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

3.4 Genomic Variation

Genomic variation is one of the fundamental aspects of biology. Difference in the DNA-sequence between two individuals can lead to a change in the translated 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.

Within a species, the genomic variation between individuals are often limited by evolutionary conservative pressure, meaning that the difference in DNA-sequence between viable specimens of the same species are often small and does not lead to big changes in neither the structure of the DNA or the translated proteins. This limitation has made possible the use of *reference genome*s for a species, a DNA-sequence though to represent the generic sequence for that species, where individual variation from the reference sequence are though to be small.

Reference genomes have helped make sense of sequencing data, that have been dominated by large numbers of small sequence fragments. Without a good reference genome, one would need to fit all those sequence fragments together like a puzzle, in a process called *assembly*. Using a reference genome, one can instead find the best sequence match for each read in the reference sequence, and thus find both where each read belongs in the genome, and also how the sequence differs from the reference. Finding the position of each read can give context to

them, since the location of biologically important parts of the genome can be represented on the genome. For instance, if the sequencing of an individual maps to a location within a the known location of a protein coding gene and differs from the reference sequence, it's possible that that individual has a genomic variant that alters the function of that protein.

Since this process of *mapping* sequence reads to the reference is such a fundamental step in many biological analyses, the quality of the reference sequence has been of high importance. Thus, since the dawn of human genome sequencing in (*year*), the Genome Reference Consortium has released (*n*) versions of the human reference genome alone. The latest version of the human reference genome (GRCh38), highlights some shortcomings of representing the reference of a species as a single sequence. GRCh38 includes, in addition to the traditional linear reference sequence, a set of alternative sequences from areas of the genome where there are known variations of the DNA-sequences which makes it problematic to map reads from those areas to the reference sequences. Secondly the new version included changes that disrupted the coordiante system of the reference. This has lead to a backward incompatibility that has prevented a widespread adoption of the new reference.

3.5 Mapping Bias

Mapping sequencing reads to the reference entails finding subsequences in the reference sequence that are highly similar to the sequenced reads. The match can be inexact due to either the actual DNA-sequence being different or sequencing errors that can substitute one nucleotide with another. It is necessary to set a limit to how dissimilar the reference subsequence can be in order to produce a match due to computational complexity and that allowing a high level of divergence can lead to a large number of matches. For instance, *BWA-mem* by default requires a shared subsequence of at least 19 bp in order to produce a match. This means that reads from regions with much variation from the reference can be unmappable using standard mapping software, and be susceptible to small amount of sequenceing error yielding them unmappable. This means that the resemblence of the sample to the reference genome will lead to better mapping quality, and that some regions of the genome will have a lower mapping rate than others.

3.6 Geometry of the reference genome

As well as being an indexable lookup table for sequencing reads, a reference genome provides a coordinate system and a geometry for sequence data that allows us to look at different sequence elements in conjunction. Most important is the analysis of overlap and distances between subsequences. For instance, the location of a potential transcription factor binding site, predicted from a ChIP-seq experiment, can be compared to the positions of known genes, pre-

dicted from amongst others RNA-seq experiments, to determine which gene is most likely regulated by the binding of a TF to the binding site. The distances between subsequences is also used in some mapping tools themselves, as a way to evaluate the match of a read to a reference subsequence. For instance Minimap2 [**?**] uses the relative positioning of kmer matches to the reference to find which chain of kmer-anchors match the read sequence best. The distance between two subsequences on the reference is invariant to SNPs, since they do not affect distances. However indels and especially structural variants affect the distances between subsequences, and can thus change the outcome of any analysis involving distances. For instance, a structural variant that changes which gene a regulatory element affects.

ENCODE data, og gen lister.

3.7 Mathematical Framework

3.7.1 Strings

3.7.2 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^n_{DNA}. 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.

3.7.3 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 : $V \rightarrow \Gamma$ 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 $L((G, \text{label}))$ is the label of each path in the set paths(v_s, v_e).

3.8 Alignment

3.8.1 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 3.8.2.

3.8.2 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

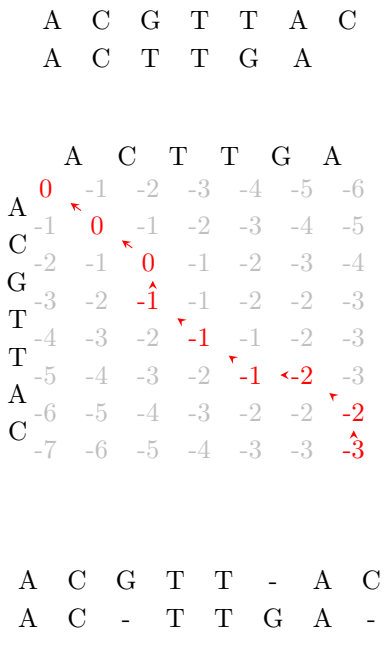

Figure 3.1: Figure showing the alignment of two sequences using Needleman-Wunch. Each cell in the matrix corresponds to the edit distance between prefixes of S and T. The red path shows the backtracking, resulting in an alignment block where each diagonal arrow gives a symbol from both sequences, while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

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 3.8.2 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.

3.8.3 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.

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

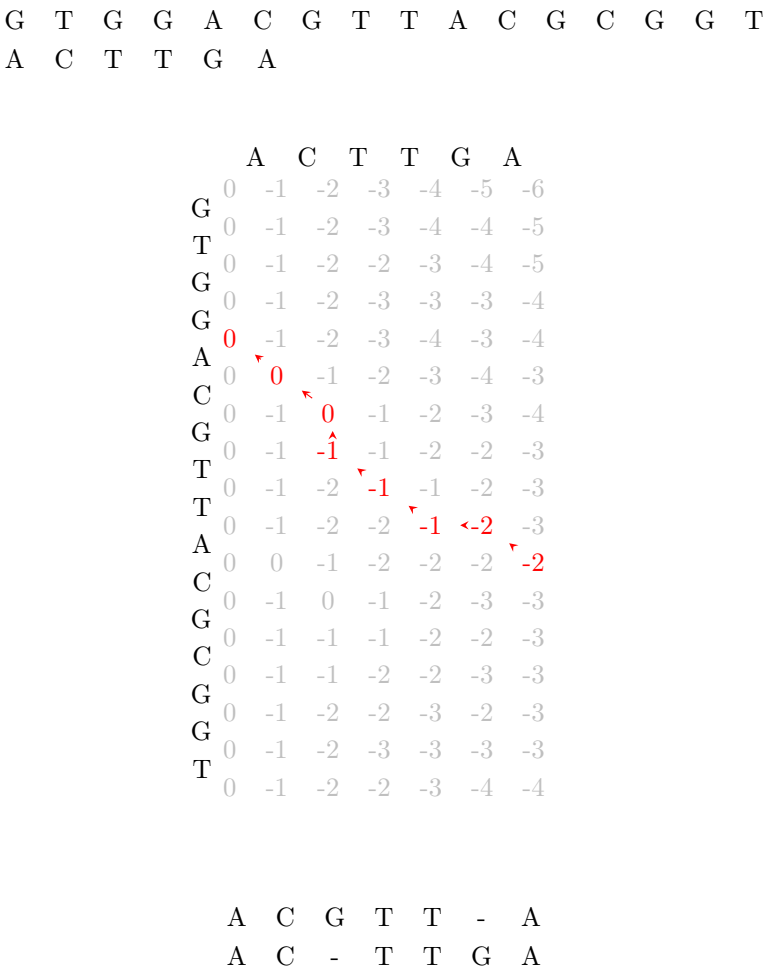

Figure 3.2: Figure showing the alignment of a query sequence Q to a reference sequence R, Each cell in the matrix corresponds to the edit distance between a prefix of S and a subsequence of R. The red path shows the backtracking, starting at the lowest value of the last column, resulting in an alignment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

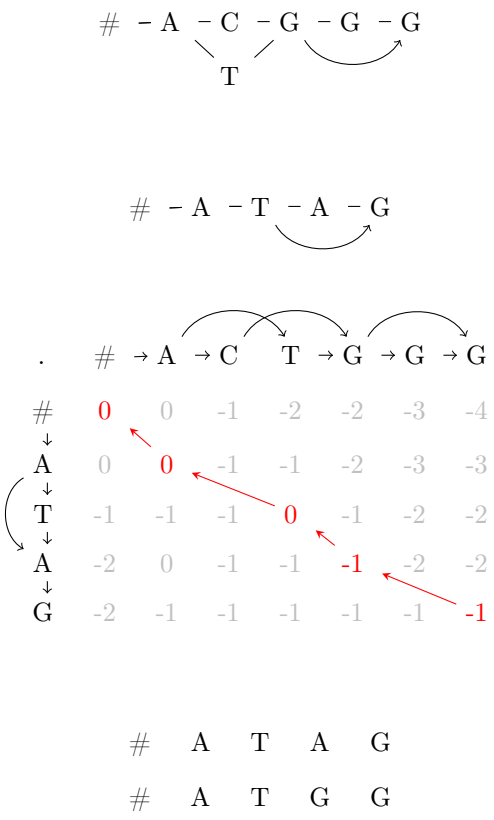

Figure 3.3: Figure showing the alignment of two sequence graphs. Marked in red is the path taken during backtracking. The result is the best alignment of a sequence from the language of each sequence graph. This alignment can again be represented as a sequence graph

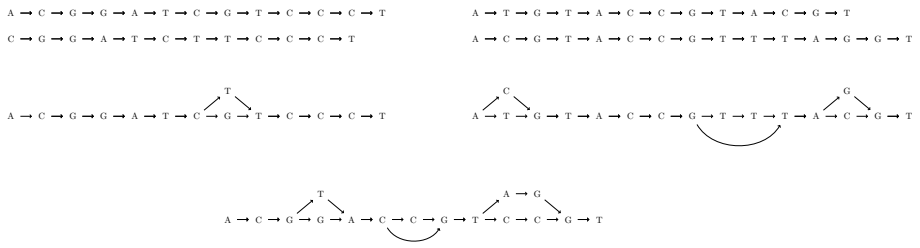

Figure 3.4: Iterative sequence graph alignments of four sequences evolved in two generations from $ACGTACGTACGT$. The sequences are represented as linear sequence graphs (a) and pairwise alignment is performed on the closest pairs yielding two sequence graphs (b). These two seqence graphs are then aligned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

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 **??**

3.9 Mapping

The alignment methods described in the previous section do not scale well to large sequences. Aligning a read $Q[:m]$ to a reference genome $R[:N]$ will require computing mN values. A typical NGS sequencing experiment typically yields hundreds of thousands of reads of length 100 which needs to be located in the human reference genome of length 3 billions. This would need to calculate 10^{16} values, which is impossible? even on modern computers.

This has led to much development the last 13 years in developing alignment methods that avoid the complexity of the exact dynamic programming methods. This has been achieved mainly by two measures, creating searchable indexes of the reference genome and using heuristics to limit the search space of possible alignments.

In the following is a brief description of the developments in this field, with focus on the aligner used in this thesis' projects: BWA-mem [**?**]. After this we will look at the methods used in for graph alignment with focus on *vg*.

3.9.1 Linear Mapping

As in section **??**(*Define this explicitly in alignmetn*), we define the mapping problem as finding the interval i_R for wich the edit distance $D(R[i_r], Q)$ is smallest, and the alignment $A(R[i_R], Q)$.

Indexing

An essential tool for quick alignment is to have a searchable index of the reference genome which is capable of returning sets of interval-pairs from the reference and query sequence for which the reference sequence is identical or near identical to the query sequence. The indexes used by different tools vary, but can mainly be divided into fixed-length(kmer) indexes [**?**, **?**, **?**, **?**] and variable-length indexes [**?**, **?**, **?**, **?**, **?**]. Fixed length indexes typically uses hash tables to store the location of all, or subsets of all, kmers (*define kmer*) in the reference sequence. Variable length indexes usually uses a variation of the *full text minute index* (FM-index) [**?**], described below. The main idea covered here is the seed-and-extend paradigm. This idea consists of finding a set of exact matches to the reference sequence, and using these as anchors for using dynamic programmic based alignment of the query. We let $EM(Q, R)$ be the set of all exact matches, represented by the tuples $\{(q, r, l) \mid Q[q : q + l] = R[r : r + 1]\}$. Since the dynamic programming methods can still be computationally expensive, the goal is to make the seeds as small a subset of exact matches as possible, while still making sure that the true alignment covers one of the seeds.

FM-index

The FM-index uses a succinct representation of suffix array [**?**] and Burrows-Wheeler transform [**?**] in order to find exact matches of a query string S in $O(|S|)$ time (see figure **??**). Thus if a read has one or more exact matches in the reference sequence it can be found directly using the FM-index. For inexact matches, it can also be used to find an exact alignment of Q and R in $O(|R|^{0.628}\,|Q|)$ time [**?**, **?**]. Some algorithms also use the FM-index to find exact matches for permutations of the query sequence [**?**, **?**], but for longer reads the search space gets too big when allowing for indels.

In order to handle indels in longer reads, the main methodology is to use the FM-index to find exact matches between substrings of the query and reference sequence, called seeds. And then aligning the reads using dynmaic programing to intervals surrounding the seed matches on the reference sequence [**?**], often called the seed-and-extend paradigm.

One way of finding seeds is to find Maximal Exact Matches (MEM) [**?**, **?**]. Maximal Exact Matches are Exact Matches that cannot be extended in either direction, ie.

$$MEM(Q, R) = \{(q, r, l) \in EM \mid (q, r, l + 1) \notin EM \wedge (q - 1, r - 1, l + 1) \notin EM\}$$

BWA-mem furthers this concept to SuperMaximal Exact Matches (SMEM) [**?**]. A SMEM is a MEM where the query interval cannot be extended either side

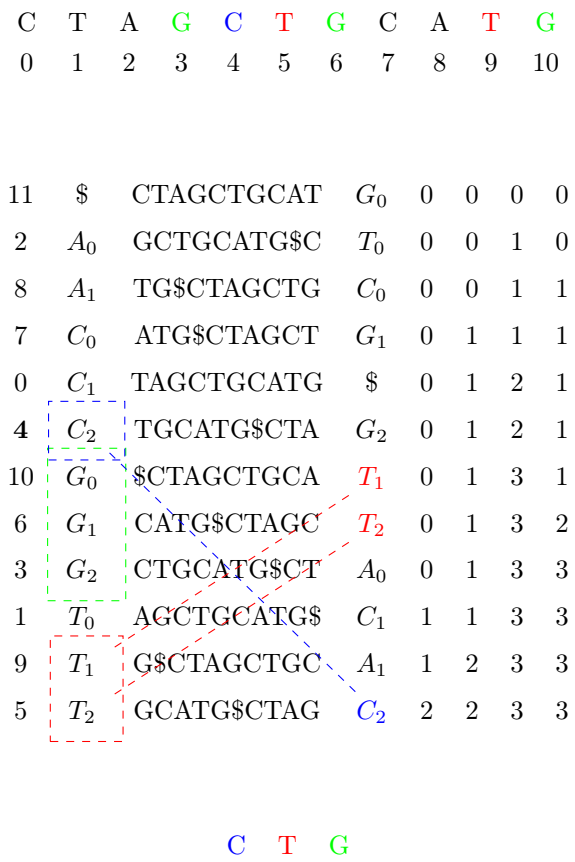

Figure 3.5: Illustration of backward extension using the last-first (LF) property of the FM index. The rows represent sorted suffixes of the reference. The SA column holds the indices in the reference sequence for each suffix, the F column holds the first character of each suffix, while the L column holds the preceding character of each suffix. Subfixes gives the occurances of each character in each column. I.e. T_i is the ith occurance of T in that column. Finding the string CTG is done by (1) starting with the range of all G's in the F column; (2) finding all T's in this range in the L column; (3) mapping by occurance number those T's to the F column; (4) mapping the C'c in the current range in the L folder by occurance number. The end result is in this case a single row which represents position 4 in the reference sequences.

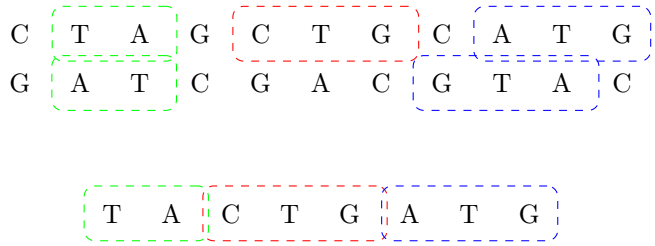

Figure 3.6: SMEMs found between query Q and both strands of reference R. Note that the MEM $(q, r, l) = (2, 0, 2)$ (CT) is not a SMEM, since it is contained in the SMEM $(q, r, l) = (2, 4, 3)$

and still yield an Exact Match **??**.

$$SMEM(Q, R) = \{(q, r, l) \in EM \mid \nexists r^*[(q, r^*, l + 1) \notin EM \vee (q - 1, r^* - 1, l + 1) \notin EM\}$$

SMEMS are natural to use as seeds as they cover for each subsequence in Q the longest exact match in R. They are however vulnerable for spurious long matches hiding shorter exact matches. To account for this BWA-mem allows an option to split long SMEMs into shorter MEMs if they are longer than a certian threshold. Splitting SMEMs like this increases accuracy, since it increases the number of seeds, but can negatively affect performance. In order to find SMEMS, an adaption of the FM index is used, the FMD index, where $FMD(R) = FM(R * \bar{R}$.

Prioritizing and Extending

The seeds found from the index are next used as seeds for DP based alignment. Q is then aligned against an area around $(r, r + l)$ for each seed (q, r, l). Since this is a computationally expensive step, further limiting the set of seeds is advanteous. BWA-mem does this by *chaining* the seeds. This is done by grouping approximately colinear, nearby seeds into chains, and removing small chains that overlap with larger chains. Approximately colinear means that $|(q_1 - q_2) - (r_1 - r_2)| < w$ for some set threshold.

3.9.2 Graph Mapping

Mapping to a graph based reference is similar to the linear case, except that instead of finding a linear interval, the goal is to find a graph interval i_r such that $D(\text{label}(G_R(i_r), Q))$ is minimized.

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. Below is a brief description of the methodology used by *vg*.

vg

vg uses much the same methodology as BWA-mem to align reads. It uses the GCSA2-index to find SMEMs, uses chain filtering to filter the SMEMs, and uses the graph adaption of Smith-Waterman to extend the seeds. *vg* is able to align reads to more complicated graph structures than the simple directed sequence graphs considered in this thesis. For simplicity the descriptions below will be contained to simple graphs, which entails that the GCSA index described is GCSA1[**?**] which is only able to index directed sequence graphs.

GCSA

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.

GCSA uses the same concept of LF mapping as the FM-index does. Here the F column holds the label of each node in the graph, sorted by the suffix starting from that node. The L column holds the label of the corresponding node's predecessor nodes. The problem with this setup is that there are many suffixes starting from each node, depending on which path is taken in the graph. To resolve this problem, the graph needs to be expanded so that for each node, all suffixes starting from that node shares a prefix that is not found from any other node in the graph(see figure 3.9.2).

For complicated regions in the graph, this expansion procedure gets too costly, and the GCSA index therefore needs to prune edges in such areas in order to be able to index it. This means that not all possible sequences in the graph gets indexed. Even after pruning, this procedure is costly and makes the GCSA index significantly slower to create, and more memory consuming than the FM-index.

Since the gcsa index provides the same functionality as an FM-index, it can be used to find SMEMs in much the same manner. These SMEMs are in *vg* used as seeds.

GCSA2 solves the problem in a slightly different way that allows for indexing general variation graphs. This is based on succinct de-Bruijn graph [**?**, **?**] structure which sets a limit to the length of the unique prefixes by allowing for false positive edges to occur in the index. This means that results from an index lookup needs to be validated by traversing the graph.

Filtering and Extending

vg employs a similar method as BWA-mem for filtering out seeds: chaining the seeds and filtering out small chains that overlap bigger chains. The chaining

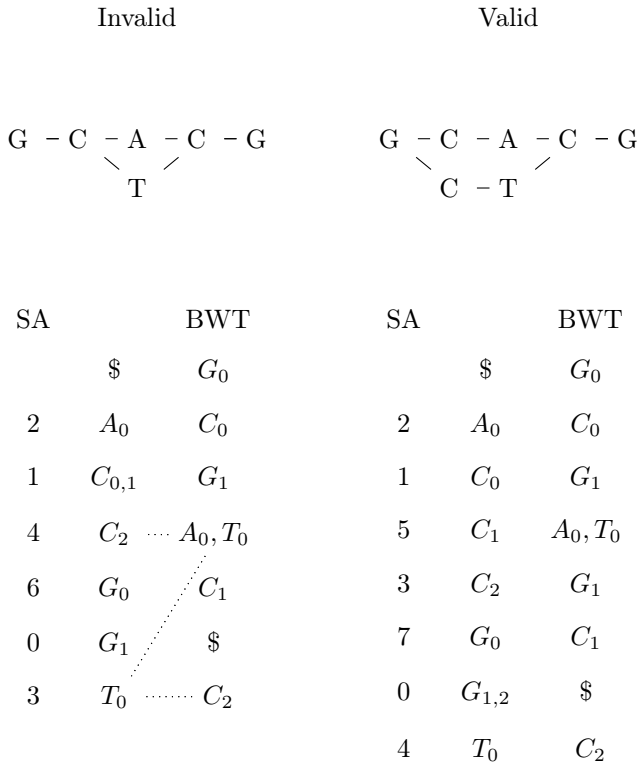

Figure 3.7: Valid and invalid GCSA index. In the first graph, the two edges from C to A and$T$ makes a direct GCSA index impossible due the other C with an edge to a G. Both T and G needs to point to the same row in the index. In the valid example, the C node have been duplicated. The bifurcation now happens at an earlier point, and the GC now present is unique so that the the LF mapping is valid.

procedure is however more complicated on a graph, and involves clustering the seeds by position in addition to a Markov Model based method to find approximately colinear seeds.

3.10 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.

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

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.

3.10.1 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