Graph Based Reference Genomes

Knut Dagestad Rand

April 15, 2019

Thesis submitted for the degree of Philosophiæ Doctor

Contents

Chapter 1

List of papers

Chapter 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.

Chapter 3

Background

3.1 Molecular Biology

3.1.1 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.

3.1.2 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.

3.1.3 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.

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

3.2 Sequencing

3.2.1 Whole Genome 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

3.2.2 Chip-Seq

3.3 Reference Genomes

ENCODE data, og gen lister.

3.4 Mathematical Framework

3.4.1 Strings

3.4.2 Notation

Strings are important in this thesis as they are the means to represent DNA-sequences. We will here formally work with an alphabet $\Gamma_{DNA} = \{A, C, G, T\}$ and say that a *sequence* of length n over that alphabet are a tuple in the set Γ_{DNA}^n. The set of all sequences over $\Gamma_{DNA} = \{A, C, G, T\}$ is represented by Γ_{DNA}^*. The length of a sequence $s \in \Gamma_{DNA}^*$ is notated $|s|$. The ith symbol in a sequence s is notated $s[i]$ and a subsequence of s starting at the ith symbol (inclusive) and ending at the jth symbol (exclusive) is denoted $s[i : j]$. A *prefix* of s, i.e. a substring of s starting from the first symbol, of length k is denoted $s[: k]$, while a suffix of s starting at the kth symbol is denoted $s[k :]$. For convenience a string of length n is sometimes represented as $s[: n]$ to convey the size of the string. Concatenation of two strings S, T are represented as $S * T$, while the concatenation of a symbol a to a string S is denoted Sa, and a string to a symbol as aS.

3.4.3 Graphs

A graph is a tuple $G = (V, E)$ of vertices and edges, where the edges are pairs of vertices $E \subset V^2$. We will here deal with directed graphs where we say that edge (v_1, v_2) is an edge from v_1 to v_2. An *path* is an alternating sequence of vertices of edges $(v_1, e_1, v_2, e_2, , , e_{n-1}, v_n)$ where $e_i = (v_i, v_{i+1})$. A cycle is a path $(v_1, e_1, v_2, e_2, , , e_{n-1}, v_n)$ where $v_1 = v_n$ and $n > 1$, i.e. a path that starts and ends at the same vertex. A directed acyclic graph (DAG) is a directed graph in which there are no cycles. We call the set of all paths starting at v_s and ending at v_e pathsv_s, v_e.

Sequence Graphs

We define a *simple sequence graph* $SG = \{G, s\}$ over an alphabet Γ as a graph $G = \{V, E\}$ and a label function label $: V \to \Gamma$ labeling each vertex with a letter from Γ. We refer to the label of a path label$((v_1, e_1, v_2, , , e_{n-1}v_n))$ as the concatenation of the labels of its vertices label$(v_1) *$ label(v_2)...label(v_n). The *language* recognized by a sequence graph $L((G,$ label$))$ is the label of each path in the set paths(v_s, v_e).

3.5 Alignment

3.5.1 Edit Distance

An edit distance is a measure of How many mutations are required to convert one sequence into another. Different edit distance measures exists, varying in the set of allowed mutations. The most common is the *Levenshtein distance* [**?**], allowing single character substitutions, insertions and deletions.

Finding the edit distance between two sequences, and which set of edits this edit distance corresponds to, is of central importance in bioinformatics since it can give an estimate of how related the two sequences are and which mutations have separated them. The process of finding these estimates is called *sequence alignment* and is important for this thesis in two respects. Firstly, it is one of the earliest and most intuitive applications of sequence graphs, and secondly it is an important part of *mapping* which will be covered in the next section.

In the following we will refer to the edit distance between two sequences, S and T, as $D(S,T)$, meaning the Levenshtein distance unless specifically mentioned. The concepts discussed are however generalisable to other (weighted) edit distances by small changes.

The set of edits between two sequences can be represented with an *alignment block* where "-" symbols are inserted into the sequences to represent indels, (see figure 3.5.2.

3.5.2 Pairwise Sequence Alignment

Finding the edit distance between two strings is easy if indels are not considered. It is merely the process of counting the number of mismatches between them. Indels introduce a dependency since an insertion at position i affects the pairings of all the symbols after position i. The number of meaningful combinations of insertions and deletions grow exponentially with sequence length, so exploring all of them is not a viable solution even for short sequences. The problem is however tractable since the best alignment of two sequences $S[:m], T[:n]$ is either the best alignment of $S[:m-1], T[:n]$ with an insertion at the end, or of $S[:m], T[:n-1]$ with a deletion at the end, or of $T[:n-1], T[:m-1]$ with a match or substitution at the end. Letting $d_{i,j} = D(S[:i], T[:j])$ and $m_{i,j}$ be zero if $S[i] = T[j]$ else 1, we can write this as the recurrence relation:

$$d_{k,l} = min \begin{cases} d_{k-1,l-1} + m_{k-1,l-1} & \text{(match/substitution)} \\ d_{k-1,l} + 1 & \text{(insertion)} \\ d_{k,l-1} + 1 & \text{(deletion)} \end{cases}$$

The fact that the edit distance to a empty string is just the sequence length gives $d_{k,0} = k,\ d_{0,l} = l$. The Needleman-Wunch algorithm uses dynamic programming to calculate the matrix of edit distances d_{kl}, where d_{mn} is the edit distance between S and T. In order to find the specific edits, a backtracking algorithm is used that starts at the $(i, j) = (m, n)$ corner and finds out which of the possible predecessors $(i-1, j), (i-1, j-1), (i, j-1)$ contributed to the (i, j) edit distance. Then repeating this until the $(0, 0)$ corner is reached. Each of these steps correspond to a column in the alignment block. Figure 3.5.2 details the Needleman Wucnch algorithm. This algorithm can be adopted in a number of ways in order to solve related problems. Notably, by using an affine gap penalty one can reduce the cost of long indels compared to many small indels [**?**], or one can use positive scores for matches to be able to find subsequences that align well to each other [**?**].

An adaption relevant for this thesis, is to find the a subsequence of one sequence R that minimizes the edit distance to another Q. I.e find $\min_{k,l}(D(R[k, l], Q)$. This can be done by changing just the initial conditions of the Needleman-Wunch algorithm, to remove the cost of gaps at the beginning and end of R. We set $d_{k0} = 0$ and start the backtracking algorithm at (m, l) where $j = argmin_i d_{mi}$. Figure**??** shows an example of this algorithm. This algorithm provides three results: the coordinate k, l of the subsequence of R most similar to Q, the edit distance from that subsequence to Q, and the set of edits contributing to the edit distance. In this way it solves in a exact, but slow way the problem of the next chapter, namely mapping a read Q to a reference genome R.

3.5.3 Sequence Graph Alignment

The framework used to align sequences extends naturally to acyclic sequence graphs [**?**, **?**]. We define the alignment of a sequence S to a sequence graph G as the alignment of S to a sequence $T \in L(G)$ in the language of G which yields the lowest edit distance. Similarily, the alignment of two sequence graphs G, H, is the alignment of a sequence $S \in L(G)$ to a sequence $T \in L(G)$ that yields the lowest edit distance.

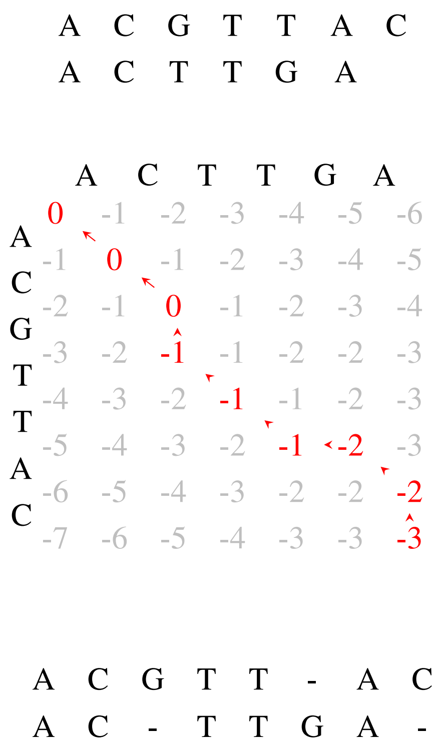
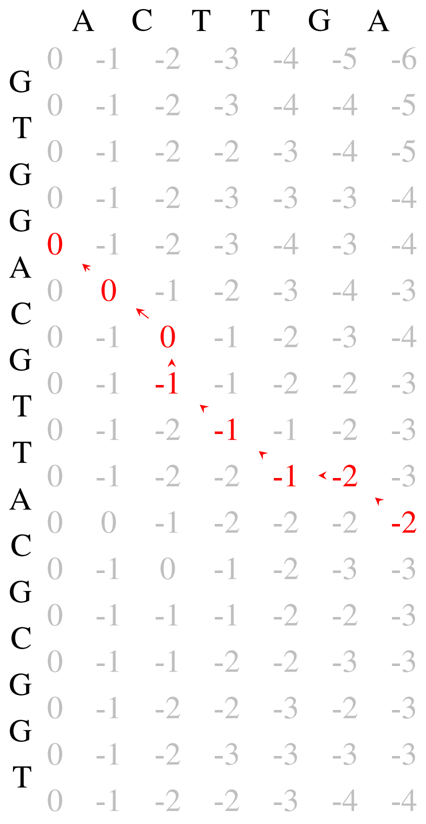


Figure 3.1: Figure showing the alignment of two sequences using Needleman-Wunch. Each cell in the matrix corresponds to the edit distance between prefixes of S and T. The red path shows the backtracking, resulting in an alignment block where each diagonal arrow gives a symbol from both sequences, while horizontal or vertical arrows give an insertion or deletion.

G T G G A C G T T A C G C G G T

A C T T G A



$$
\begin{array}{c|ccccccc}
 & & A & C & T & T & G & A \\
\hline
 & 0 & -1 & -2 & -3 & -4 & -5 & -6 \\
G & 0 & -1 & -2 & -3 & -4 & -4 & -5 \\
T & 0 & -1 & -2 & -2 & -3 & -4 & -5 \\
G & 0 & -1 & -2 & -3 & -3 & -3 & -4 \\
G & 0 & -1 & -2 & -3 & -4 & -3 & -4 \\
A & 0 & 0 & -1 & -2 & -3 & -4 & -3 \\
C & 0 & -1 & 0 & -1 & -2 & -3 & -4 \\
G & 0 & -1 & -1 & -1 & -2 & -2 & -3 \\
T & 0 & -1 & -2 & -1 & -1 & -2 & -3 \\
T & 0 & -1 & -2 & -2 & -1 & -2 & -3 \\
A & 0 & 0 & -1 & -2 & -2 & -2 & -2 \\
C & 0 & -1 & 0 & -1 & -2 & -3 & -3 \\
G & 0 & -1 & -1 & -1 & -2 & -2 & -3 \\
C & 0 & -1 & -1 & -2 & -2 & -3 & -3 \\
G & 0 & -1 & -2 & -2 & -3 & -2 & -3 \\
G & 0 & -1 & -2 & -3 & -3 & -3 & -3 \\
T & 0 & -1 & -2 & -2 & -3 & -4 & -4
\end{array}
$$

A C G T T - A

A C - T T G A

Figure 3.2: Figure showing the alignment of a query sequence Q to a reference sequence R, Each cell in the matrix corresponds to the edit distance between a prefix of S and a subsequence of R. The red path shows the backtracking, starting at the lowest value of the last column, resulting in an alignment block where each diagonal arrow gives a symbol from both sequences, while horizontal or vertical arrows give an insertion or deletion.

If we let $d_{ij} = \min_{p_g \in \text{paths}(v_0,v_i), p_h \in \text{paths}(w_0,w_k)}(D(\text{label}(p_h), \text{label}(p_g)))$ we get a recurrence relation:

$$d_{ij} = \min_{v_k \in e^- v_i, w_l \in e^- w_l} d^*(i,j,k,l)$$

$$d^*(i,j,k,l) = \min \begin{cases} d_{il} + 1 \\ d_{kj} + 1 \\ d_{kl} + m_{kl} \end{cases}$$

$$m_{ij} = 1 \,\text{if}\, \text{label}(v_i) \neq \text{label}(w_j) \,\text{else}\, 0$$

This is the same as for ordinary sequence alignment, except that all predecessor nodes of v_i and w_k have to be considered, not only $i-1$ and $j-1$ as in the linear case. If the graph is acyclic, then all the d_{ij} can be calculated using dynamic programming, without incurring any infinite loops.

An alignment between two sequences S, T can be represented by a sequence graph in a meaningful manner in that one can construct a sequence graph $AG(S,T)$ from the alignment of the sequences in such a way that

$$\forall (R \in L(AG(S,T)))\, [D(S,R) + D(R,T) = D(S,T)]$$

. For an optimal alignment, this means that all sequences recognized by the sequence graph have the property that the sum of the distance to the original sequences is as low as it can be. These sequences thus represents combinations of S and T that are natural estimates of an ancestor of the two sequences. Thus aligning a sequence S to an alignment graph $AG(T,R)$ can be seen as aligning S to the best fitting ancestor sequence of T and R Similarily, aligning two alignment graphs can be seen as finding the ancestors of two pairs of sequences that fits best together.[**?**]. This is illustrated in figure 3.4.

The sequence graph alignment algorithm can also be adapted to find the subsequence of a sequence graph G_R that minimizes the edit distance to a sequence Q. This solves the problem of mapping a read to a graph, which will be discussed further in section **??**

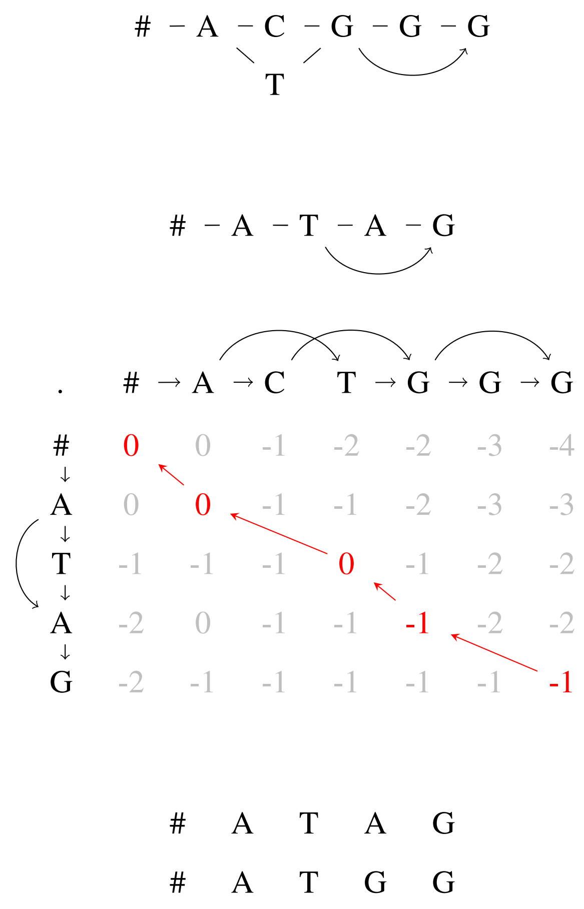


Figure 3.3: Figure showing the alignment of two sequence graphs. Marked in red is the path taken during backtracking. The result is the best alignment of a sequence from the language of each sequence graph. This alignment can again be represented as a sequence graph

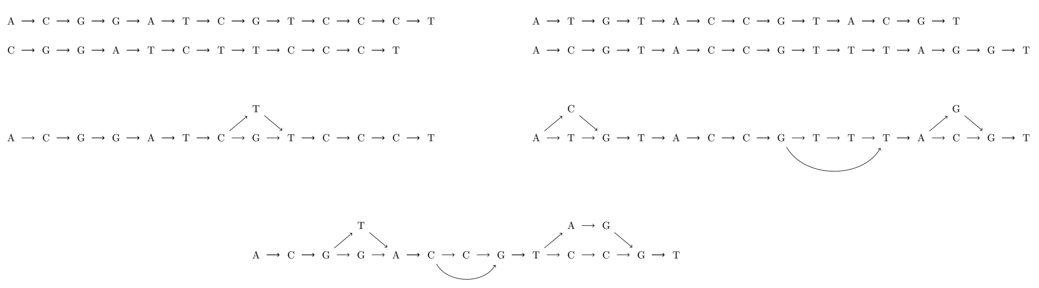


Figure 3.4: Iterative sequence graph alignments of four sequences evolved in two generations from *ACGTACGTACGT*. The sequences are represented as linear sequence graphs (a) and pairwise alignment is performed on the closest pairs yielding two sequence graphs (b). These two seqence graphs are then aligned 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 language of the final sequence graph

3.6 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.

3.6.1 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_iS_i.
- If r_i is the nth occurence of that letter in R, the suffix r_iS_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*)

3.6.2 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*).

3.7 Peak Calling

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.

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

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 described in Paper 3 and Paper 4.

3.7.1 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 )
```

Chapter 4

Summary of Papers

Chapter 5

Discussion

Using sequence graphs as reference structures

5.0.2 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.

5.0.3 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.

5.0.4 Pseudo-linearity

5.1 Recent Developments

Haplotype aware mapping