**: Reads can now again be aligned, this time to the inferred path, after which variants may be called.

2 Methods

2.1 Graph specification

A *directed acyclic graph* (DAG), $G = (V, E)$ is defined as a set of vertices $V = \{v_1, \dots, v_N\}$, where $N = |V|$, and a set of edges E, where $N - 1 \leq |E| \leq \frac{(N-1)N}{2}$. Each of these edges is an ordered pair of vertices $(u,\ v) \in E$, defined as an edge from vertex $u \in V$ to vertex $v \in V$.

A graph is *directed* if all edges are ordered pairs, in which a *source* vertex connects to a *target* vertex. This implies that any edge $(u,\ v) \in E$ is distinct from its complement $(v,\ u) \in E$, whereas in undirected graphs $(u,\ v)$ and $(v,\ u)$ describe the same edge. For each vertex $v \in V$ the number of incoming edges of this vertex is defined as the *in-degree*; $in(v)$, the number of distinct edges $(u,\ v) \in E$ for any $u \in V$. Conversely the number of outgoing edges of vertex v is defined as the *out-degree*; $out(v)$, the number of distinct edges $(v,\ u) \in E$ for any $u \in V$.

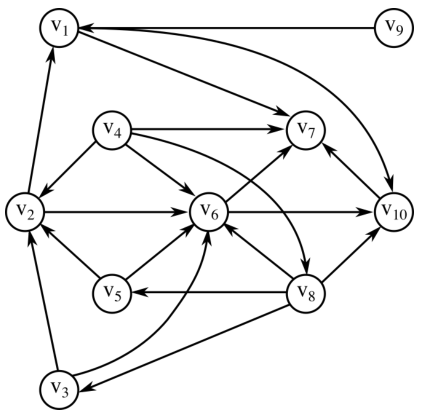
If a graph is *labeled*, a string $S = S[0,\ n-1]$ is assigned to each vertex $v \in V$, as $v.S$, where $n = |v.S|$, and each string is a sequence of characters over an alphabet Σ. The length of any such string in any vertex $v \in V$ is $1 \geq |v.S| <= N$, where N is the length of the largest genome encoded. Any subsequence of a string S is denoted as $S[i,\ j]$, important types of subsequences are prefixes $S[0,\ j]$ and suffixes $S[i,\ n-1]$, which capture the boundaries of any string S. Substrings of the form $S[i,\ j]$ with a length of k are defined as k-mers.

Any sequence of vertices is defined as a *path* $P = u_1 \dots u_{|P|}$ where $(u_i,\ u_{i+1}) \in E$ for all $i < |P|$. The string $P.S$ of a path P is defined as the concatenation of strings contained in the vertices, as $P.S = u_1.S \dots u_{|P|}.S$. A *cyclic* graph is a graph containing at least one graph cycle, where a cycle in a graph is a path P where the first vertex of the path also corresponds to the last. Any graph that does not contain such cycles is *acyclic*. A graph is *connected* when there is a path P between every pair of vertices, i.e. there are no unreachable vertices in a connected graph.

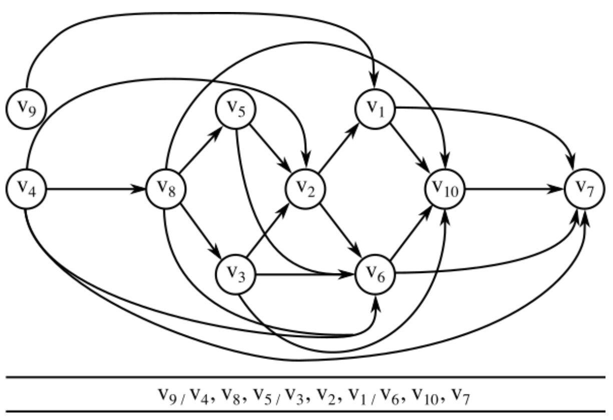
An *edgeless graph* is defined as $G^E = (V, \emptyset)$, and shares all defined vertex properties of a DAG. The interval $u.I$ of any vertex $u \in V$ in G^E is defined as the mapping of contained vertices in $u.I$ to their subsequences in the string $u.S = u_{I[0]}.S[i,\ j] \dots u_{I[|I|]}.S[i,\ j]$, relating the subsequences in u to their vertices of origin.

2.2 Topological ordering

The properties of a DAG allow for the determination of a topological sorting of the vertices in the graph. This sort yields a linear ordering of vertices such that for all edges $(u,\ v) \in E$, vertex u always precedes v in the ordering, as shown in Figure 2. Determining this ordering follows from the notion that in a DAG there should always be at least one source; a vertex that has an in-degree of 0. It may happen that there are multiple sources and in such cases the final ordering of these respective vertices is equivalent, i.e. there are multiple valid solutions for the topological ordering in the same DAG. The process by which the remaining vertices are sorted can best be thought of as process of eliminating source vertices. As the removal of all source vertices in a DAG along with their corresponding edges results in a new set of source vertices (as long as vertices remain in the graph). Gradually adding vertices to the ordering and removing them from the graph is guaranteed to result in a topological ordering of vertices that satisfies the aforementioned requirement. Acquiring such an ordering in a graph is favorable as it allows for linear time determination of longest/shortest paths in (un)weighted DAGs.



(a) Directed acyclic graph G



(b) Topological sorting of G

Figure 2: a): An input graph G with vertices $V = \{v_1, v_2, \cdots, v_{10}\}$. G has two sources $\{v_4, v_9\}$ and one sink v_7. **b)**: The resulting topological sorting Ts displays the linearized (left-to-right) ordering of the vertices in G as: $v_4.Ts = 0$, $v_9.Ts = 0$, $v_8.Ts = 1$, $v_5.Ts = 2$, $v_3.Ts = 2$, $v_2.Ts = 3$, $v_1.Ts = 4$, $v_1.Ts = 5$, $v_6.Ts = 5$, $v_{10}.Ts = 6$, and $v_7.Ts = 7$.

2.3 Indexing k-length paths in a DAG

Read alignment is complicated in a population graph representation because depending on the vertices that are walked, different arrangements of sequence are possible (paths). Although the alignment of reads onto sequence that is contained in vertices is straightforward, depending on the variation modeled in the graph, many reads will match sequence that stretches two or more vertices. To efficiently map reads, the indexing should therefore capture all such paths.

We propose a method that is capable of indexing all k-length paths in a DAG, i.e. it ensures that all overlaps across vertices and over edges are supported up to a length of k (see Figure 3). This is accomplished in a process where a DAG G (3_I) is transformed into an edgeless graph G^E (3_{III}). This transformation is made possible through the application of three distinct operations: *extension*, *duplication*, and *collapsing*. Capturing overlap between connected vertices is realized by *extending* the sequence on a vertex with a k-length prefix from a neighboring vertex (3_{IIa}). Conversely, here we also use the term extension to refer to the operation in which subsequence is prepended with the suffix of a neighboring vertex, depending on the directionality of the edge. The introduction of overlap into the graph occurs on a per edge basis. Each time overlap has been introduced, an edge can be removed, resulting in a simplified graph structure, until ultimately we end up with an edgeless graph G^E.

Simplification of the graph is crucial, because it allows for the extension of subsequence between pairs of vertices that could initially not be resolved without introducing ambiguity. This ambiguity refers to situations in which a vertex has multiple neighbors to which it can extend (3_{IIb}). To ensure that all vertices have a single candidate neighbor for extension it is necessary to modify the local graph structure by means of *duplication*. The duplication operation creates a copy of a vertex for each incoming edge, while keeping it connected to all of its previous outgoing neighbors (3_{IIb}). Although this operation increase the number of vertices and edges in the graph, it simplifies the local graph structure, such that extension can be performed again.

It may happen that after duplication subsequence extension is still not possible because the sequence length contained in the vertices of a pair is shorter than the path length for which is indexed (3_{IIc}). In those situations, a *collapse* operation is performed, merging the vertex pair, and concatenating the defined sequence.

Since the input graph is modified during the indexing it is necessary to track the origins of sequences S, the subsequences, and the respective vertex intervals therein to allow for a one-to-one mapping of indexed paths in G^E to G. This is made possible by initializing an *interval tree* for each vertex in the graph and consequently updating these trees as the indexing progresses.

Through repeated rounds of *extension*, *duplication*, and *collapsing* a graph is simplified until all edges are resolved. Resolving all edges is also the stopping condition of the indexing, signifying that an edgeless graph is generated that indexes all *k*-length paths. To ensure that the number of duplications and collapses are minimized during indexing, an initial pass is made over the edges of the input graph where only extension is allowed. Edges are evaluated, which either leads to extension or skipping of a particular edge. All further passes over the edges allow for the full modification of substructures in the graph through duplication and collapsing. Whenever a modification is performed, the first order neighbors of the respective vertices in the current edge are evaluated. This is because any simplication of the graph can simplify the operations necessary to resolve neighboring edges.

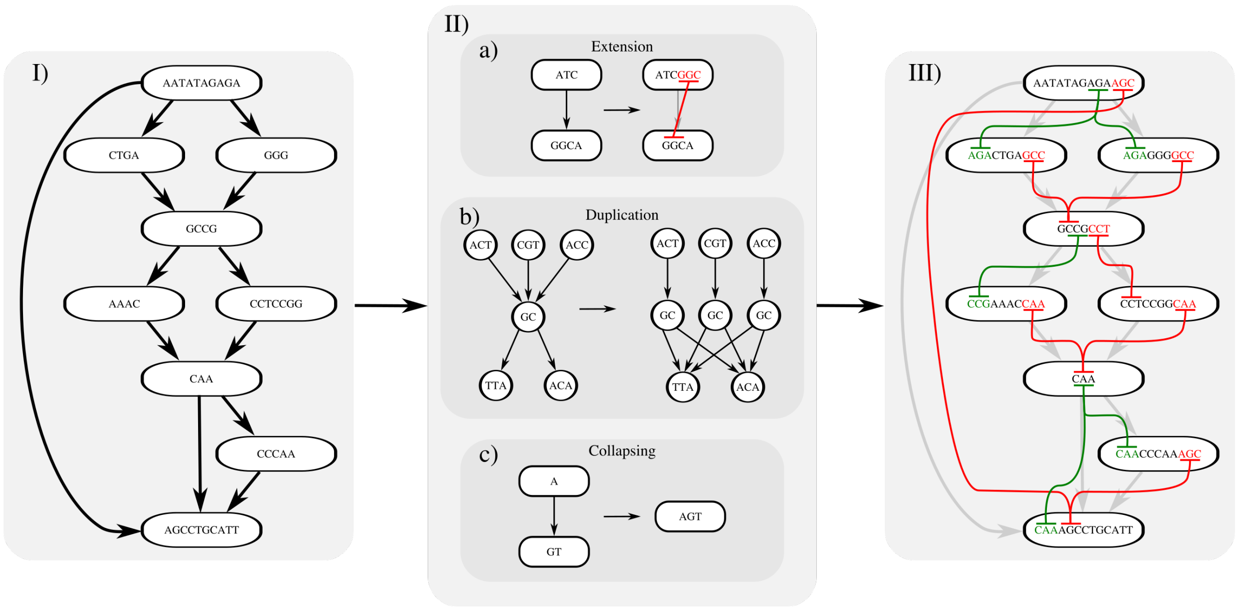


Figure 3: Indexing *k*-paths transforms an input DAG G into a edgeless graph index G^E, in this example $k = 4$. **I)**: Input DAG with sequence annotations on the vertices. **II)**: The *k*-path indexing procedure is based on three subroutines. **IIa)**: Extension; subsequence extension from one vertex to another to capture overlap. **IIb)**: Duplication; the duplication of vertices to eliminate ambiguity for extension. **IIc)**: Collapsing; the collapsing of chain-like graph motifs when subsequence extension is not possible. **III)**: The output edgeless graph index captures all 4-length paths in the original graph, colored lines denote the origin of assigned prefixes (green) and suffixes (red). A step-by-step guide of this process can be found in the appendix in Figure A1.

2.3.1 Subsequence extension

The extension of subsequence is only allowed if two requirements are fulfilled, these are the k-length and degree-requirement, which for a given edge $(u,\ v) \in E$ are defined as follows:

- if $k \leq |u.S|$ then $in(v) = 1$ or $k \leq |v.S|$ then $out(u) = 1$

To elaborate on this extension consider Figure 4, here the vertices u and v both capture enough sequence to allow for extension. This means that to capture the k-length overlap between the pair and resolve the edge $(u,\ v)$, two configurations for extension are possible. The first is to extend vertex v with a prefix originating from vertex u yielding $v.S = (u.S\,[|u.S| - k - 1,\ |u.S|]\,,\ v.S)$. The second is to extend vertex u with a suffix originating from vertex v, and is defined as $u.S = (u.S,\ v.S\,[0,\ k-1])$. In situations where both a prefix or suffix extension is allowed, the choice of extension is arbitrary.

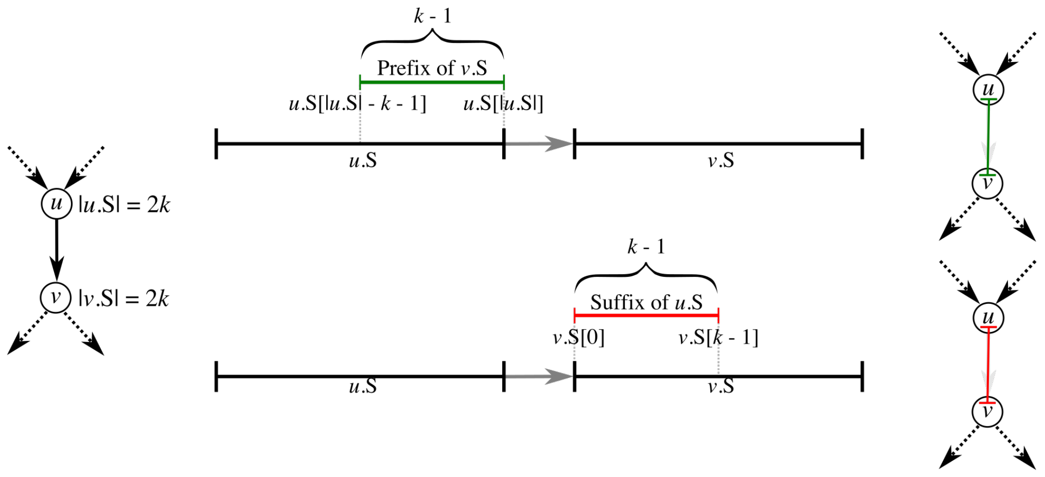
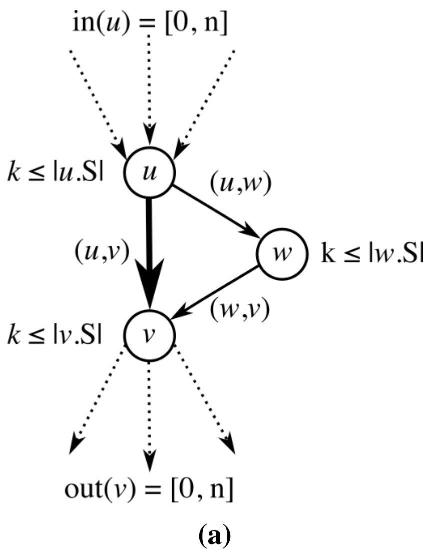


Figure 4: A pair of vertices u and v, where $|\{u,v\}.S| = 2k$. In this example two overlap extensions are possible. For clarity the starting and ending index positions are given for each subsequence relative to their sequence of origin.

Details of the degree requirement can be put into context of Figure 5a. In this example a network motif is given in which extension is only possible between the edges $(u,\ w)$ and $(w,\ v)$. Extension is not allowed between $(u,\ v)$, because $out(u) = 2$ and $in(v) = 2$. This means that there is no unambiguous assignment of subsequence between these vertices, which would otherwise lead to missing paths in the indexing. If one of either vertex fulfills this degree requirement ($out(u) = 1$ or $in(v) = 1$), this ambiguity would be resolved and extension becomes possible. The only way to accommodate for that is to either resolve edge $(u,\ w)$ first or $(w,\ v)$, as shown in Figure 5b.



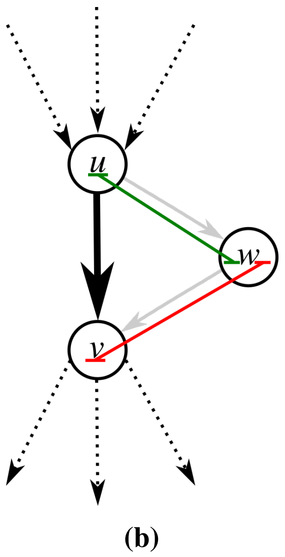


Figure 5: a): Given the edge $(u,\ v)$ immediate extension is not possible unless either edge $(u,\ w)$ or $(w,\ v)$ are resolved first. **b)**: By solving either (or both) edges $(u,\ w)$ and $(w,\ v)$, it becomes possible to solve $(u,\ v)$. This can be done by assigning a prefix originating from vertex u to w or by assigning a suffix to w originating from v.

Extension for a given edge $(u,\ v) \in E$ in G can be generalized as:

```
if out($u$) == 1 and $k \leq |\ v$.S$\ |$ then
    Assign suffix to $u$.S from $v$.S as:
    $u$.S = ($u$.S, $v$.S[ 0, $k$ - 1 ])
    Remove ($u$, $v$)
else if in($v$) == 1 and $k \leq |\ u$.S$\ |$ then
    Assign prefix to $v$.S from $u$.S as:
    $v$.S = ($u$.S[ | $u$.S | - $k$ - 1, | $u$. S | ], $v$.S)
    Remove ($u$, $v$)
else
    Skip
```

2.3.2 Vertex and edge duplication

There are situations where extension is not possible, even if neighboring edges are resolved. In such cases multiple edges enter or exit a vertex, causing the subsequence extension to be ambiguous. This means that there are multiple subsequences that can be assigned to a single vertex (see Figure 6).

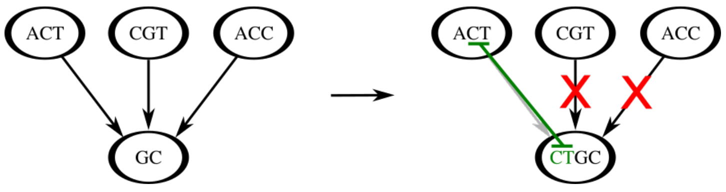


Figure 6: A subgraph in which there are three ways to assign a prefix to the target vertex, for $k = 3$. Assigning one of the prefixes to the target, leaves two edges unresolvable.

To overcome such situations it is necessary to modify the local graph topology through duplication of vertices and edges, shown in Figure 7. The example can be resolved by duplicating the target vertex twice and then assigning two of the edges connected to the original vertex to these duplicates.

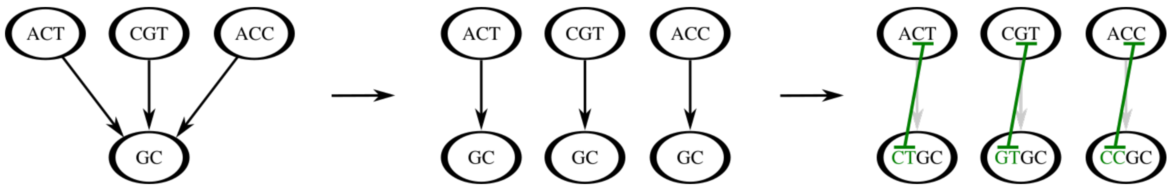


Figure 7: Resolving ambiguity follows from duplicating the target vertex and distributing the edges, as such that each duplicate has one incoming edge.

The duplication can be generalized as follows, where for a given edge $(u,\ v) \in E$ the direction of duplication and the total number of duplicates is decided by $min(in(u)-1,\ out(v)-1)$, where $in(u)-1 \geq 1$ or $out(v)-1 \geq 1$. Duplicates inherit all properties of their parent and also remain connected to the edges in the opposing direction. For more details refer to Figure 8.

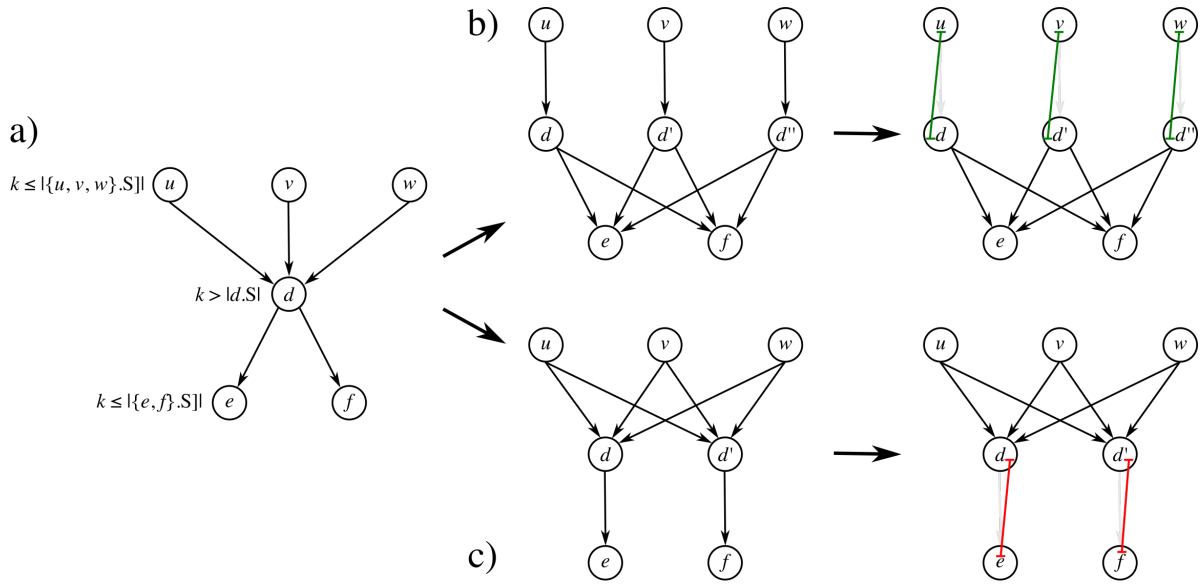


Figure 8: a): A subgraph where subsequence extension is not possible in any configuration without introducing ambiguity in the paths. This ambiguity is caused by the lack of sequence contained in vertex d, and must therefore be duplicated. Vertex d can either be duplicated in the incoming or the outgoing direction, requiring $in(d)-1=2$ or $out(d)-1=1$ duplicates respectively. For completeness both directions of duplication are shown. **b)**: Resolving d in the incoming direction requires two duplicates, denoted as d' and d''. The incoming edges $(v,\ d)$ and $(w,\ d)$ are reassigned to these duplicates, whereas the outgoing edges of d are all inherited. After this, the ambiguity in incoming direction is resolved and prefixes can be assigned to d, d', and d''. **c)**: Resolving d in the outgoing direction requires 1 duplicate, denoted as d'. The outgoing edge $(d,\ f)$ is assigned to this duplicate, whereas the incoming edges of d are all inherited. After this, the ambiguity in incoming direction is resolved and suffixes can be assigned to d and d'.

One can argue that it is also possible to duplicate a vertex in both directions simultaneously to resolve ambiguity. In the example above this would require $in(d) \cdot out(d) - 1 = 5$ duplicates after which all extensions can be completed. However, by doing this unnecessary duplication operations are performed. If we consider duplications by direction, we simplify the graph and may make further duplications unneeded or less expensive. For instance, in Figure 8c, the subgraph can be resolved after the first duplication of d, by duplicating two of the vertices in $\{u,v,w\}$ once, requiring 3 duplications in total.

Note that if haplotype information is available, only the k-length paths supported by haplotypes have to be captured. In terms of graph operations this can have a significant influence on the number of duplications required to index the graph. The intuition here is that if duplication is necessary there should be as many duplicates as there are haplotype paths traversing a given vertex. This contrasts the extension with duplication, in the sense that the former is unaffected by the inclusion of haplotype information. This is obvious because the haplotype relation is already implicit in vertex pairs, where the set of haplotypes contained in the source is always a subset of those in the target. In duplication this is different, since the ambiguous vertex is a junction wherein haplotypes come together and consequently diverge again in the outgoing direction. This divergence will often group haplotypes together, because there is high similarity in the sequence of genomes. Furthermore, the variants in them often have linkage if they are in proximity, making this grouping more likely. The grouping can be used as a means of constraining duplication, ensuring only haplotype supported paths are indexed. Groupings can be identified through the intersection of the sets of haplotypes contained in the incoming edges with those on the outgoing edges. This may be better understood given the example in Figure 9.

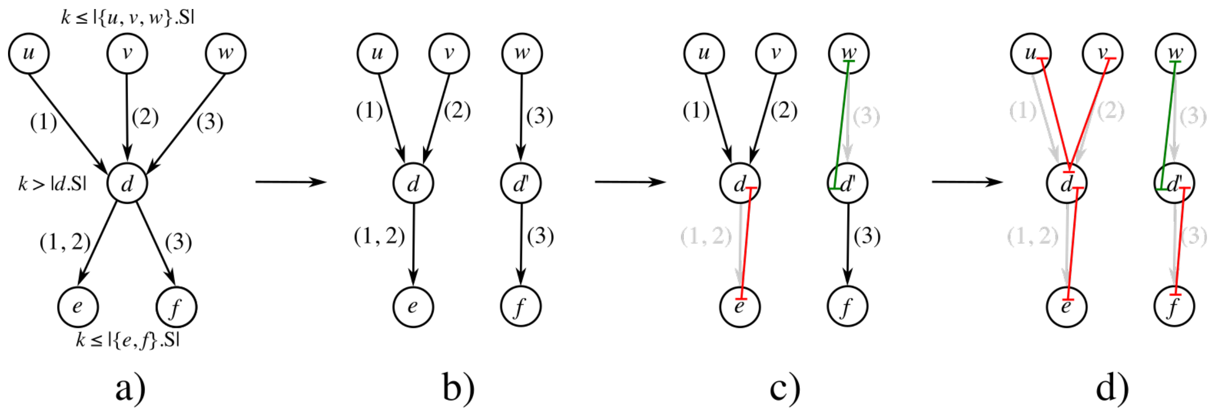


Figure 9: **a)**: Subgraph with the haplotypes: $\{1,2,3\}$. In this situation vertex d causes ambiguity in the extension, meaning it has to be duplicated to resolve this. The duplication can be simplified, because there is a grouping of the haplotypes $(1,2)$ in vertex e, which were previously disjoint in vertices u and v.**b)**: The duplication is guided by this grouping, identifying the groups by intersecting the sets of haplotypes on the incoming edges with those that are outgoing. This yields two groups $(1,2)$ and (3), hence only one duplication of d is required to isolate them. **c), d)**: Following duplication the subsequence extension is possible again, resolving the subgraph.

2.3.3 Vertex collapsing

It may happen that after duplication an edge $(u,\ v) \in E$ where $out(u) = 1$ and $in(v) = 1$ can still not be resolved, because $u.S > k$ and $v.S > k$, (as shown in Figure 10). If there is not enough sequence in either vertex, an overlap can not be expressed through subsequence extension. The solution is to collapse one of either vertex into the other, concatenating or prepending their sequences together. The receiving vertex inherits all properties of the transmitting vertex, which is removed from the graph. Direction of collapsing is guided by minimizing the number of edge reassignments required to complete it, given by $min(in(u),\ out(v))$.

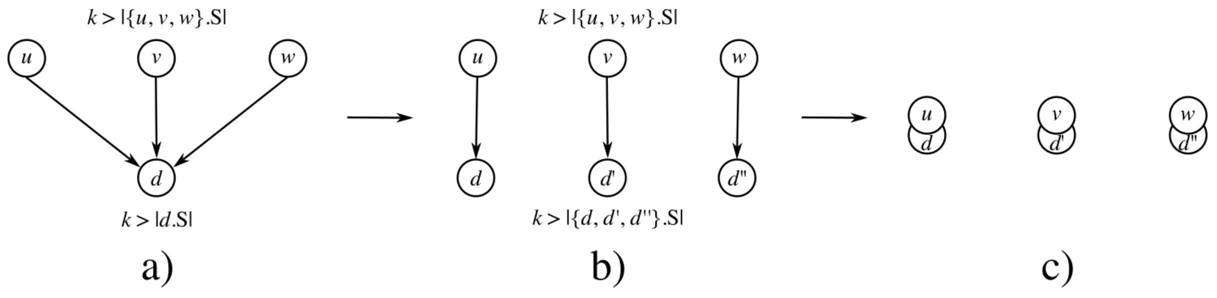


Figure 10: a, b): The duplication of vertex d is insufficient to resolve the edges, because there is not enough sequence to capture an overlap between the vertex pairs. **c)**: To solve these chain substructures a collapse operation is performed which merges each pair into a single vertex.

2.4 Read alignment to an edgeless graph

With the construction of index G^E it now becomes possible to directly align reads with established mapping methods. Such an alignment can be interpreted as a one-to-one mapping of reads to k-paths in the input graph G that are embedded in index G^E. To clarify, G^E captures all overlap between vertices in graph G, meaning that any read may be aligned onto a single vertex or on a path, spanning vertices. Vertices in G^E no longer describe single vertices with sequence, but now represent *intervals* over which sequence is distributed on neighboring vertices, where the sequence length that an interval describes is always $\geq k$. We know what subsequence belongs to which vertex in the interval, because of the associated interval tree that is constructed for each vertex in G^E.

To clarify the interpretation of intervals consider Figure 11, in which a subgraph of G^E is given, and an interval of it is projected to G.

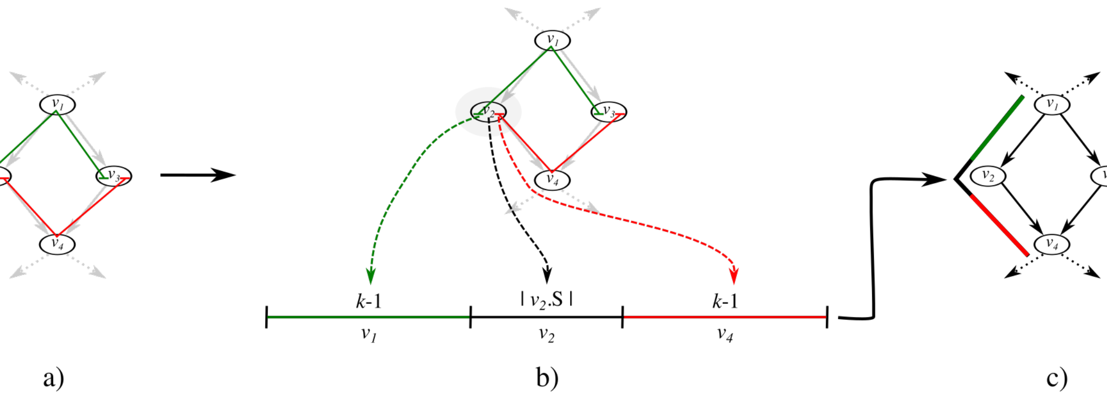


Figure 11: **a)**: Edgeless graph G^E. **b)**: Vertices v_2 and v_3 are both prefix and suffix extended by v_1 and v_4 respectively. This means that the interval contained in vertex v_2 describes the sequence in vertices v_1, v_2, and v_4 as $k-1$, $|v_2.S|$, and $k-1$. **c)**: Projecting the interval on vertex v_2 onto the graph G, describes the path $v_1 \rightarrow v_2 \rightarrow v_4$.

The read alignment onto G^E relies on the application of BWA-MEM to map single-end reads, for which the index is first converted to a format that is readable by BWA. Standard parameters were used with BWA, there being no constraint in the number of mismatches allowed in reads. Multi-mapping reads (multiple locations on the reference with equivalent alignment scores given a read) are handled by random selection, furthermore only the primary alignment of a read is considered.

2.5 Projection to input DAG

Given a mapping of reads against each interval in G^E, we are now interested in projecting that information back to the original graph G, to obtain a coverage distribution of reads on edges and vertices. This is possible, because each interval in G^E has an associated interval tree that maps the relative positions of the vertices in that interval back to G. Figure 12 describes how this read mapping can be translated back to the original graph G.

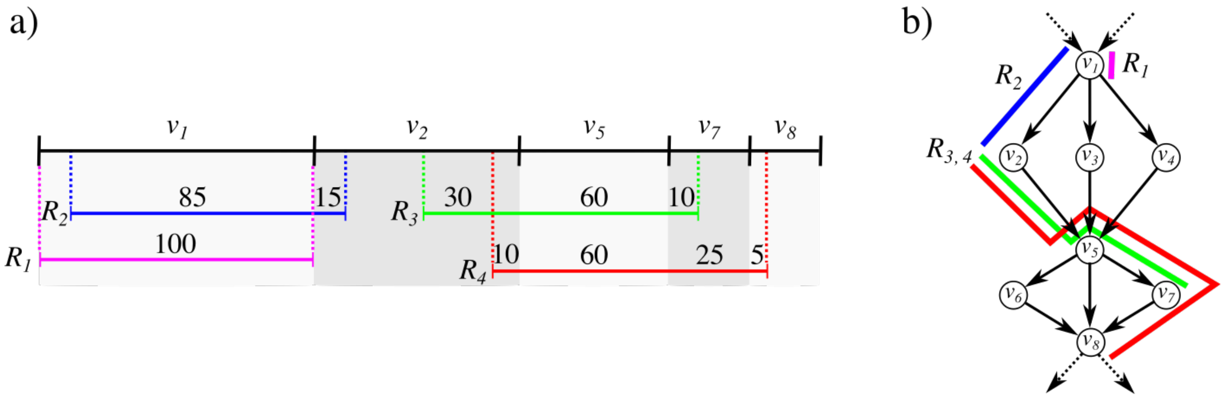


Figure 12: a): Four $k = 100$ reads $R_{1,2,3,4}$ are found to be aligned to the same interval of vertices v_1, v_2, v_5, v_7, v_8 at different start and ending positions within this interval. Only one of the reads (R_1) is fully contained in a single vertex, whereas the others all describe paths between vertices of varying edge traversals: R_2: 1 edge, R_3: 2 edges, and R_4: 3 edges. **b)**: A subgraph of G, wherein the reads can according to their relative positioning within the interval alignment in **a)** be placed to their respective locations in G.

As noted in the above Figure, a read that covers an interval may be contained entirely in a single vertex or span multiple vertices. The contribution of a read is tracked as coverage on each vertex and edge, where the relative contribution on vertices is assigned based on the number of nucleotides that are spanned in each vertex. For example, in Figure 12 read R_2 contributes 85 and 15 nucleotides to v_1 and v_2 respectively, which given a read length of 100 scales this to 0.85 and 0.15.

Weighting the contributions on edges is approached differently, given that if a read maps, the sequence is only contained on the vertices it spans and not on the edges. Meaning that the mapping of reads on paths has a different significance. For a given edge $(u,\ v) \in E$ over which a read spans both vertices, it describes the *transition* from u to v. This means that the contribution that reads have on edges, should be indicative on how often particular vertex pairs are observed together. Which is why the contribution of a read on edges remains constant regardless of how many are crossed.

2.6 Path inference

After all alignments are projected back onto G, a coverage distribution is obtained across the graph, which allows for the inference of a path through it that best reflects the sampled reads. Because G is a DAG, it is possible to obtain a topological ordering of the graph which can be used to obtain the most supported path of the graph in linear time. This is possible through a simplification of the Bellman Ford algorithm modified for longest paths and DAGs, where for a DAG it is only necessary to traverse each edge only once.

The workings of this method can be better explained in context of Figure 13, in which the inference is based on the premise of finding the longest 'distance' between source and sink. Yet this is only possible if the distance between the source and the predecessors of the sink are known, and recursively the distance to the predecessors of these predecessors. Where the distance between any two vertices is a summing over the coverage contained in the targeted vertex and connecting edge with the distance of the source vertex. For example, if the most supported path between v_1 and v_4 in Figure 13 has to be obtained, only two paths have to be considered: $v_1 \rightarrow v_2 \rightarrow v_4$ and $v_1 \rightarrow v_3 \rightarrow v_4$. To know whether one path should be chosen over the other, the distances from v_1 to v_2 and v_3 should first be determined. After which the distances from v_2 and v_3 to v_4 have to be found. Since v_4 has two predecessors, we need to maximize over them to obtain the predecessor with the greatest summed distance.

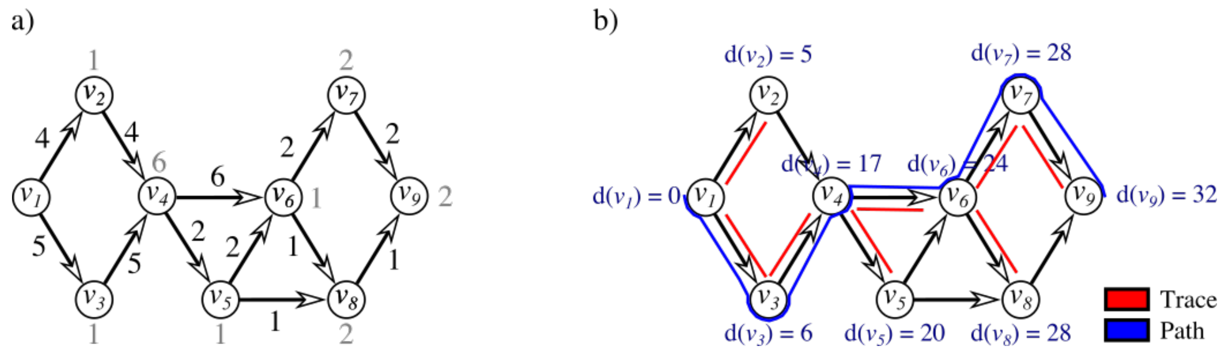


Figure 13: **a)**: The result of projecting aligned reads back onto G, where every edge and vertex has coverage from reads. **b)**: Determining the most supported path is based on evaluating the distance between source and sink and all vertices that precede this sink, which are all co-dependent. Distance between u and v is defined as the sum of the coverage on v and $(u,\ v)$ and the current distance on u. In the event that v has multiple predecessors a maximization is performed to find the highest scoring predecessor, which is then tracked as a trace. Obtaining the distance between source and sink allows for a traceback of the most supported path based on following the traces.

This displays that a topological ordering is necessary to ensure that dependencies on distances in predecessors are evaluated beforehand, meaning that calculation of distances should start at the source vertex of the graph. When maximizing for distance, the highest scoring predecessor is always tracked as a trace. This is to ensure that after finding the distance between source and sink, a traceback can be performed to obtain the most supported path.

Path extraction can finally be generalized as follows, here the *distance* and *trace* are considered attributes for any vertex $v \in V$, and W denotes the observed coverage in both edges and vertices. W_E and W_V are introduced as a means of obtaining a weighted sum of vertex and edge coverages.

```
**Most supported path**($G$)
  tsort $\leftarrow$ **Topologically sort** $G$
  tsort[first]**.distance** = 0
  **for** each vertex $v \in V$ in tsort[second:last] **do**
    $v$**.distance** = **max**$(u, v) \in E$ { $u$**.distance** + $(W_E \cdot (u, v).W) + (W_V \cdot v.W)$ }
    $v$**.trace** = $\hookleftarrow$ **max**$(u \in V)$
  path $\leftarrow$ **Traceback** from tsort[last]
  **return** path
```

Although a path may now be obtained efficiently, there are still problems in doing so in multi-genomes. This is especially the case in highly repetitive regions, where the coverage of uniquely mappable reads is sparse and distributed over parallel paths, in which the path extraction biases towards dense regions in the graph. To illustrate this, consider Figure 14 in which two parallel paths are shown, where one is enriched with variants and the other skips over this variation. Incorrect paths may be inferred in situations like this if cumulatively the coverage is higher than the shorter correct path, even if in this dense region there are portions that are uncovered.

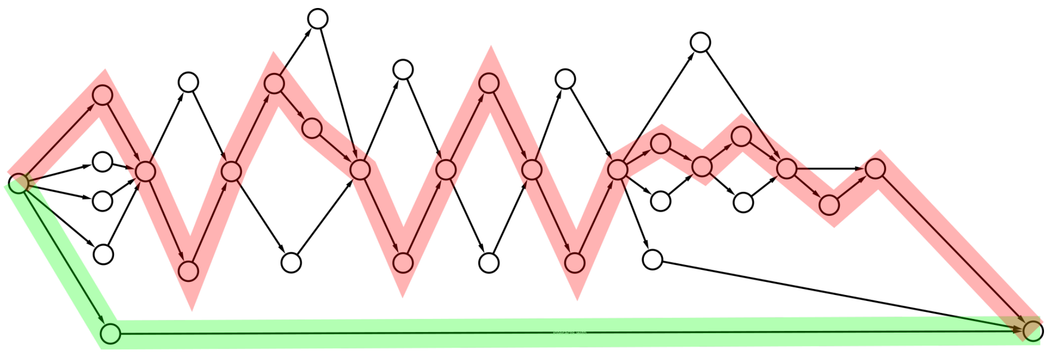


Figure 14: Graph topology in which the source vertex branches into a number of parallel paths, only to join again in the sink. The green path is noted as the ground truth for a given genome, the red path is what is inferred.

To account for this bias, a scheme is introduced that penalizes coverage on vertices and edges that diverge too much from an expected read (R) coverage distribution (C), where $|R|$ is the read length. This is guided by the deviation of the observed coverage $u.R$ and $(u,\ v).R$ in a vertex or edge obtained from the coverage $u.W$ and $(u,\ v).W$ given an expected coverage exp_u and $exp_{(u,\ v)}$. The latter being derived from the theoretical coverage that a aligned read set could provide on the longest path in the graph. Vertices and edges are penalized if $u.R < exp_u$ and $(u,\ v).R < exp_{(u,\ v)}$. The degree of penalization is determined by the height of deviation, penalizing more stringently if the difference is larger.

For each vertex $u \in V$ and each edge $(u,\ v) \in E$ the observed coverage is defined as:

$$u.R = \frac{u.W \cdot |R|}{|u.S| \cdot C} \textbf{ and } (u,\ v).R = \frac{(u,\ v).W \cdot |R|}{(k-1) \cdot C}$$

Where the coverage is penalized when $obs < exp$ in vertices and edges respectively as:

$$u.W = u.W - |u.S| \cdot (exp_u - u.R) \textbf{ and } (u,\ v).W = (u,v).W - (k-1) \cdot \left(exp_{(u,\ v)} - (u,\ v).R\right)$$

2.7 Measuring performance

To evaluate the performance of path inference and read alignment, a number of measurements are taken. The quality of an inferred path can be expressed in terms of its deviation from a given ground truth. Meaning that if a reference assembly is available, that is build from the reads used to sample a path in graph G, a direct comparison is possible. This difference is indicated by the presence or absence of (un)expected sequence in the path relative to the assembly, as:

- True positive: a sequence that is expected to be in the path.
- True negative: a sequence that is expected to not be in the path.
- False positive: a sequence that is included in the path, but is foreign from the ground truth.
- False negative: a sequence that is not in the path but is expected in the ground truth.

These four measures are determined on a per nucleotide basis for each inferred path. The false positives and negatives are related, given that if there is an over-representation of the former it implies that in the inference the longest path is taken rather than the shorter at certain points. The opposite of this holds true for an over-representation of false negatives.

The read alignments generated by BWA are summarized in more general terms, for which we use the number of *mismatches*, *unmapped reads*, and *multi-mapping reads*. The number of mismatches reflects how often a mismatch has to be introduced to allow for the mapping of a read to the reference. This can only be given in absolute counts, given that the number of mismatches required for a read to be mappable is variable. The number of unmapped reads represent the reads for which no sequence context is found on the reference, making their contribution null. Along with the number of mismatches they give an indication of the divergence of sequence content in the reads relative to the reference system. Where the mismatches describe smaller variations which may still be resolved by reads, and the unmapped reads refer to larger structural variations. The number of multi-mapping reads, describe reads for which multiple equally good mappings have been found on the reference, making alignment ambiguous. Whenever there is a multi-mapping of a read in BWA, one of the mappings is chosen randomly. The number of multi-mappings does therefore not denote the total multi-mappings found for each read, but rather the reads for which at least more than one mapping is found. This can be used as an indicator for the relative ambiguity in the alignment against a reference.

3 Results

3.1 Data specification

M. *tuberculosis* (TB) was chosen as a subject given its small genome size of ~4.4Mb. For this purpose a number of assembled genomes were available which could function as benchmarks. A population graph was build from 20 TB genome assemblies using REVEAL with standard parameters generating a graph with 25,008 vertices, 34,311 edges and 4.8Mb of sequence. Out of these genomes H37Rv_BR was selected as a reference to be used as a baseline in analyses. The reads used to assemble these genomes were not available. In place of that, error-free 100 nucleotide reads were sampled from the 20 genomes in the graph, and also from 5 other genome assemblies, obtaining for each an average coverage distribution of ~100. Note that the read alignment relied on matching 100 bp length reads, hence the index size may from this point onward be assumed to be $k = 100$ unless otherwise specified. The experimental data (Illumina) that was used is accessible from the NCBI Sequence Read Archive by identifier: PRJNA183624 [27], from which the read sets of 7 samples were used. Two indexes were build given the input graph G, one that captures all k-length paths denoted as G^E, and another that is restricted by including haplotype information, G^E_H.

3.2 Indexing k-length paths

The difference between indexing of G^E and G^E_H can be expressed in terms of the graph operations required to fully index the graph, the total sequence in the index, and the distribution of subsequence extensions over vertices. To better describe the relation of path length and indexing complexity, the same indexes were build again but now for $k = 20$. The measurements of both path lengths indexes are shown in table 1.

Table 1: Statistics of the 20 genome graph G before and after indexing 100- and 20-paths for G^E and G^E_H.

		100-paths		20-paths	
	G	G^E	G^E_H	G^E	G^E_H
Vertices	25,008	+895.96%, 249,070	+14.74%, 28,694	+3.94%, 25,994	+1.72%, 25,438
Edges	34,311	0	0	0	0
Nucleotides	4,766,011	60,066,776	8,623,685	5,466,142	5,431,370
Duplication op.	-	229,627	4162	1022	433
Collapse op.	-	5565	476	36	3
Prefix	-	15.53%, 38,670	42.23%, 12,118	31.19%, 8107	32.35%, 8228
Suffix	-	0.88%, 2196	4.15%, 1190	2.79%, 725	3.40%, 864
Prefix and Suffix	-	82.08%, 204,438	41.84%, 12,005	51.95%, 13,503	49.96%, 12,710
No extension	-	1.51%, 3766	11.78%, 3381	14.07%, 3659	14.29%, 3636

The number of graph operations required to index 100-paths decreases by a factor of 55 and 12 for duplication and collapse operations respectively in G^E_H. This is also reflected in the increase of vertices, where in G^E there is almost an order of magnitude increase in vertices with respect to the input graph, whereas this increase is only marginal in G^E_H. The amount of vertices translates into the number of paths, and by extend the amount of sequence that is indexed by both approaches, where G^E captures much more sequence than G^E_H. This gives an indication on why indexing longer paths is more difficult in graphs, where through combinatorics an explosion is observed in the number of paths. The skewed distribution seen in the assignment of both prefixes and suffixes in G^E can be attributed to the increase in duplications. Because after a duplication is completed, it generally becomes possible to perform subsequence extension in at least one direction, if not both.

The effect of path length in indexing becomes clear when comparing $k = 100$ to $k = 20$, wherein the differences between G^E and G^E_H become marginal. Important to note is that while an increase in path length is directly related to the exponential growth in paths, the root cause of this is actually the size of variants and their densities. To put this into context, consider if a graph is build where all the vertices are guaranteed to exceed the length of k and there is never ambiguity in extension, duplication would then never be necessary. Meaning G^E_H would become equivalent to G^E.

In principle, it is possible to index paths equivalent to the longest sequence encoded in the graph. This would generate an index that contains all possible path combinations of the original graph, encoded in a number of vertices equivalent to this number of combinations. Building such an index without a haplotype constraint is intractable given the problem of exponential growth, yet it was possible when including haplotypes. This generated an index with 20 vertices each describing one of the input genomes, and displays that the indexing can be interpreted as a reversal of the genome alignment used to build graphs, which may be better understood in the context of haplotype constraints. Because as the path length increases, the more substructures in the graph are pulled apart into parallel paths describing single genomes, effectively simplifying the graph.

3.3 Evaluation on synthetic data

3.3.1 Performance of genome inference

Before the path performance in synthetic data was evaluated, three parameters where first optimized. Two of which are the cutoff values for penalization of coverage exp_u and $exp_{(u,v)}$ in vertices and edges respectively, and the last being the weighting of vertex and edge coverages when determining the distance in neighboring vertices for path inference, as W_E, W_V. An estimate of these three parameters was obtained through a grid search on $[0,1]$ with step size 0.1, minimizing the sum of false positives and negatives. This search yielded the parameter set $exp_u = 1.0$, $exp_{(u,v)} = 0.6$, and $W_E = 0.9$, $W_V = 0.1$ for both G^E and G^E_H. Of note is the 9 to 1 weighting for edges and vertices. Indicating that if a read aligns, it is more informative if the mapping is on an edge, given that this actually provides support of a short path through the graph rather than describing a single vertex in it.

The read sets of all 25 genomes were aligned to both G^E and G^E_H. Given the estimated parameters, paths were inferred from the alignments, where for 20 of the samples the ground truth could be compared to the paths in terms of the defined performance metrics. The results of this can be found in Figure 15, where the deviation of the ground truth with respect to H37Rv_BR is also included.

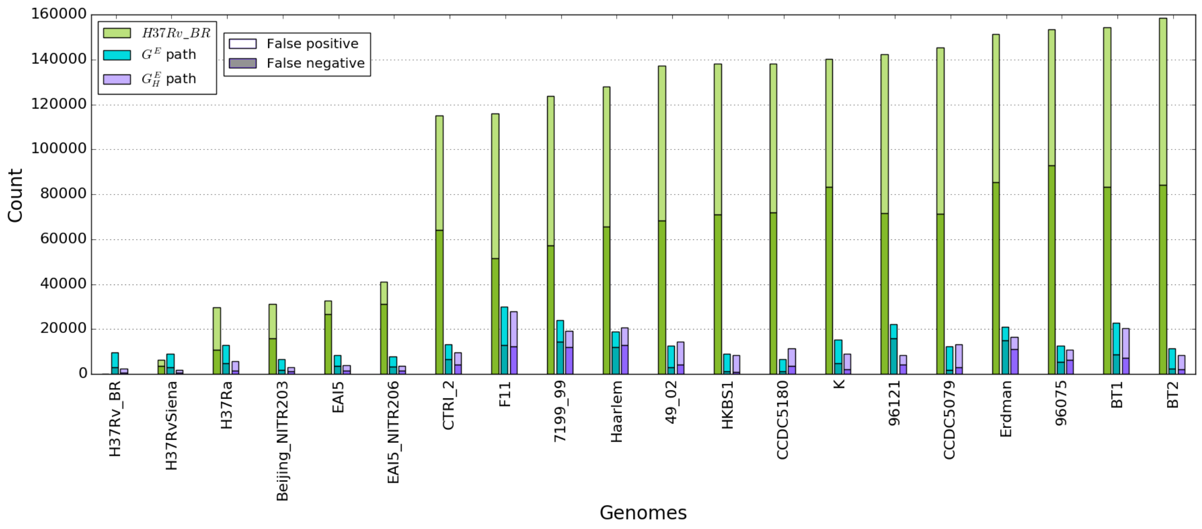


Figure 15: Evaluation of inferred path quality by false positives and negatives for 20 genomes, comparing the paths obtained from aligning reads of a sample to G^E and G_H^E to the ground truth of this sample. Additionally, the difference between ground truth and H37Rv_BR is included for each sample, which also decides the ordering in the figure.

The performance of inferred paths from the alignments on G^E and G_H^E are always better than the reference (excluding H37Rv_BR against H37Rv_BR). The reason that path reconstruction is not perfect in any of the samples, is related to both the length of reads and paths. The inference can in some cases not be resolved correctly because alignment is ambiguous in regions with repeat content, in which reads cannot span these repeats.

G^E and G_H^E offer similar performance, although G_H^E does better on average, which may attributed to the increase in multi-mapping reads on G^E as a result of indexing more paths. In some samples path inference is more difficult, an example of this is F11, which has the worst performance. The complicating factor has already been mentioned, which is the repeat content of the target genome, and the read length.

3.3.2 Alignment of reads to reference representations

To better understand the differences between the paths that were obtained from the index mappings, the alignments themselves were investigated. The read sets of all 25 samples were aligned to 6 targets, these are: the sampled genome (self), the reference H37Rv_BR, the indexes G^E and G_H^E, and the paths obtained from the alignments of these two indexes. The results of which are shown in Figures 16, 17, 18, 19 in terms of the number of mismatches, unmapped reads, and multi-mapping reads.

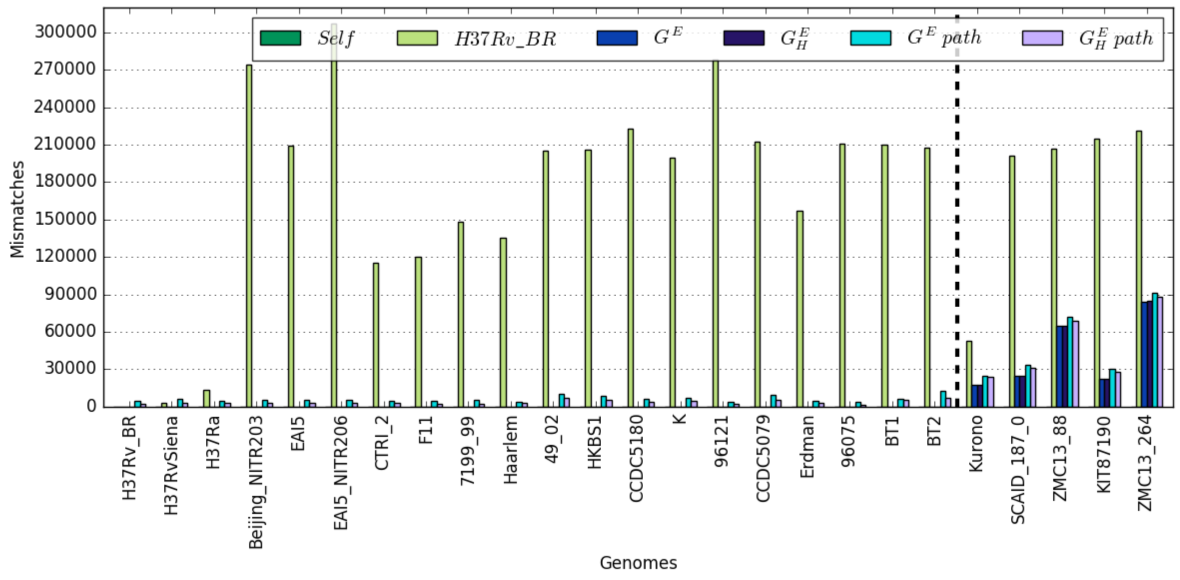


Figure 16: The total number of mismatches for a read set given each of the 6 reference systems. Input and hold-out genomes are separated by the black vertical dotted line.

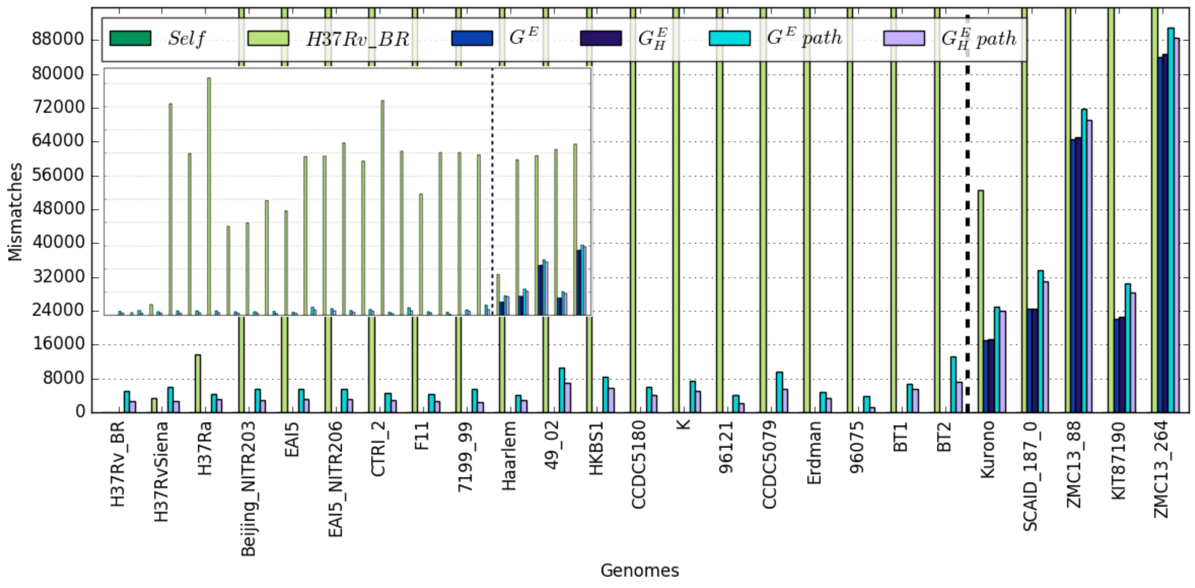


Figure 17: Same measurements as in Figure 16.

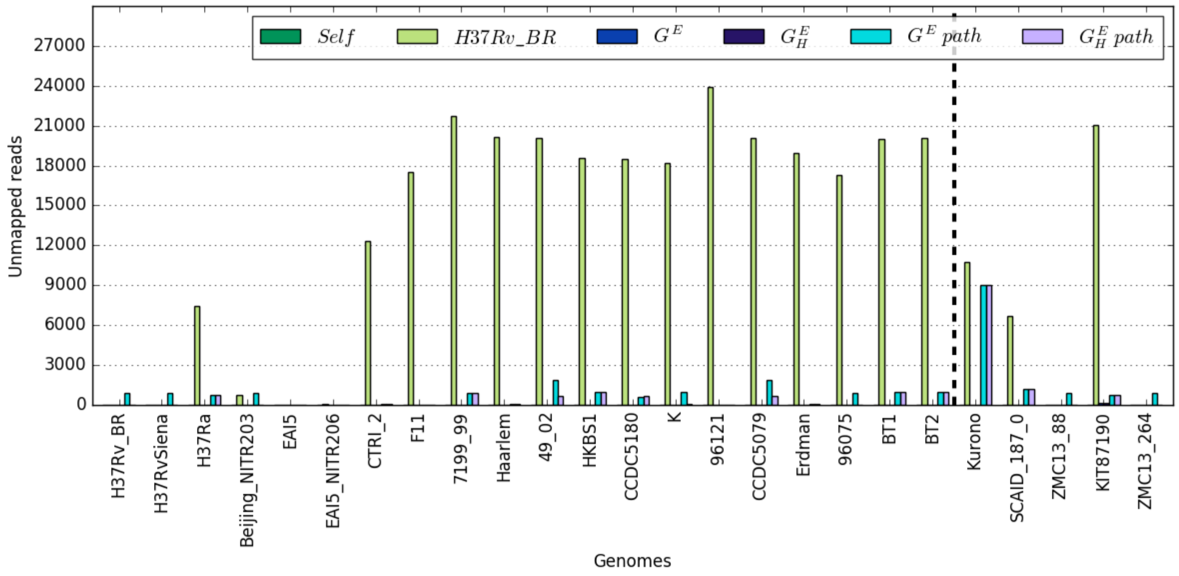


Figure 18: The number of reads for which no mapping was found.

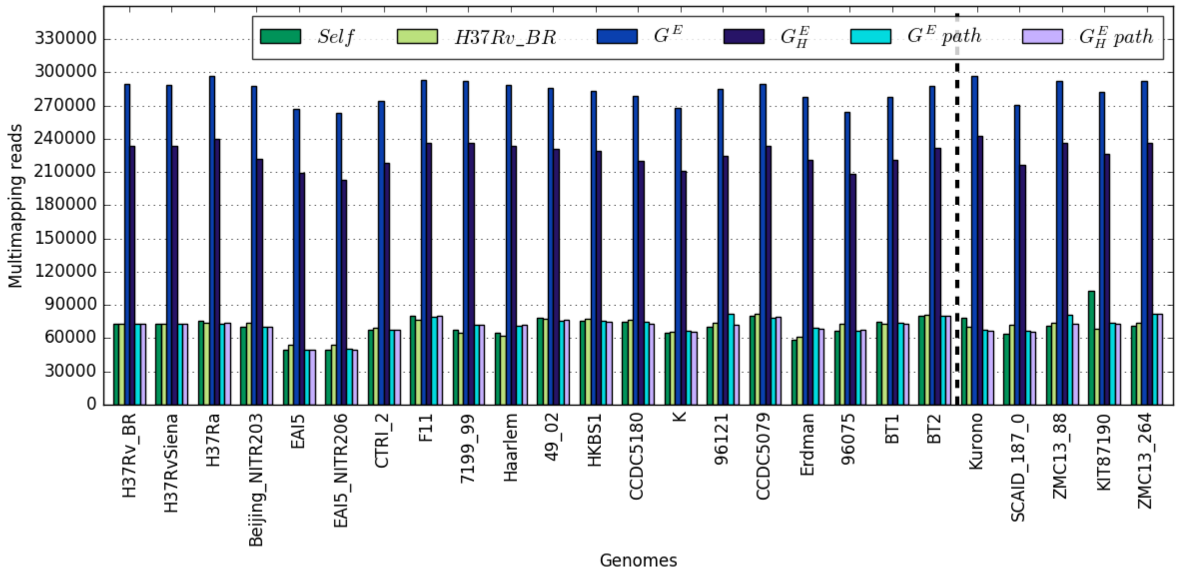


Figure 19: The number of reads for which there are multiple positions in the reference system with equally good mappings.

Alignment of reads to the reference H37Rv_BR again provides an indication of the relative variation in the graph, where mismatches describe small variations, and unmapped reads do this for variants larger than the read length. As expected, the indexes G^E and G^E_H both provide the same statistics in terms of mismatches and unmapped reads for the 20 input genomes. Differences between G^E and G^E_H only become apparent for mismatches and unmapped reads in the 5 hold-out genomes, where of both fewer are found in G^E. This is also expected, given that G^E indexes more paths, increasing the likelihood of finding a match for a read. This advantage in G^E is offset by an increase in the number of multi-mappings, resulting in more ambiguity in the alignment and consequent path inference. Comparing the alignments to inferred paths displays that the G^E_H path always has fewer mismatches, but that in terms of unmapped reads they are almost always equivalent. Similar trends are observed with the 5 hold-out genomes, where alignment against the indexed graphs and by extension the inferred paths are always better than using the reference as a target.

However, there are two samples in the hold-out set that deviate from the norm, these are: Kurono (in terms of unmapped reads) and KIT87190 (multi-mappings). Both cases can be explained in the context of repetitive sequence. For Kurono we see a disproportionate amount of unmapped reads in the paths, given that there are none in the alignments on the indexes. This can be attributed to the 9 to 1 weighting of edges and vertices, which can in one specific situation lead to problems. This problem occurs in a branching of vertices that have the same starting and ending subsequences (that are longer than any overlap can capture) and that diverge between these identical flanks. Given the weighting, evaluating such cases is often ambiguous and may lead to missing sequence. In KIT87190 a higher rate of multi-mapping is expected than found in the alignments on the inferred paths, meaning the incorrect multiplicity of repeats is inferred. Both of these problems may be resolved by using longer reads, since those offer more unique mappings and are more likely to span repeats.

To test whether longer reads can help improve the path inference, we build graph indexes with increasing k-sizes, along with appropriately sized read sets. Where with an increase from $k = 100$ to $k = 200$ it was now possible to properly resolve certain substructures, as shown in appendix Figure A2. However, perfect reconstruction of paths required far longer reads to span the repeats. Where only a fourth of the genomes could be perfectly inferred with $k = 500$, and this only increased to the majority when $k = 1000$.

3.3.3 Influence of coverage on genome inference

The influence of coverage on path inference was also evaluated to get an intuition of how little coverage is needed to still reconstruct a path acceptably. This was done for the reference H37Rv_BR, the result of which is shown in Figure 20.

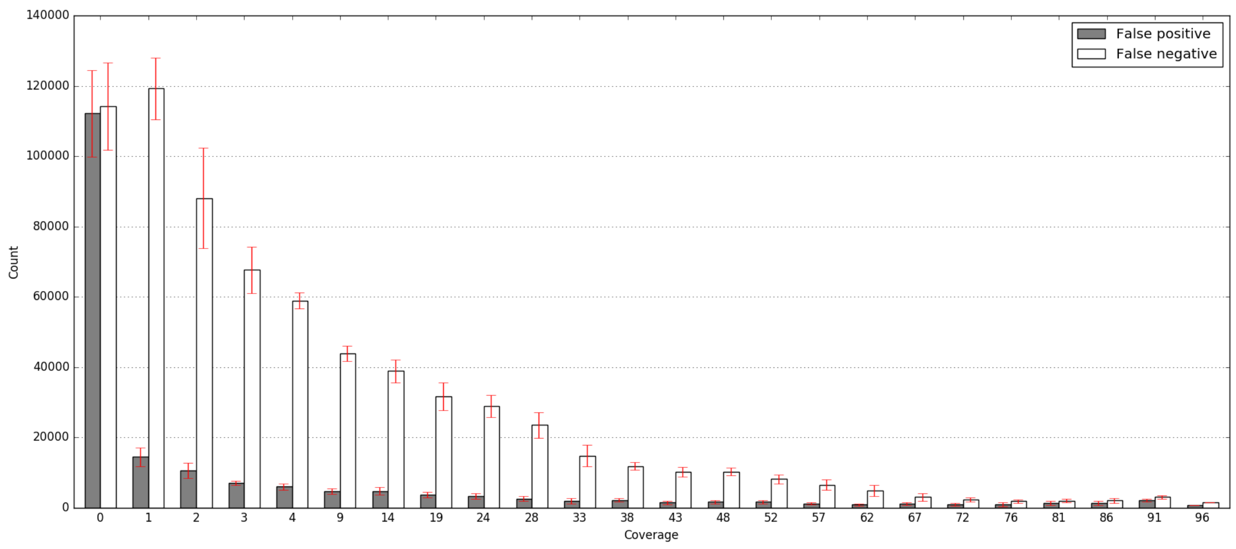


Figure 20: False positive and false negative nucleotides at varying degrees of coverage in H37Rv_BR, where for each measurement 10 samples were taken.

Note that no additional grid searches were performed for each point of coverage, hence the same parameters are used everywhere, which explains why the performance is best at maximum coverage. Overall, a steady decline in performance is observed as the coverage drops, mostly affecting the number of false negatives.

3.4 Evaluation with experimental data

3.4.1 Performance of genome inference

Experimental data was used to determine how well the methods generalize in a realistic setting, in which noise, sequencing specific biases, and contaminations starts playing a role. Of the 7 read sets, one represents the reference genome H37Rv, this is not the read set used to build H37Rv_BR, but they are used here interchangeably. A new estimate was made of the parameter set given this reference through a grid search, which yielded $exp_u = 0.0$, $exp_{(u,v)} = 0.3$, and $W_E = 1.0$, $W_V = 0.0$, with a total of ~17,000 false positives and negatives nucleotides. Although the contributions of vertex coverage was already restricted with artificial data, it is now ruled out. Notable is the difference of path quality when compared to the artificial data, where for the same strain only ~2000 nucleotides were incorrectly assigned. This may be attributed to the noise in the sequencing data, but that can only be verified by investigating the alignments themselves.

3.4.2 Alignment of reads to reference representations

The same alignment analysis as in artificial data is performed. Here the read sets of all samples are aligned to 5 targets, which include: the reference, H37Rv_BR, the indexes G^E, G^E_H, and the paths obtained from the alignments against these indexes. The results of which are shown in Figures: 21, 23, 22. Note that the number of reads available for each sample differs, for example. there are 16.2 million reads for H37Rv and 3.5 million for XDR_KZN_605. This is why the absolute values are shown for the unmapped and multi-mapping reads, the relative values are shown for the mismatches since these are quantified on a per nucleotide level.

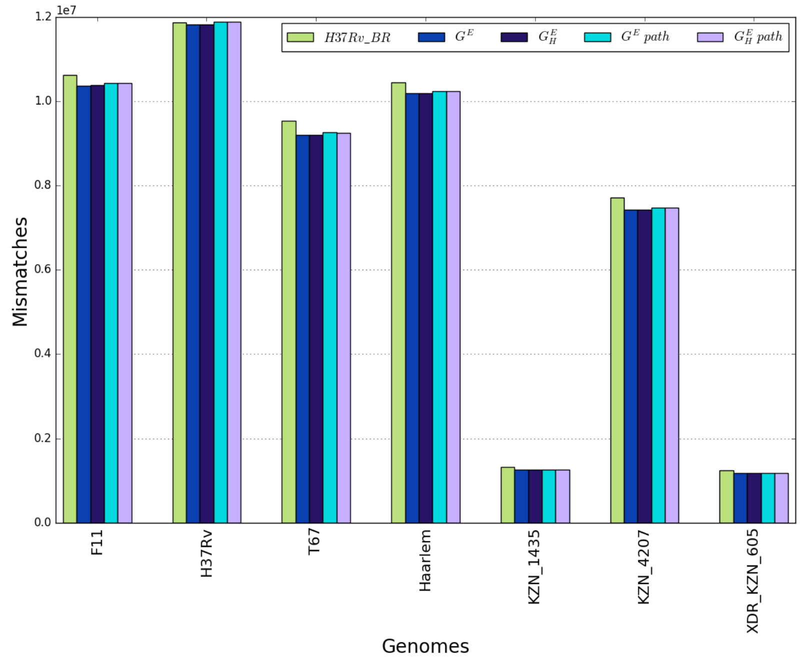


Figure 21: The total number of mismatches.

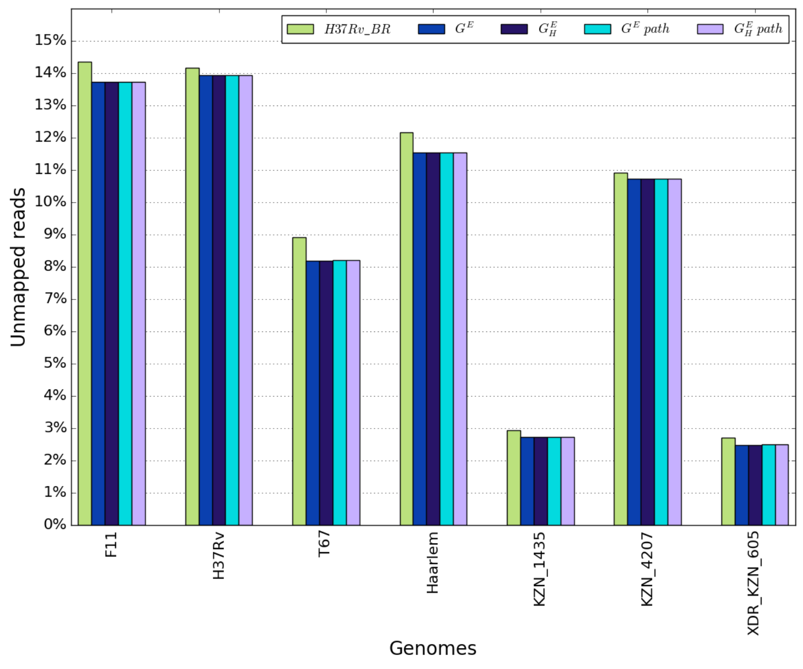


Figure 22: The number of reads for which no mapping was found.

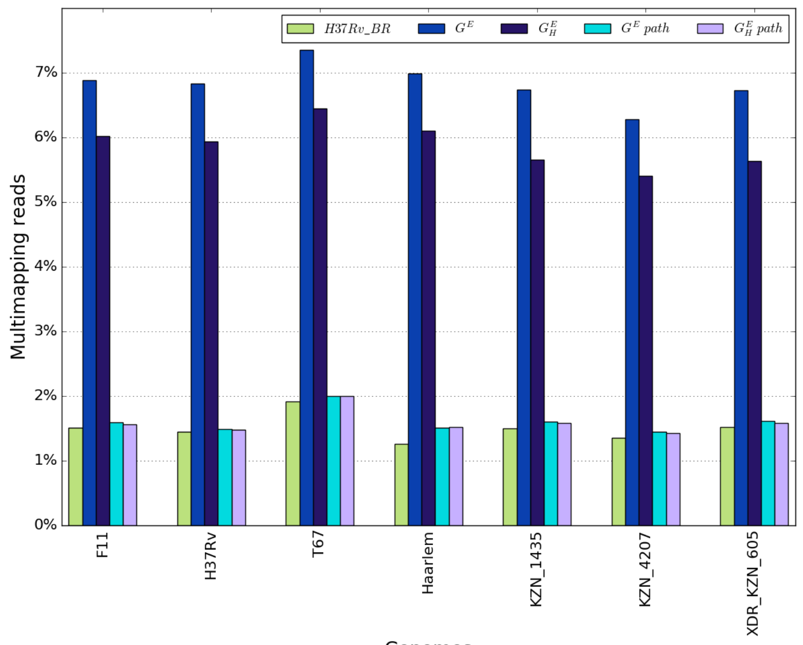


Figure 23: The number of reads for which there are multiple positions in the reference system with equally good mappings.

Interpretation is more difficult in experimental data given that we now have to deal with noise. Regardless of this, we observe the same trends as in the artificial data although these may in proportion not be as apparent. Again, the number of mismatches and unmapped reads is lowest when aligning to the indexes G^E and G_H^E, where the inferred paths do slightly worse given that they cannot be reproduced perfectly with the given read length.

What was unexpected is that there are more unmapped reads and mismatches in the alignment of H37Rv reads to the reference H37Rv_BR, than on the graph indexes. To investigate this, the reads that mapped on the indexes and are unmapped on H37Rv_BR were extracted, yielding 40,000 reads. These were consequently realigned to the graph to determine on which intervals they matched. What we found was surprising, as more than 32,000 of the reads were found to be mapped on the exact same interval, the coverage on these edges and vertices exceeding 5000 while 350 was expected. Closer inspection of these mappings showed that 62 of the nucleotides in these reads were soft clipped and only 38 of the nucleotides actually mapped to the interval. To determine the origin of these reads, a number of them were used as queries in BLASTN, and resulted in best matches in the common carp genome. It is unclear whether this is a case of contamination in the sequencing, or a chance event given that the match is only 38 nucleotides yet this does not explain the high recurrence of this particular read. The remainder of this set of unmapped reads map on entirely different strains of TB than the reference, which possibly indicates that there is also some population variation between the H37Rv_BR assembly and the

H37Rv reads.

3.4.3 Influence of coverage on genome inference

Variable coverage is again evaluated but now for the H37Rv reads, the result of which is shown in Figure 24.

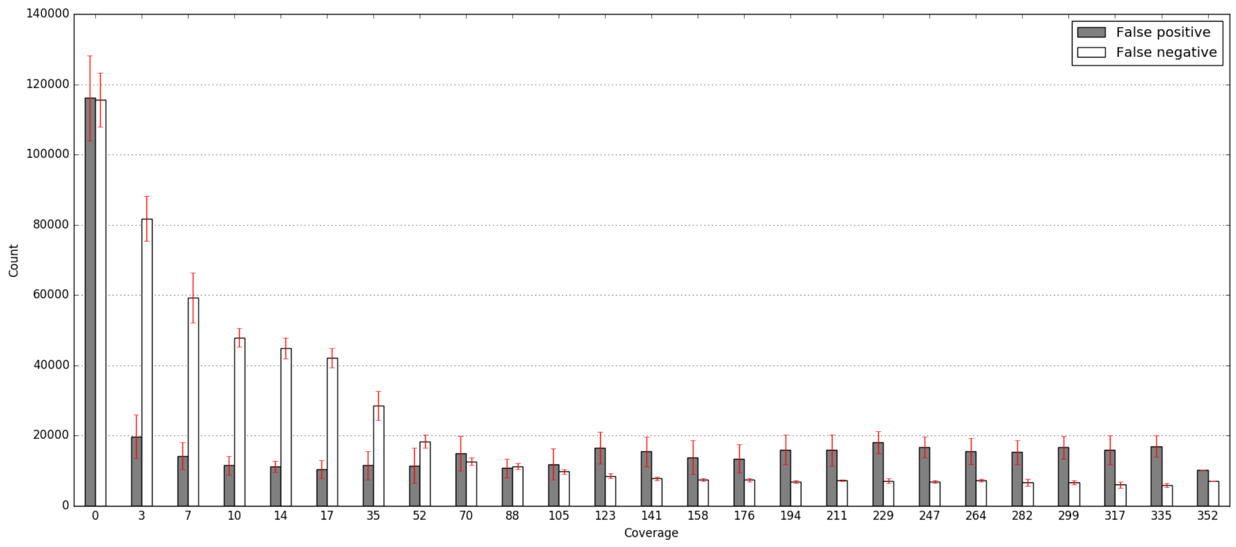


Figure 24: False positive and false negative nucleotides at varying coverage samplings in H37Rv, where for each measurement 10 samples were taken.

The trend of performance is not entirely consistent with what was observed in the artificial data, where it is not clear what happens between [128, 335] coverage. This may be attributed to insufficient samplings required to overcome the noise in experimental data. Starting and below 100 read coverage the same decrease in performance is observed as in the synthetic data.

4 Discussion and future work

The presented indexing strategy captures all *k*-length paths in directed acyclic population graphs, and can through the integration of haplotype encodings, constrain paths within haplotype blocks. The use of such constraints is crucial to make scalability of graph alignment possible, since it avoids the exponential growth of paths that occurs when trying to capture all combinations of paths and therefore allows for longer paths. It ensures that fewer paths are indexed, the resulting index becomes smaller, and only contains sequence that has been previously observed. Which is beneficial for read alignments as it reduces the size of the search space, and decreases the number of multi-mappings. This fact is also recognized by the authors of VG, which have recently expressed interest in reducing the complexity of the GCSA2 indexing, which they are planning to base on a graph generalization of genotype inference methods [28, 29].

In the experiments it was always better to align to the graph index rather than the reference, where the use of haplotype information often yielded better results, given that there are more unique mappings of reads than when all paths are indexed, which introduces more ambiguity. By inferring a path from the graph index we solve the problem of multi-mapping reads on the population graph, now providing a target on which rapidly reads may be mapping using existing alignment tools. The perfect reconstruction of paths is impossible if the reads are shorter than the repetitive content contained in the genomes encoded in the graph. The obvious solution to this problem is to use longer reads. However, a significant improvement can already be made by using paired-end reads rather than the single-end reads that are currently used for path inference. Using paired-end reads on graphs is not immediately straightforward, as it requires some sort of locality metric in the graph to estimate the distances between read pairs.

This locality metric is also necessary for the support of long reads; reads that are longer than the *k*-paths that are indexed. With such reads a seed-and-extend approach is necessary to align them. Meaning that *k*-mers are extracted from these reads and mapped onto the graph index, where the spatial positioning of subsequent *k*-mers has to be consistent to allow for accurate mapping, i.e. there must be cohesion in the intervals that are mapped. By doing this the effect of multi-mapping can also be reduced, since in the context of neighboring *k*-mers the likelihood of a particular *k*-mer to be mappable elsewhere in the genome becomes much lower.

Tuberculosis as a model organism may have been a poor choice, given the low amount of variation found between strains, which may make the advantage of using a graph representation not as apparent as it could be. This was further complicated by the experimental data for which there was only one that somewhat represented a ground truth. Neither can we show the scalability, which can further be improved by introducing parallelization of the indexing which is easily implemented, since during the process of indexing the graph is simplified into disjoint substructures, which can consequently be detected and be solved individually. Ideally the scalability is tested on the major histocompatibility complex of the human genome, a region where there is a high degree of variation found between populations. The evaluation of the path extraction is also difficult to quantify from the read alignments themselves, where ideally variation detection and recall of variants should be used as a benchmark for quality.

It may be possible to infer a diploid path or at least a simplified graph that closely describes a provided read set. Given the read distributions on the graph we can prune vertices and edges from it that are not sufficiently covered. Generating a simplified graph that only includes the vertices and edges actually supported by the reads.

References

[1] Nicholas Loman, Raju Misra, Timothy Dallman, et al. Performance comparison of benchtop high-throughput sequencing platforms. *Nature biotechnology*, 30(5):434–439, 2012.

[2] Jason Reuter, Damek Spacek, and Michael Snyder. High-throughput sequencing technologies. *Molecular cell*, 58(4):586–597, 2015.

[3] Teri Manolio, Francis Collins, Nancy Cox, et al. Finding the missing heritability of complex diseases. *Nature*, 461(7265):747–753, 2009.

[4] 1000 Genomes Project Consortium et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.

[5] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012.

[6] Eric Lander, Lauren Linton, Bruce Birren, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[7] International Human Genome Sequencing Consortium et al. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, 2004.

[8] Deanna Church, Valerie Schneider, Tina Graves, et al. Modernizing reference genome assemblies. *PLoS Biol*, 9(7):e1001091, 2011.

[9] Yu Liu, Mehmet Koyutürk, Sean Maxwell, et al. Discovery of common sequences absent in the human reference genome using pooled samples from next generation sequencing. *BMC genomics*, 15(1):685, 2014.

[10] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, Mar 2010.

[11] Stephen Sherry, Minghong Ward, M. Kholodov, et al. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.

[12] Michael Cariaso and Greg Lennon. SNPedia: a wiki supporting personal genome annotation, interpretation and analysis. *Nucleic acids research*, 40(D1):D1308–D1312, 2012.

[13] Robert Handsaker, Joshua Korn, James Nemesh, and Steven McCarroll. Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nature genetics*, 43(3):269–276, 2011.

[14] Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, 2013.

[15] Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, et al. Simultaneous alignment of short reads against multiple genomes. *Genome Biol*, 10(9):R98, 2009.

[16] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. 1994.

[17] Erik Garrison, Adam Novak, Glenn Hickey, et al. Variation graph tools. `https://github.com/vgteam/vg`, 2016.

[18] Simon Gog, Matthias Petri, et al. Succinct Data Structure Library (SDSL). `https://github.com/simongog/sdsl-lite`, 2016.

[19] Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *String Processing and Information Retrieval*, pages 164–175. Springer, 2009.

[20] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing finite language representation of population genotypes. *arXiv preprint arXiv:1010.2656*, 2010.

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

[22] Jouni Sirén. Indexing variation graphs. *CoRR*, abs/1604.06605, 2016.

[23] Christopher Lee, Catherine Grasso, and Mark Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[24] Mengyao Zhao, Wan-Ping Lee, Erik Garrison, and Gabor Marth. SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. *PloS one*, 8(12):e82138, 2013.

[25] Jasper Linthorst, Marc Hulsman, Henne Holstege, and Marcel Reinders. Scalable multi whole-genome alignment using recursive exact matching. *bioRxiv*, 2015.

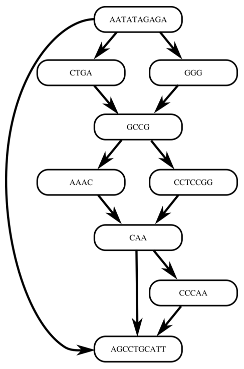
[26] Jasper Linthorst. REVEAL (REcursiVe Exact-matching ALigner). `https://github.com/jasperlinthorst/reveal`, 2016.

[27] Keira Cohen, Thomas Abeel, Abigail McGuire, et al. Evolution of extensively drug-resistant tuberculosis over four decades: whole genome sequencing and dating analysis of Mycobacterium tuberculosis isolates from KwaZulu-Natal. *PLoS Med*, 12(9):e1001880, 2015.

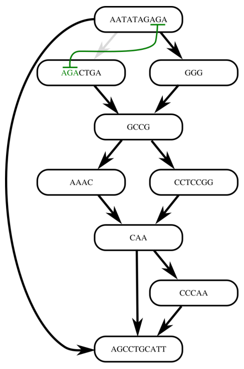
[28] Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

[29] Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional burrows-wheeler transform and its applications. *bioRxiv*, page 051409, 2016.

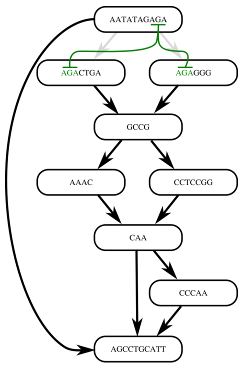
5 Appendix



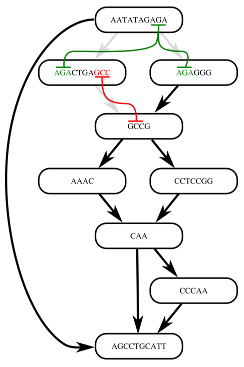
(a) Input graph



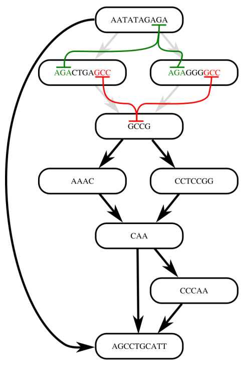
(b)



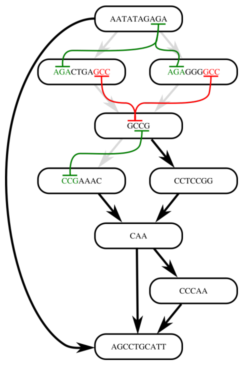
(c)



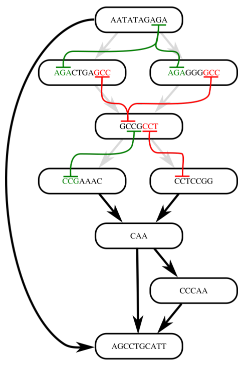
(d)



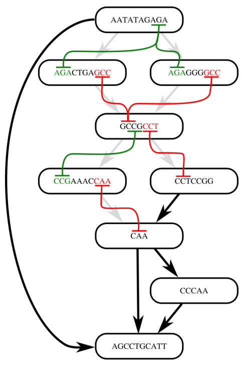
(e)



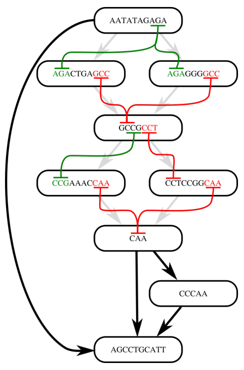
(f)



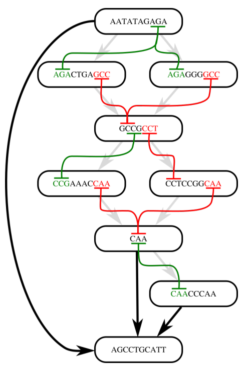
(g)



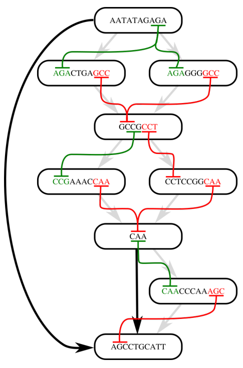
(h)



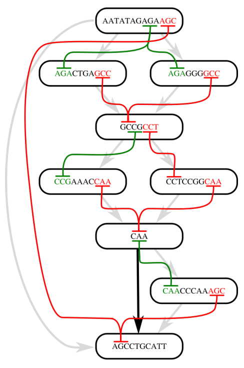
(i)



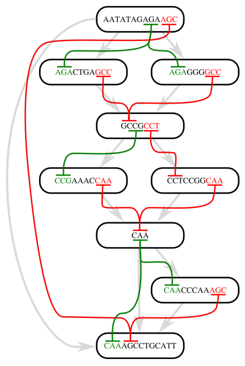
(j)



(k)

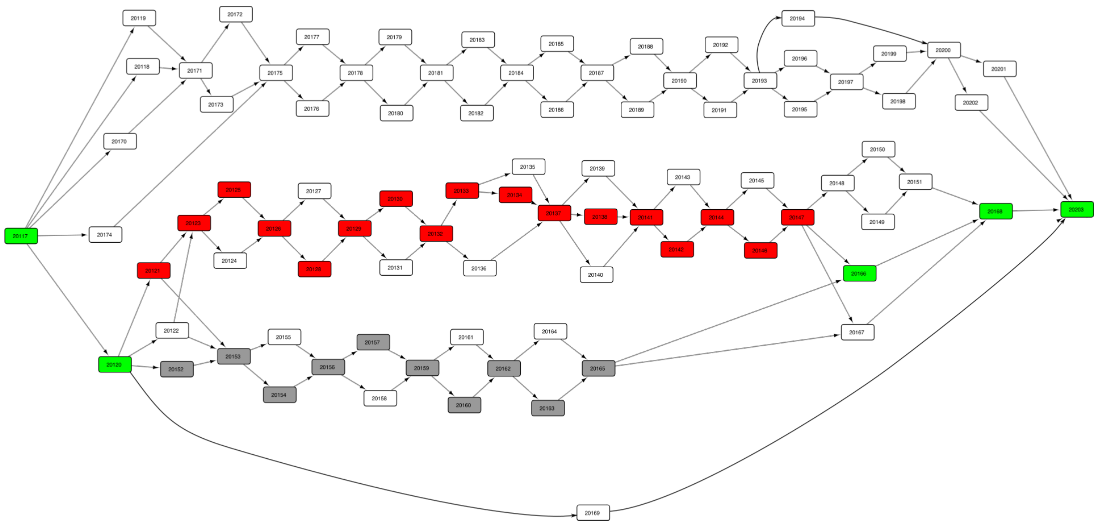


(l)

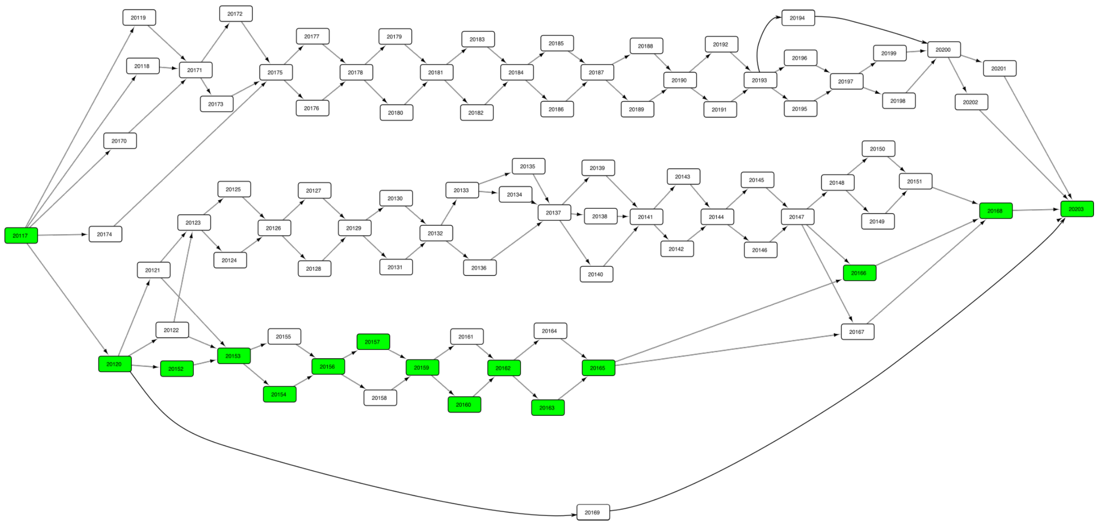


(m)

Figure A1: Step by step indexing of the graph in figure 3, which is completed in 12 subsequence extensions from **b)** to **m)** (no duplications or collapses are required), line coloring indicates prefix (green) or suffix (red) extension.



(a) $k = 100$



(b) $k = 200$

Figure A2: Subgraph in the 20 genomes graph where for $k = 100$ the incorrect branch is inferred for a given genome, denoted as the red vertices in **a)**. By increasing the path length to 200, it was now possible to properly resolve this subgraph as shown in **b)**.