Chapter 1

List of papers

Chapter 2

Intrvoduction

2.1 Introduction

2.2 Introduction

The broad topic of this thesis graph based reference genomes, and specifically of mapping and peak calling on such references. The introduction here is meant to provide an introduction to the topic as well as a coherent brief of the content of the provided papers. As is common in bioinformatics, full understanding of the field of graph based references requires at least a rudimentary understanding of the underlying biology, mathematics and informatics. An brief introduction to the specific subtopics

The first section will be an introduction to two main biological topics, reference genomes and genomic variation, in order to familiarize the reader with topics in addition to establish a perspective that establishes the benefits of graph based references. Section 2 introduces the concept of sequence graphs and their interpretation in different settings. Section 3 describes the mathematical formalism used in this thesis as well as the data structures embodying this formalism. These sections might be persived as too rigorous and opaque for a brief introduction, but this highlights one of the main drawbacks of graphical models, namely their complexity. Section 4 and 5 gets to the main focus of this thesis, covering the topics of read mapping and peak calling on graph based reference genomes.

2.3 Biology Background

2.3.1 DNA

At the heart of molecular biology is *deoxyribonucleic acid*, *DNA*. DNA-molecules consists of pairs of four different *nucleotides* linked together to form a double stranded helix. The four different nucleotides are often represented by the letters

A (*adenin*), C (*cytosin*), T (*thymin*) and G (*guanin*), forming an alphabet of nucleotides $\Gamma_{DNA} = \{A, C, T, G\}$. Each nucleotide can only be paired with one of the others, such that each base-pair in the helix is either (A-T) or (G-C) or opposite. (*chromosomes*)

The most important function of DNA is to serve as a template for the synthesis of proteins, which in turn are responsible for a wide range of bio-molecular functions. This is a two-step process where DNA is first *transcribed* to *RNA*, molecules similar to DNA but single stranded and with *thymin* replaced by *uracil* (U). The resulting RNA-sequence can in turn by translated to proteins by mapping triplets of ribonucleotides to amino acids, which are the building blocks of proteins.

Sequneces of DNA that are transcribed to RNA that either has a function in itself, or is in turn translated to protein, are called *genes*. Genes constitute only a minor proporotion of human DNA. The remaining DNA can be important due to the biochemical properties of the DNA itself.

2.3.2 Replication and Mutation

In addition to serving as template for RNA molecules, DNA can also be replicated. This leads to inheritance both on cellular and organismal level. In normal cell division, the chromosomes are replicated such that each of the two resulting cells have a copy of the DNA from the original cell. Furthermore, germ line cells are copied and transferes half of the organisms chromosomes to an offspring.

However, this DNA-replication is not always accurate. Errors can occur leading to a copy of the DNA molecule which is not identical with the original. Most common are the substitution of a single nucleotide, insertion of a small dna sequence, or deletion of a small subsequence. Such errors lead to difference between cells within an organism, or difference between individuals.

(*more about variation*)

Difference in the DNA-sequence between two individuals can lead to a change in the transcribed RNA sequence and further in the sequence, and thereby the form and function, of the expressed protein. Or it can lead to less drastic changes such as changes in the RNA-structure or the shape of the DNA molecule itself. In this way genomic variation can determine differences between specimens of the same species and also differences between the species themselves.

2.3.3 Epigenomics

Difference in DNA-sequence can explain phenotypic (*explain geno/pheno*) difference between individuals, but fails to explain the difference between the different cells within an organism. Within an organism, the cells contain mostly the same DNA-sequences, but exhibit vastly different phenotypes.

The reason for this difference is fundamentally mostly attributed to differences in expression levels of genes. Much attention has been devoted in recent years to the mechanisms determining gene expression levels. Two high-level consortia, ENCODE **??** and Roadmap Epigenomics **??**, have systematically

conducted experiments geared to understanding some of these factors, including: 3D-configuration of the chromosomes, accessiblity of the DNA in different regions, transcription factor binding sites, and methylation patterns across the genome. This thesis is consentrated transcription factor binding sites, but for an introduction to epigenomics as a whole see **??**.

Transcription factors are proteins that bind to DNA, with an affinity for certain DNA-sequences. These proteins functions as signals to other proteins that a gene in it's vicinity should be transcribed, thus affecting expression levels of genes. As such the precence or absence of transicripion factors binding to a binding site can affect the phenotype of a cell. Also since the binding of transcription factors are dependent on the DNA-sequence, a mutation in the DNA-sequence in a binding site can affect the binding of the protein and thus the expression of a nearby gene.

2.3.4 Sequencing

Since differences in DNA-sequence can explain differences in biological function, it has long been a goal to read the DNA-seqeunce of a sample as accurately and effieciently as possible. Currently, no technology exists to accurately sequence whole molecules like a human chromosome with accuracy and thus sequencing has been dominated by reading small (50-600bp) sequence fragments. The ability to massively sequence short reads have been very influential in medicine and biology, with a wide range of applications. The ability to deduce DNA-sequences have

- DNA
- mutations,
- sequencing,
- assembly,
- alignment,
- read mapping,
- chip-seq,
- etc.

This should explain basics:

2.4 Alignment

This section is about sequence alignment. Although not in itself central for this thesis, it is introduced here for two reasons. Firstly it is the area where sequence graphs where first introduced, and also provides the most clear interpretation of a sequence graph. Secondly, it provides a good starting point for introducing read mapping which is discussed in the next section.

2.4.1 Pairwise Sequence Alignment

Pairwise sequence alignment is the problem of aligning letters in two sequences in a way that minimizes some distance measure between them. Typically one assumes that the two sequences are separated by a set of mutations (SNPs, deletions and insertions), where a cost is given to each mutation. For instance a singe nucelotide substititution. For instance the alignment of the DNA-sequences $ATAGTGCCGTG$ and $ACAGTGGTG$, would be ($ATAGTGCCGTG$, $ACAGTG–GTG$), which would have the simple edit distance 3 (1 substitution and 2 deletions). In general the optimal alignment given an affine gap penalty can be found using Needleman-Wunch [**?**] in $O(n * m)$ time by using the recurrence relation

$$D(s_{:k}, q_{:l}) = min(D(s_{:k-1}, q_{:l-1}) + D(s_{k-1}, q_{l-1}), D(s_{:k-1}q_{:l} + G), D(s_{:k}q_{:l-1} + G))D(0, k) = D(k, 0) = Gk$$

(*Explain Backtracking*)

2.4.2 Multiple Sequence Alignment

2.4.3 Sequence Graph Alignment

Hein et al [**?**] showed that the needleman wunch could be generalized to align a sequence to a pair of already aligned sequences. This was done by representing an alignment as a simple *sequence graph*. A sequence graph in this setting is a graph where each node represents a letter, and an edge between two nodes means the two corresponding letters are consecutive in one of the sequences. Formally we let a *simple sequence graph be a graph* $G = (V, E)$ with a labeling $s : V \to \Gamma_{DNA}$. An pairwise alignment can be represented as a simple sequence graph by (*Alignment to sequence graph*). Given a topological ordering $O(v)$ of the vertices in the sequence graph, the needleman wunch algorithm can be extended to sequence graphs by:

$$D(v_1, v_2) = min(\{min(D(p_1, p_2) + D(p_1, p_2), D(p_1, v_2) + G, D(v_1, p_2) + G) \mid p_1 \in P(v_1) \wedge p_2 \in P(v_2)\})$$
$$D(v_1, v_2) =$$

The interpretation of the sequence graph alignment is that a path in the sequcence graph created from the alignment of two sequences represents a possible ancecstor sequence of the two sequences and that the alignment of a third sequence to this graph represents the best alignment of this sequence to one of the possible ancestor sequences. Each sequence s_a created from a path in the graph has the property that $D(s_a, s_1) + D(s_a, s_2) = D(s_1, s_2)$. By doing a similar adaption to align seqeucne graphs to sequence graphs, and representing the alignments as a sequence graph of the the alignment of the two ancecstor sequences, the sequence grpah alignment represents a closed operation on simple sequence graphs. Thus a multiple sequence alignment can be produced by iteratively aligning sequence graphs [**?**].

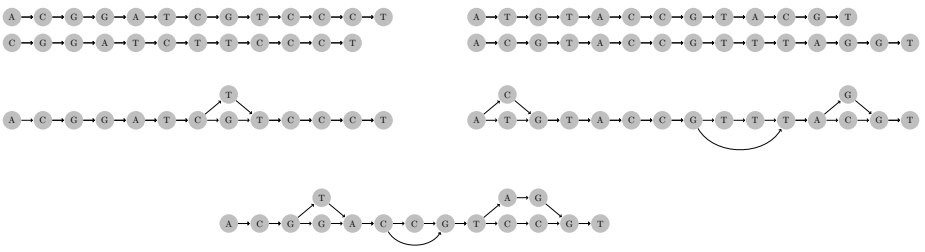

Figure 2.1: Iterative sequence graph alignments of four sequences evolved in two generations from $ACGTACGTACGT$. The sequences are represented as linear sequence graphs (a) and pairwise alignment is performed on the closest pairs yielding two sequence graphs (b). These two seqence graphs are then aligned to eachother, yielding a sequence graph representing an alignment of the two closest paths in the graphs (c). As seen the original sequence is in the alphabet of the final sequence graph

2.4.4 Large Scale Sequence Alignment

In theory, pairwise sequence alignment could be used to find out where a small sequence originates from a large sequence. However, since the Needleman-Wunch alogorithm has complexity $O(n * m)$, it is too computationally demanding to use this directly when dealing with large numbers of sequences and large seqeunces.

2.5 Mapping

Making sense of massive amounts of short reads from NGS experiments is of fundamental importance to modern bioinformatics. This can be done by analyzing the sequence overlaps of the reads to deduce longer contiguous sequences such as in genome assembly [**?**] and other reference-free methods [**?**]. This is however usually very computationally expensive, and demands large amounts of data in order to achieve certain results. When available, mapping the reads to a *reference genome* can both reduce the complexity and improve the results of the analysis. A reference genome gives a prior estimate of what the genome sequence of an individual of that population is. This makes it possible to look at each read from an experiment independently and get an estimate of where that read comes from in the genome. In this thesis' projects reference genomes for *Arabidoposis thaliana*[**?**], *Drosophila melanogaster* [**?**] and human [**?**] populations have been used. (*Maybe mention graph based here*) The following is a brief introduction to the field of mapping reads to a reference genomes.

2.5.1 Linear Mapping

Mapping a read $r \in \Gamma^*$ to a linear reference genome $G \in \Gamma^*$ can be seen as solving the problem of finding an interval $M(r|G) = (\hat{s}, \hat{e})$ that minimizes some edit distance $D(r, G_{\hat{s}:\hat{e}}$ as well as returning an uncertainty estimate of the mapping $\epsilon(r, G, (s, e))$. Given no restraints on computation time or memory usage, a local alignment of r versus G finds the exact minimum (*which distance*). However since this algorithm has complexity $O(|r|\,|G|)$, this is not possible in practice when dealing with million of reads and reference genomes with a length in the billions. Thus indexing and further approximations are needed to meet the demands of modern bioinformatics. A common approach is to generate an index that allows for efficient searching of subsequences, and to combine the matches from subsequences into a match for the whole sequence r. The projects in this thesis have used $BWA - mem$ as the program to map reads to a linear reference genome. In the following a brief description of the methods used in $BWA - mem$ [**?**] is explained.

FM-index

The BWA-mem uses the strategy of first finding exact matches to subsequences followed by chaining these subsequences together. Finding the exact matches is done using an FM-index.

An FM index [**?**] is a data structure that allows lookup of a sequence s in a reference sequence R in $O(|s|)$ time. It is based on a lexiographic sort of the suffixes of R (*BWT?*) and the index structure contains representations of two character vectors. One for the first letter of each suffix in sorted order and one for the letter preceding each suffix. The power of the FM-index comes from properties of these columns.

- The letters in L are sorted, so that it can be represented by the offset of the firt occurance of each letter in the alphabet
- Each letter r_i in R represents a new suffix, $r_i S_i$.
- If r_i is the nth occurence of that letter in R, the suffix $r_i S_i$ is represented by the nth occurance of r_i in L.

These properties makes it possible to start from the last letter of a query string and traverse the FM-index in order to find all occurances of that string in the reference, or finding the longest suffix of the string that is precent in the reference string.

BWA-mem

The FM-index can be used to find *Super-Maximal Exact Matches* (SMEMs). These are exact matches of subsequences of the query to the reference, with the property that they cannot be extended further and are not contained in another. The set of SMEMs on (*mem-chaining*)

2.5.2 Graph Mapping

Mapping to a graph based reference is similar to the linear case, except that instead of estimating a liner interval $(\hat{s}, \hat{e})$, a graph interval $(\hat{s}, \hat{e}, \hat{v})$ is estimated. It is however deceptively more complicated. Firstly because the number of subsequences in a sequence graph grows exponentially with the complexity of the graph. And secondly since chaining subsequence-matches is more complicated due to the possible existance of multiple paths between two matches. *vg* [**?**] has been at the forefront of mapping to a graph reference, showing that it can lead to better mapping accuracy than BWA-mem. But it is still too slow and memory- and disk-consuming for widespread use. Below is a brief description of the alogrithms used by *vg*, but first a brief discussion of variation on genome scale.

Genome Scale Variation

When representing variation on genomic scale, it is common to represent the variation with respect to a singe reference sequence. Thus instead of representing each contributing sample as a row in a block, as in multiple sequence alignment, one represents each variant with a position in the reference sequence, a subsequence of the reference from that position, and the alternative sequence that replaces that sequence. The full sequence of a sample can then be represented as a sample of variant ids. The most common format for representing such variants and sample sequences is the Variant Call Format (VCF)[**?**].

On a genome scale, other types of variants than SNPs and indels are of interest.

As with a MSA block format, a reference sequence with a set of variants can be represented uniquely by a sequence graph. This is done by first representing the reference sequence as a linear sequence graph and then adding to the graph the nodes and edges to represent the variants. The sample sequences can also here be represented as paths through the graph.

The most common variations (SNPs and indels), leads to directed acyclic graphs (DAG) when converted to a sequence graphs. However other types of variants, relevant on a genomic scale, does not have this property and thus leads to more complicated graphs.

Large scale variants that affects the ordering of the nucleotides are not well suited to being represented by DAGs. An example of this is *transposable elemets* (TEs), where a subsequence of DNA is moved or copied (*(only copied?)*) to another location in the genome. This can be represented as a DAG by adding a new variant in the graph covering the whole sequence from covering both the old an new postions of the subsequence. However this can lead to much redundancy since the sequence between the two positions will be represented twice. The other alternative is to only add new edges to the graph, representing the sequence when the substring has been moved. This will however break the acyclisity of the graph, and thus complicate most operations one would do on the graph.

Another case which will lead to redundancy if represented as a graph is reversals. Here a piece of the DNA is reversed leading to the subsequence being substituded with its reverse complement. Adding a subsequence in the graph representing the reverse compliment is inderictly redundant. Even though the reverse compliment is not included as a path in the graph, it is directly deducible from a sequence in the graph. In order to represent suche reversals, and other things needing the reverse compement, the concept of a side graph has been introduced. In a side graph, all the nodes representing nucleotides have to sides and an edge is defined as going from one side of a node to a side of another node. One node-side represents the nucleotide, while the other side represents the complement in the other reading direction.

vg

At the heart of *vg* is the concept of Variation Graphs. A Variation Graph is a side graph togheter with a set of paths that represents the sample sequences. Since it uses a side graph and not a simple sequence graph it can represent SNPs and indels in addition to the large scale variants discussed above. This however comes at the price of complexity. A key functionality of *vg* is to map reads to the the side graph. This is done similarily to how BWA-mem maps read to a linear reference sequence, by finding matches for subseqeunces and chaining these togheter. The process is more involved due the the complexity of graphs. (*vg chaining*)

The GCSA-index [**?**, **?**] is a generalization of the FM-index, where arbitrary-length sequences can be looked up in a sequence graph. Originally constructed to work on acycilic sequence graphs, gcsa2 extends the functionallity to general variation graphs. For simplicity we will here constrain the discussion to acyclic graphs.

The link between the GCSA-index and the FM-index can be most clearly seen by considering a linear sequence as a linear sequence graph.

Each pair of values (l, r) can then be seen as an edge in the linear sequence graph. The backwards traversal done when looking up a sequence can then be seen as traversing the linear sequence graph backwards.

In the GCSA-index each edge in the sequence graph is represented as a pair in the L and R columns, allowing a backwards traversal. However, the ordering properties necessary for the backwards traversal to work can only be obtained if for each node the sequence from the node to the next bifurcation in the graph is unique. If this property is not met, nodes that does not fulfil it have to be duplicated. In highly complex regions this is a costly procedure, so it is necessary to prune variants in complex regions before creating the gcsa index.

After finding the SMEMs (*explain SMEMs*) for a read, these need to be chained in order to find a complete match. In *vg* this is done by first clustering the SMEMs by their location in the reference graph, and then using a markov chain method to find the one with optimal colinearity (*not clear to me*).

2.6 ChIP-seq

Chip-seq is an experiment designed to elucidate where a transcription factor binds. Immunoprecipitation is used to isolate small fragments of DNA which have the transcription factor bound to them. Ends of these framgents are then sequencesed yielding reads that in general are assumed to be in the vincinity of a binding site. In order to find from this where the transcription factor was bound bioinformatics pipelines are needed. This is in general consists of mapping the reads to a reference genome, and performing *peak calling* on the coordinates of the mapped reads. Peak calling is required to estimate from the mapped reads, only known to tend to be in the vicinity of a binding site, where the binding site is. The input to peak calling is a set of directed intervals and the output is generally either a set of positions, or a set of undirected intervals that denote the predicted binding sites.

$$PC : \mathbb{I}_G \to \mathbb{I}_G^+$$

A range of methods for performing peak calling exists [**?**, **?**, **?**]. In the following is described the algorithms used by MACS2 and the adaptions to those algorithms used by Graph Peak Caller descirbed in Paper 3 and Paper 4.

2.6.1 MACS2

The input is a set of intervals $\{(s_k, e_k, d_k)\}$. The first step is to count how many extended reads cover each position of the reference genome, where each interval is extended to a length equaling a previously estimated fragment length f. I.e for each position i we want to find the count

$$P(i) = |\{k \mid d_k = 1 \wedge s_k \leq i < s_k + f\} \cup \{k \mid d_k = -1 \wedge e_k - f \leq i < e_k\}|$$

The pileup function P is represented sparsely by a set of indices $\{s_k\}$ and values $\{v_k\}$ such that

$$\forall k \forall j \in \{s_k, s_k + 1\}, , , s_{k+1}(P(j) = v_k)$$

the indices and values are found algorithmically as

```
def count_extended(intervals, f):
    starts = [s if d==1 else e-f for s, e, d in intervals]
    ends = [e if d==1 else s+f for s, e, d in intervals]
    changes = starts+ends
    args = argsort(changes)
    codes = [1 if arg<=len(starts) else -1 for arg in args]
    s = changes[args]
    v = cumsum(codes)
    return s, v
```

Which is done in $O(n \log n)$ time. The counts for each position is compared with a background track which represents a local average of reads. This is generated by using a large extension length and dividing the pileup values

by the extension size. s, v = count_extended(control_intervals, E)/E; v/=E. Assuming that each count $P(i)$ are Poisson distributed as P_i $Poisson(\lambda_i)$, a null hypothesis of $H_0^i : \lambda_i = C(i), H_a^i : \lambda_i > C(i)$ is made for each position and a p-value calcluated: $p_i = P(P_i >= P(i))$ *(too many ps)*. Too adjust for multiple testing, a final set of q values is calculated as $q_i = p_i N_i$ where $N_i = |\{k \in \{1, 2, , , |G|\} \mid p_k \leq p_i\}|$. The q values are thresholded on a given significance level α so we get $T(i) = q_i \leq \alpha$.

```
def get_p_values(pileup, background):
    indices = pileup.indices + background.indices
    indices.sort()
    pileup_values = pileup.values[search_sorted(indices, pileup.indices)]
    background_values = background.values[search_sorted(indices,
        background.indices)]
    p_values = poisson.sf(pileup_values, background_values)
    return Pileup(indices, p_values)
```

Which is done in $O(n)$ time.

```
def get_q_values(p_values):
    counts = diff(p_values.indices)
    args = argsort(p_values.values)
    values, idxs = unique(p_values, return_index=True)
    N = cumsum(counts[args])[idxs]
    q_values = N*values
    all_q_values = zeros_like(p_values.values)
    all_q_values[idxs] = diff(q_values)
    all_q_values = cumsum(all_q_values)
    restored = zeros_like(all_q_values)
    restored[args] = all_q_values
    return Pileup(p_values.indices, restored)
```

The thresholded values are obtained simply by Pileup(q_values.indices, q_values.values>alpha).

The final peaks are called in two steps from the thresholded values. First, joining peak intervals that are separated by less than the estimated read length, and secondly removing peaks that are shorter than the estimated fragment length f. Both steps can be made using the same function.

```
def remove_small(indices, size):
    indices = indices.reshape((-1, 2))
    lengths = indices[:, 1]-indices[:, 0]
    big_enough = lengths>=size
    return indices[big_enough].ravel()

peaks = r_[indices[0], remove_small(indices[1:-1], read_length),
    indices[-1]]
```

peaks = remove_small(peaks, fragment_length)

2.6.2 Graph Peak Caller

The following is a brief description of the algorithms and data structures used in the adatptation of MACS2 to graph genomes. It is assumed here that the sequence graph is a compressed simple sequence graph, and therefore also acyclic. We also assume that the node ids are sorted according to a topological sort of the graph.

The fundamental data structure in for graph peak caller is the linear representation that converts coordinates from $LR : \mathbb{N}_{|V|} \times \mathbb{Z} \to \mathbb{N}_N$, where $N = \sum_{i=0}^{|V|-1} L(i)$ is the number of positions in the graph. This linear mapping is represented in the LinearRepr class, which has an array node_offsets where node_offsets[i]=$\sum_{k=0}^{i} L(k)$ such that a coordinate $(v, o) \in \Gamma$ is converted to $i \in \epsilon$ by node_offsets[v]+o. The inverse conversion of a coordinate $i \in \epsilon$ is done by v=searchsorted(node_offset, i); o = i−node_offsets[v])

A pileup is created in the linear coordinate space by the following method. For each interval (s, e, v, d) we define the extended interval as a tuple

$$ext(I_k, f) = \begin{cases} (\{s_k\}, E_k^+, X_k^+), & \text{if } d = 1 \\ X_k^-, E_k^-, \{e_k\}), & \text{if } d = -1 \end{cases}$$

where

$$\begin{aligned} E_k^+ &= \{(v_1, v_2) \in E \mid D(e_k, (v_2, 0)) \leq f\} \cup \{(v_1, v_2), (v_2, v_3), , , (v_{k-1}, v_k)\} \\ X_k^+ &= \{((v, o) \in \Gamma \mid D(e_k, (v, o)) = f\} \\ E_k^+ &= \{(v_1, v_2) \in E \mid D((v_2, 0), s_k, \leq) f\} \cup \{(v_1, v_2), (v_2, v_3), , , (v_{k-1}, v_k)\} \\ X_k^+ &= \{((v, o) \in \Gamma \mid D((v, o), s_k) = f\} \end{aligned}$$

From this we can get the sparse pileup as

$$\delta(LP((v, o))) = \begin{cases} N_s((v, o)) - N_e((v, o)) + \delta_E((v, o)), & \text{if } o = 0 \\ N_s((v, o)) - N_e((v, o)), \text{else} \end{cases}$$

$$\begin{aligned} N_s((v, o)) &= |\{(X_s, E, X_e) \in Ext \mid (v, o) \in X_s\}| \\ N_e((v, o)) &= |\{(X_s, E, X_e) \in Ext \mid (v, o) \in X_s\}| \\ \delta_E((v, o)) &= |\{(X_s, E, X_e) \in Ext \mid \exists v_0 \in V\, [(v_0, v) \in E]\}| \\ &\quad - |\{(X_s, E, X_e) \in Ext \mid \exists v_2 \in V\, [(v - 1, v_2) \in E]\}| \end{aligned}$$

Algorithmically this pileup is more complex to create than the linear pileup. The $O(|V| + fk\,|I|)$ alogrithm used by graph peak caller is described below:

```
def extend_interval(interval, f, graph):
    starts = set([ interval.start ])
    edges = {(vertex,  interval.vertices [i+1])
             for i, vertex in enumerate( interval.vertices [:-1])}
    \TODO{Rest of code if simple enough}
```

Given the sparse pileup in linear coordinates, the steps for calculating $p-$ and $q-$ values and thresholding for significance is the same as for the linear case.

Removing small holes in the intervals is again more complex than in the linear case. First the intervals in linear coordinates corresponding to significant q values are converted to intervals in graph coordinates, s.t

Secondly, these are graph intervals are clusetered into connected subgraphs so that

Lastly, we join any two subgraphs that are connected by a path with distance shorter than the read length, i.e:

The penultimate step is to convert the resulting subgraphs into graph intervals, by choosing the edges with the highest coverage in the edge pileup. Finally these paths are filtered by their length so that all remaining intervals are longer than the estimated fragment length.

2.7 Discussion

Using sequence graphs as reference structures

2.7.1 Complexity

Indexing a sequence graph has been shown to be inherently difficult [**?**]. It is therefore no surprise that *vg*, which is the graph mapper that shows the most promising results suffers from long run times and high memory consumption.

2.7.2 Kmer Sinks

The combinatorial growth in subsequences from variant density not only leads to problems with computational efficiency. It also leads to possibilities for false positives. As is briefly discussed in Paper 3, areas with a high density of variants, can represent a large number of possible sequences. This can lead to such areas matching read sequences which does not originate from that area. Such mismappings is a problem for the mapping accuracy in general and the bias it introduces can be a problem for downstream analysis.

2.7.3 Pseudo-linearity

2.8 Recent Developments

Haplotype aware mapping

2.9 Summary of Papers