UiO : Department of Informatics
University of Oslo

Read alignment against a graph-based reference genome

Fuzzy searching in large and complex structures

Esten Høyland Leonardsen

Master's Thesis Spring 2016

Read alignment against a graph-based reference genome

Esten Høyland Leonardsen

1st March 2016

Abstract

Contents

List of Figures

List of Tables

Preface

Part I

Introduction

Chapter 1

Background

1.1 DNA

Deoxyribonucleic acid (DNA) is a molecule which allows organisms to store and pass on genetic information. The molecule is stored in the cells and encodes proteins which regulates the vital functions of the organism. DNA is able to replicate itself and can thus live on for generations through reproduction.

The DNA of an individual is made up by two complementary strands of nucleotides bound together in a double helix. The nucleotides of DNA can contain the bases Adenine, Cytosine, Guanine or Thymine, typically denoted A, C, G and T. Complementary in this context means that instead of one singular sequence of bases DNA is made up by a sequence of paired bases, A's with T's and C's with G's, called base pairs (bp). Due to the chemical structure of the molecules making up a single strand of DNA each strand can be said to have a direction, upstream towards the 5′ end or downstream towards the 3′ end. Two complementary strands have opposing directions and are thus called the reverse complements of eachother.

The size of DNA varies across species, from a couple of thousand basepairs (kb) in some viruses to several hundred billion basepairs (gb) in larger, more complex organisms[**Citation needed?**]. The human genome comes in at the higher end of this range, with a length of roughly 3 gb. MORE

1.1.1 Genomic variations

Over the span of time DNA is subject to change. Through random mutations and recombination a genetic sequence can be changed either within an individual or as a product of reproduction. The fact that these changes are able to survive and propagate through generations leads to a genepool where even though the DNA comes from a common ancestor, different individuals will have different variants of the original sequence. These variations form the basis for the division into species, but even within species alot of natural variation will occur.

The least complex of these variations are Single Nucleotide Polymorphisms (SNPs), where a single nucleotide has changed between two individuals, and insertions and deletions (Indels) where either one or a short sequence of bases have appeared or disappeared in the DNA of an individual. Longer and more complex structural variations can also occur when a larger part of a chromosome breaks free and disappears completely or inserts itself in a different place or the opposite direction.

1.1.2 Alignment

The fact that a genome is built by discrete entities, the bases A, C, G and T, means that any DNA sequence can be represented by a text string. The double-stranded nature of DNA could be encoded into the string, but as one side can easily be derived from the other representing one of the strands is usually expressive enough. The process of determining genetic variation between two separate individuals can then be seen as the problem of finding the similarities and differences in the two corresponding text strings. Motivation?

Levensthein difference

There exists several ways of determining the difference, also called the edit distance, between two strings mathematically. The main difference between the approaches are which operations are possible on the two strings and how the result is scored. The most common technique is called Levenshtein distance, which allows the operations deletion, insertion, and substitution. All of the preceding operations works on a single character, and one operation by itself yields a distance of 1. Finding the optimal Levenshtein distance can be done by dynamic programming in linear time.

DNA sequence comparison

(Needleman-Wunsch)

1.1.3 Reference genomes

To say that an individual of a species deviates from the norm in any way there has to be a standard to compare it to, a reference genome. For humans the current reference genome is GRCh38, developed and maintained by the Genome Reference Consortium. Something about the represented variation

1.2 Graph-based reference genomes

As sequencing technology improves and cost decreases we are able to sequence an increasing number of genomes. A natural consequence of sequencing more genomes is discovering a larger number of variations.

When mapping reads against a reference genome, knowledge of possible variants will provide a better basis for completing a correct mapping of new sequences[2]. A natural way to represent this variation is through a graph, where different sequences share regions they have in common and branch out wherever they diverge from eachother. Motivation

1.2.1 De Bruijn graphs

1.2.2 Mapping to a graph

When a reference genome is represented by a single sequence of letters, each letter can be uniquely identified by its position. Variations can then be stored based on the position they occur in, or the position they start in if they span several bases. When using a graph-based reference the nodes are not a part of any naturally occuring coordinate system. If one also imagines the reference should be able to merge in new data at any given point the relationship between nodes might change and yield a fixed mapping scheme generated for the old graph stale and incorrect.

1.2.3 Context driven mapping

In the article *Mapping to a reference genome structure*[12], Benedict Paten et al. introduce the concept of context-driven mapping. They define a context for a node N in a reference as a triplet (L, B, R) where B is the base contained in the node, L is a set of strings generated by traversing the paths leading in to N and similarily R is a set of strings generated by traversal of the paths going out from N. A unique context is a context mapping to exactly one node in the reference genome graph.

1.2.4 SOMETHING FROM VG

MORE

1.2.5 Alignment against a graph

Aligning two sequences against eachother can as mentioned be done by dynamic programming in linear time by searching a 2-dimensional search space. A naive approach to the alignment problem could be to do such linear searches against all possible sequences present in the graph, represented by traversal of all complete paths from the start-node to the end-node. However the number of possible paths would grow exponentially in relation to the variants represented in the graph and the complexity of this approach would quickly grow out of control.

Dynamic programming

The most obvious problem with the naive solution is that many nodes are visited several times as they are traversed as members of separate paths. Lee, Grasso and Sharlow propose an algorithm for multiple sequence

alignment on Partial Order Graphs (MO-PSA) in their paper *Multiple sequence alignment using Partial Order Graphs***??**. The general idea is to use a regular string alignment algorithm, Needleman-Wunsch, extended to multiple dimensions. When determining the score for a given position for a node in the graph, instead of looking at the previous indexes MO-PSA looks at paths from preceding nodes.

Context driven alignment

Chapter 2

Algorithm

2.0.6 Definitions

Definition 1 (Graph-based reference genome (graph))
A pair $G = \{V, E\}$ where V is a set of vertices and E is a set of edges. $|G|$ denotes the number of vertices of G.

Definition 2 (Vertice)
A pair $v = \{b, i\}$ where $b \in \{A, C, T, G\}$ and i is a unique index. Written $v_i = b$

Definition 3 (Edge)
An ordered pair $e = \{i_s, i_e\}$ where both elements are indexes for vertices.

Definition 4 (Complete Path)
An ordered list P of indexes such that for all consecutive ordered pairs $\{i_x, i_{x+1}\} \in P$ there exists an edge $e = \{i_x, i_{x+1}\}$.

Definition 5 (Path)
An ordered list L of indexes such that for all consecutive ordered pairs $\{i_x, i_{x+1}\} \in L$ there exists a complete path P which starts at $\{i_x\}$ and ends at $i_{x+1}\}$.

Definition 6 (Input sequence)
A string s over the alphabet $\{A, C, T, G\}$. The length of the string is given by $|s|$. An individual character on position x is referenced by s_x

Definition 7 (Mapping score)
A score produced by mapping two characters $c_1, c_2 \in \{A, C, G, T\}$ against a scoring matrix

Definition 8 (Path score)
A score produced by traversing a path P through a graph G to create a linear sequence, scoring gaps according to the gap penalties given by a scoring schema.

Definition 9 (Alignment)
Given a sequence s and a graph G, a list A of indexes such that every $a_x \in A$ is either a valid index for a vertice in G or 0. 0 indicates an unmapped element of the input sequence

Definition 10 (Alignment score)
Given a sequence s a graph G and an alignment A, the score produced by combining mapping scores for the pairs $\{a_x, s_x\} for 0 <= x < |s|$ with the path score for the path(s) provided by A aligned against both G and s.

Definition 11 (The optimal alignment score-problem)
For any pair $\{G, s\}$, where G is a graph and s is a sequence, find the alignment A which produces the highest alignment score. If multiple max scores: Provide all or chose one?

2.0.7 Overview

2.0.8 Precomputation of the graph

There are two data structures needed for aligning a string against the graph, a suffix tree for left contexts and a suffix tree for right contexts. Before either of the two are built the algorithm needs to decide a length for the contexts. Currently in the tool**??** there are two ways of setting the context length: A user given parameter or an approximation based on the probability of sharing contexts SUPPLEMENTARY?. The length of a context does not impact the quality of the alignments found by the algorithm PROOF IN SUPPLEMENTARY? but will have an impact on the runtime. A further discussion on context lengths can be found in DISCUSSION SOMEWHERE

When a context length $|c|$ is set, the algorithm can start building the index. Two sets of strings, a left context set and a right context set, is generated for every node in the graph G. The generation of the two sets happen by the same procedure, by swapping around the starting point and the direction of the iteration. When creating left contexts the algorithm starts in the start-node of G and traverses following the direction of the edges, for right contexts the opposite is done. Apart from this the two are equal. To generate the context set $c(n_x)$ for a given node n_x the algorithm looks at every string $c \in c(n_y)$ for every incoming neighbouring node n_y. Every c is modified into a new context string c' by trimming away the last character and prefixing the context with the character b stored in n_y. All the generated c' is then added to $c(n_x)$. As sets per definition does not allow duplicates the impact of a branching occuring in the graph will fade away after exactly $|c|$ steps as the difference is trimmed away (see Fig. 2.1), and thus avoid explosive exponentiality in the context set sizes.

The iteration starts in the node defined as the starting point which has the empty string ϵ as its only context and whenever a node is finished it enqueues everyone of its outgoing neighbours in a regular FIFO queue. If a node has more actual incoming neighbours than incoming neighbours which are finished generating contexts, the node puts itself back in the queue. The algorithm halts when the queue is empty. Every node has to be visited exactly once to generate its context and as the procedure runs twice to generate both sets the total runtime for the operation is $O(2|G|)$.

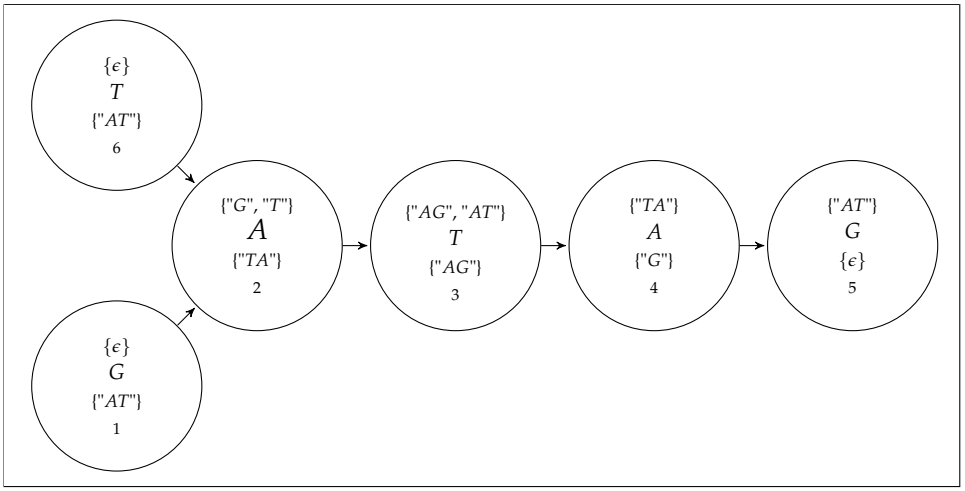


Figure 2.1: A small reference graph with left contexts (top) and right contexts (bottom) and $|c|$=2 explicitly shown

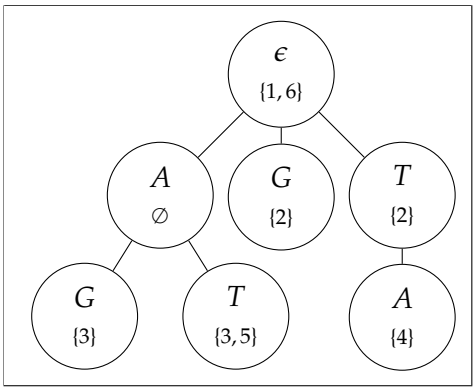


Figure 2.2: The left suffix tree corresponding to the graph in **??**

After generating the two context sets for every node, the elements of each one is inserted into their corresponding suffix tree. In theory every node can have $4^{|c|}$ contexts in each set. When the graph is more or less linear with few branches a more fair approximation is $B * |c|$ where B is the observed branching factor. The current implementation in the tool **??** uses a naive suffix tree where insertion is $O(|c|)$. This is done for every node in the graph, yielding a total time complexity of $O(|G|B|c|^2)$. A discussion on more efficient suffix structures can be found in SOMEWHERE IN DISCUSSION. Every suffix is stored as a key with the index of it's originating node as a value. The total runtime for building a searchable index for a graph is $O(3|G||c|^2B)$

2.0.9 Generating the new graph G'

Creating G' is the process of determining which nodes qualifies as candidate nodes for a given input string s and combining them correctly. For every character $s_x \in s$ a left-context string and a right-context string is generated by looking at the $|c| + maxPossibleGapGivenThreshold(T)$ surrounding characters. The two strings are treated as contexts, one left and one right, and used as a basis for a fuzzy search in it's corresponding suffix tree.

The search is a recursive function based on PO-MSA. The root node is supplied with a one-dimensional scoring array corresponding to the context string c, which is initialized with all zeroes. Then, for every child, a new scoring array is computed by regular edit distance rules: For each index i take the maximal score for either a gap in the graph, a gap in the string or matching the character c_i with the character contained in the child node (Reference actual code in supplementary?), (more explanation needed?). This new array is supplemented to the same recursive function in the child node. When a leaf node is reached the last index of the supplied scoring array corresponds to mapping the entire string c against the entire context achieved by concatenating the path traversed by the recursion. If the score is within the threshold T the indexes contained in the leaf node is added to the set of candidates.

The newly formed graph G' can be defined formally as:
$G'(G,s) = \{V', E'\}$ where V' is an ordered set of sets of length $|s|$ where each set V'_i is a set of nodes such that

$$V'_i = \{v_x | v_x \in G \wedge contextScore(v_x, s_i) > contextScore(s_i, s_i) - T\}$$

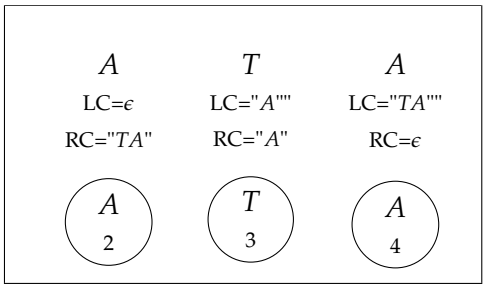
and E' is a list of weighted edges such that
$E' = \{e' = \{i_s, i_e, w\} | i_s \in V'_x \wedge i_e \wedge V'_y \wedge gapPenalty(x,y) < T \wedge gapPenantly(distance(i_s, i_e)) < T\}$

In theory every leaf node has to be visited in order to check the score for every represented context in the tree. In practice the tree can be pruned by cutting off the search whenever the *maximal potential score*(clumsy name) falls outside the threshold. The maximal potential score for a node is found by adding together the currently highest score in the scoring array with the maximal matching score for the remainder of the string. This reduces the number of nodes to be searched from $O(4^c)$ to (something alot smaller. Needs calculations).

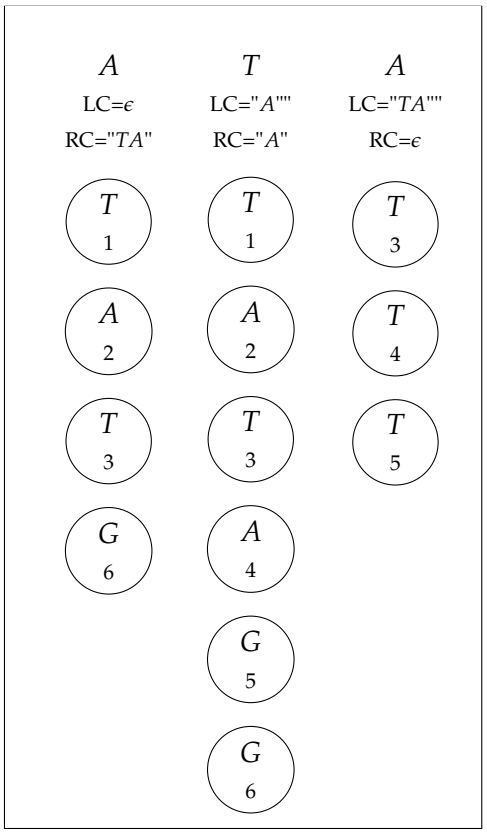
After the fuzzy search is concluded there are two sets of candidates for every index, one containing the nodes matching the left context and an equivalent for nodes matching the right context. These two sets are joined by combining the scores for every node present in both sets and simply keeping the scores for nodes only present in one of the sets. During this combination operation the set is again pruned to remove all the nodes which falls under the scoring threshold when combined. The resulting set is the final candidate set for an index i. Intuitively these nodes needs to be connected to produce an actual graph. In the implementation this is done in real time by the following search algorithm by connecting every node in the candidate set for index i with every node on every other index j which can be reached without letting the gap penalty for moving from i to j exceed the threshold T.

2.0.10 Searching G' with the modified PO-MSA

When the candidate nodes for each position has been chosen there remains one final step: How to combine them in the best possible way. This corresponds with finding the path through G' which traversal gives an optimal score within the scoring schema. Conceptually this is in many ways similar to a regular PO-MSA search where instead of searching through the reference graph with an input string we are searching through a graph of the string with the graph as our input. Instead of giving every node a score for every index in the string we give every index of the string a score for every candidate node. This presents one problem: We need to decide the gap score for jumping between arbitrary nodes in our graph. In the regular case with strings this is easily done by taking the differences in the indexes of two elements, whereas for graphs we need to find the distance between nodes through searching for shortest paths. We do however know we are not interested in alignments which does not have a score within our given threshold, which means we can bound the search to every node which is reachable without overstepping the score boundary T.



(a) $T = 0$



(b) $T = 1$

Figure 2.3: The resulting candidate sets for mapping the string "ATA" against the reference genome from 2.1 with varying threshold T values)

The implemented search solves the problem through dynamic programming. For every index i look at every node n_x in the candidate set. For each of these nodes make pairs with every node n_y of every candidate set for the preceding T indexes j. Once again we know scores higher than T is not interesting and can bound the search backwards. For each of these pairs make a score by combining the score in the previous node with the mapping score for the character b

stored in n and s_i aswell as the gap penalty for traversing from index j to i and for traversing the graph from n_x to n_y. The score for a node is the maximal attainable score found by making these pairs. In order to store the scores we need an array of integer arrays the same size as the array of arrays of candidate nodes. To handle non-linear gap penalties we need a boolean array the same size to store gap-information. Lastly, in order to backtrack to find the path yielding the highest score we need a back-pointer array consisting of pairs of integers. The resulting space complexity is $O(5SIZEOFCANDIDATESETS)$. The average case time complexity of the search is $O(\frac{|s|T^4c^2|G|^2}{4^{|c|^2}})$ Reference supplementary.

2.0.11 Handling invalid threshold values

Chapter 3

Method

In this chapter we will present an algorithm for finding guaranteed optimal alignments for arbitrary reads against a graph-based reference genome given an error threshold. The notion of optimal will be defined in a more detailed specification of the problem alongside a rigorous description of the graphs involved. The PO-MSA algorithm briefly mentioned in the background has been used as a baseline in the experiments so an accurate description of the implementation of both this and the scoring matrix used is included.

3.0.12 Scoring schema

The algorithm is general enough to work with any scoring schema that yields a scoring function which is monotone in relation to the length of the input sequence. For these experiments a variant of the Levensthein distance schema is used (as seen in **??**), with the extension of penalizing both opening and extending a gap with -1. The impact of nonincreasing vs nondecreasing scoring functions is discussed SOMEWHERE IN THE DISCUSSION.

3.0.13 Data

The reference data used in testing is real test data borrowed from Erik Garrisons project vg**??** and the tool developed from

	A	**C**	**G**	**T**
A	0	−1	−1	−1
C	−1	0	−1	−1
G	−1	−1	0	−1
T	−1	−1	−1	0

Figure 3.1: The scoring matrix used in the experiments

the article *Canonical, Stable, General Mapping using Context Schemes*??. Read data is generated by taking random sequences from the reference graph and introducing noise. When not specified the length of the original sequences are 60 and the probabilities of introducing a SNP, an insert or a deletion is 0.01 for each.

3.0.14 PO-MSA implementation

3.0.15 Validation

In the results reads are categorized as either correctly or incorrectly aligned against the reference. This separation is made on based on whether the produced alignment has an alignment score higher than or equal to the alignment produced by PO-MSA. As PO-MSA is guaranteed to find an optimal score, results where the algorithm finds a better alignment then PO-MSA is interpreted as an error either in the implementation of PO-MSA or the scoring process.

List of Figures

Bibliography

[1] Deanna M. Church et al. 'Extending reference assembly models'. In: (2015).

[2] Alexander Dilthey et al. 'Improved genome inference in the MHC using a population reference graph'. In: (2015).

[3] Zamin Iqbal et al. 'De novo assembly and genotyping of variants using colored de Bruijn graphs'. In: (2012).

[4] Schneeberger K. et al. 'Simultaneous alignment of short reads against multiple genomes'. In: (2009).

[5] Birte Kehr et al. 'Genome alignment with graph data structures: a comparison'. In: (2014).

[6] Christopher Lee, Cathrine Grasso and Mark F. Sharlow. 'Multiple sequence alignment using partial order graphs'. In: (2001).

[7] Arthur M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, 2014.

[8] Shoshana Marcus, Hayan Lee and Michael Schatz. 'SplitMEM: Graphical pan-genome analysis with suffix skips'. In: (2014).

[9] Joong Chae Nal et al. 'Suffix Array of Alignment: A Practical Index for Similar Data'. In: (2013).

[10] Ngan Nguyen et al. 'Building a Pan-Genome Reference for a Population'. In: (2015).

[11] Adam Novak et al. 'Canonical, Stable, General Mapping using Context Schemes'. In: (2015).

[12] Benedict Paten, Adam Novak and David Haussler. 'Mapping to a Reference Genome Structure'. In: (2014).

[13] PA. Pevzner, H. Tang and MS. Waterman. 'An eulerian path approach to DNA fragment assembly'. In: (2001).