# Succinct Pattern matching on Sequence graphs

Jyotshna Rajput(19943)

April, 2022

## 1 Introduction

Much of the field of genomics revolves around the existence of 'reference' genomes, which are used as the basis for a wide range of genetic analyses, including studies of variation within and across species [1]. But scientists have realized that a single reference genome is inadequate for many purposes, specifically when it comes to **'reference bias'** as it does not provide adequate information about the genetic variations found in a diverse population [2]. This reference bias, although might look minor for us to ignore in many cases. But some important parts of our genome which is highly variable even within the same individual in the RNA and protein level and sometimes even in the DNA level due to some advanced mechanics present in human cells. Hence, finding those variations are crucial in many scientific development. The recent statement by **National Human Genome Research Institute** signifies the importance of "pan-genome," which says that one reference genome is not enough to represent all of the world's diverse populations [3].This creates the need for efficient construction and representation of pan-genome to enable all the desired operations/queries that can be performed on a single reference genome.

### 1.1 Pan-genome as a Graph

The term 'pan-genome' refers to any collection of genomic sequences to be analyzed jointly or used as a reference[2]. These sequences can be linked in a graph-like structure or simply constitute sets of (aligned or unaligned) sequences. The efficient data structures, algorithms, and statistical methods

to perform bioinformatic analyses of pan-genomes give rise to the 'computational pan-genomics.' One of the most efficient ways to represent a pan-genome is graph data structures. Still, the use of graph data structures as a reference genome (genome graph) comes up with additional challenges when it comes to the problem of pattern matching, indexing, and efficient storage pan-genome. These problems tried to be solved using different types of graph-based data structures like de-Bruijn graphs, Wheeler graphs, variation graphs, string graphs, and partial order graphs [4].

### 1.2 Problem Formulation

We want to represent a pan-genome in a graph-based data structure using less space and perform fundamental operations like matching patterns as efficiently as possible. There are number of applications of pattern matching to a pan-genome graphs ranging from knowing the characteristics of a DNA sequence, variant calling to hybrid genome assembly. Since, the low sequencing cost techniques has been developed, the pattern matching/sequence alignment to graph algorithms with efficient space complexity are going to be one of the most important building block of major computational biology applications.

**Wheeler graphs** are one such graphical representation of the pan-genome, which efficiently represents the sequences in the memory and indexes them to perform fast pattern-matching operations. Through this project, we will dig deep into the theory of Wheeler graphs and analyze the time and space complexities of such operations.

## 2 Motivation

We know that FM-index, which is a compressed full-text substring based on the BWT, can be used to efficiently find the number of occurrences of a pattern within the compressed text and locate each occurrence's position. The query time and the required space storage have a sublinear complexity with respect to the size of the input data [5]. We can generalize this concept for more than one string using graphs.

We can convert a string into a graph such that edge labels represent the characters in a string and vertices are the Burrows-wheeler of the string, i.e., nodes will be ordered according to the BW order of their outgoing edge

labels. So, matching a pattern in the graph can be interpreted in two ways. First, we are finding matching substrings in the string; second, we are finding matching paths in a graph. Like BWT follows the consecutivity property, this graph also holds the same for labels of nodes in the BW range, i.e., rows with the same prefix are consecutive in BW order. Here, Nodes can be thought of according to what comes after(outgoing edges) and or just before (incoming edges). Incoming edges spell out BWT, and Outgoing paths spell out suffixes/rotations. But this idea fails when ordering the graph for more than one string (non-linear graph). A node can have multiple suffixes leading out from them, and they might not follow the total ordering of suffixes. Hence we need some properties in such graphs to be ensured so that we can have the total ordering of the nodes without any ambiguity, and it also holds the consecuitivity.

# 3 Wheeler Graphs

Consider a directed edge-labeled graph G such that each edge is labeled by a character from an totally-ordered alphabet $A$. We use $\prec$ to denote the ordering among $A$'s elements. Labels on the edges leaving a given node are not necessarily distinct, and there can be multiple edges linking the same pair of nodes.

$G$ is a ***Wheeler graph*** if there is an ordering of the nodes such that nodes with in-degree 0 precede those with positive in-degree and, for any pair of edges $e = (u, v)$ and $e' = (u', v')$ labeled $a$ and $a'$ respectively, the following monotonicity properties hold:

- $a \prec a' \implies v < v'$,
- $(a = a') \wedge (u < u') \implies v \leq v'$. (1)

For each pair of edges: The first property says that if edges have different labels, the destination of the edge with the smaller label must come before the destination of the edge with the larger label. Consequently, a node can not have two incoming edges with different labels. Second property says that if edges have the same label but different sources, the destination of the edge from the low source must not come after the destination of the edge from the high source.

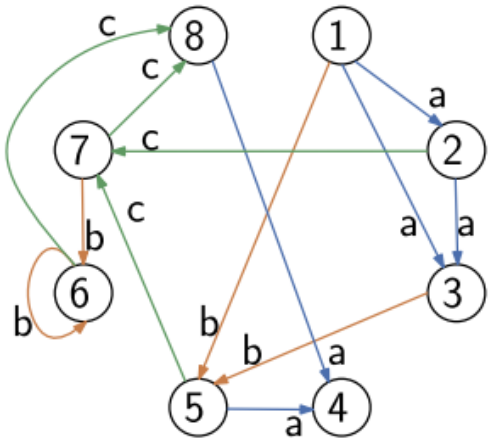


Figure 1: An eight-node Wheeler graph. Node 1 has in-degree 0; edges labeled $a$ enters in nodes 2, 3, 4; edges labeled $b$ in nodes 5, 6; edges labeled $c$ in nodes 7, 8.

$G$ is *path coherent* if there is a total order of the nodes such that for any consecutive range $[i, j]$ of nodes and string $\alpha$, the nodes reachable from those in $[i, j]$ in $|\alpha|$ steps by following edges whose labels for $\alpha$ when concatenated, themselves form a consecutive range[4]. So, If $G$ is a Wheeler graph by an ordering $\pi$ on its nodes then it is path coherent by $\pi$. This can be proved easily.

Suppose $G$ is a Wheeler graph by $\pi$. Consider a consecutive range $[i, j]$ of nodes and let $[i', j']$ be the smallest range that contains all the nodes reachable from those in $[i, j]$ in one step by following edges labeled with some character $a$. By our choice of $[i', j']$, both $i'$ and $j'$ are reachable from nodes in $[i, j]$ in one step by following edges labeled a. By our definition of a Wheeler graph, nodes with in-degree 0 precede those with positive in-degree, such as $i'$, so every node in $[i', j']$ has at least one incoming edge.

Assume some node $v$ strictly between $i$' and $j'$ has an incoming edge labeled $a' \neq a$. Since $i' < v$ we have $a \prec a'$, by our definition of a Wheeler graph and modus tollens; similarly, since $v < j'$ we have $a' \prec a$, thus obtaining a contradiction. It follows that the edges arriving at nodes in $[i', j']$ are all labeled $a$. Furthermore, since the labels are equal, by the second implication in (1) and modus tollens we get that any edge with destination strictly between $i'$ and $j'$ must originate in $[i, j]$. It follows that the nodes reachable

in one step from those in [i, j] by following edges labeled $a$ are the ones in $[i', j']$, which is a consecutive range.

## 3.1 Representation of Wheeler Graphs

This section will discuss how a Wheeler graph is represented efficiently. T.Gagie et al. use the bit-vectors to store the whole graph, and with the help of rank and select query, they perform the desired operations. Given a Wheeler graph $G$, let $x_1 < x_2 < \ldots < x_n$ denote the ordered set of nodes. For $i = 1, \ldots, n$ let $l_i$ and $k_i$ denote the out-degree and in-degree of node $x_i$ respectively.We define the binary arrays of length $e + n$.

$O = 0^{l_1}10^{l_2} \ldots 0^{l_n}1, I = 0^{k_1}10^{k_2} \ldots 0^{k_n}1$.

Note that O (resp. I ) consists of the concatenated unary representations of the out-degrees (resp. in-degrees).Let $L_i$ denote the multiset of labels on the edges exiting from $x_i$ arranged in an arbitrary order, and let $L[1..e]$ denote the concatenation $L = L_1L_2 \ldots L_n$ . By construction, $|L_i| = l_i$ and there is a natural one-to-one correspondence between the $0'$s in $O$ and the characters in $L$.

For example, in Figure 1 it is

$I = 101001001001001001001$

$O = 000100101100100100101$

$L = aabacbacbcbca$

Let $C[1 \ldots |\sigma|]$ denote the array such that $C[c]$ is the number of edges with label smaller than $c$. Given an array $Z$ we use the standard notation $rank_c(Z, i)$ to denote the number of occurrences of $c$ in $Z[1, i]$, and $select_c(Z, j)$ to denote the position of the $j-th$ $c$ in $Z$. We can easily get the out-degree and their corresponding edge labels of a node using above rank and select. The out-degree $l_i$ of node $x_i$ is $select_1(O, i) - select_1(O, i-1)$. The labels in the edges leaving $x_i$ are $L[w_i - i_i + 1, w_i]$ where $w_i = select_1(O, i) - i$.

Hence, space required to store $I$ and $O$ is $2(e + n)$bits and array $L$ takes $e \log |\sigma|$ bits. $C$ array uses $|\sigma| \log e$ bits. Also, Wavelets trees are used to perform rank and select query on $L$ which requires $o(n + e \log |\sigma|)$ and supports $O(\log |\sigma|)$ time operations.

So, it is possible to represent an $n-$node, $e-$edge Wheeler graph with labels over the alphabet $\sigma$ in $2(e + n) + e \log |\sigma| + |\sigma| \log e + o(n + e \log |\sigma|)$ ***bits.*** The representation supports the forward and backward traversing of the edges in $O(\log |\sigma|)$ ***time***.

### 3.2 Pattern matching in Wheeler graphs

Given $I, O, L$ ,and $C$, we can perform one step of matching for given character $c$ and next character $c'$ of pattern $P$ as follows:

WheelerGraphPatternMatching ($I$, $O$, $L$, $C$, $c$, $c'$)

$r_{top} \leftarrow C[c]$
$i_{top} \leftarrow I.select_0(r_{top})$
$j_{top} \leftarrow I.rank_1(i_{top})$
$k_{top} \leftarrow O.select_1(j_{top} - 1)$
$l_{top} \leftarrow O.rank_0(k_{top})$
$m_{top} \leftarrow L.rank_c(l_{top}, c')$
$r_{bot} \leftarrow C[c + 1]$
$i_{bot} \leftarrow I.select_0(r_{bot} - 1)$
$j_{bot} \leftarrow I.rank_1(i_{bot}) + 1$
$k_{bot} \leftarrow O.select_1(j_{bot} - 1)$
$l_{bot} \leftarrow O.rank_0(k_{bot})$
$m_{bot} \leftarrow L.rank_c(l_{bot}, c')$

The above algorithm performs one step of matching a pattern $P$ in $O(\log |\sigma|)$ time because there are only rank and select operations in the algorithm. Rank and select operations on bit-vectors $I$ and $O$ are $O(1)$ operations, while rank operation on array $L$ is done using wavelet trees takes $O(\log |\sigma|)$ time. Hence, for a pattern $P$, matching in the Wheeler graph can be done in $O(|P| \log |\sigma|)$ time.

# 4 Experiments

For the experiment, we built a linear Wheeler graph for a single sequence over $\sigma = \{A, C, T, G\}$ and explicitly made out-degree, in-degree bit-vectors, and edge-label array $O$, $I$ and $L$ respectively. After that, we used SDSL(Succinct data structure library) to build rank and select data structures for the above vectors and array. To perform pattern matching, we took a random string $|P|$ and gave it to the WheelerGraphPatternMatching algorithm described above.

**Github Repository:** All the data and the codes used for the above experiment can be found here, with detailed instructions to reproduce the results.

# 5 References


1. Sherman, Rachel M., and Steven L. Salzberg. "Pan-genomics in the human genome era." Nature Reviews Genetics 21.4 (2020): 243-254

2. T. C. P.-G. Consortium. Computational pan-genomics: status, promises, and challenges. Briefings in Bioinformatics, 19(1):118–135, 10 2016.

3. National Human Genome Research Institute's post: https://twitter.com/genome_gov/status/1513897531146702850?t=eqdF107PAAwrY2QHYug9Rws=08

4. Gagie, Travis, Giovanni Manzini, and Jouni Sir en. "Wheeler graphs: A framework for BWT-based data structures." Theoretical computer science 698 (2017): 67-78.

5. Hon, Wing-Kai, et al. "Practical aspects of Compressed Suffix Arrays and FM-Index in Searching DNA Sequences." ALENEX/ANALC. 2004.

6. Equi, Massimo, et al. "On the complexity of string matching for graphs." 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

7. Lecture series by Benjamin Langmead: https://youtu.be/_EgAui4_AX8