Efficient Construction of Assembly String Graphs using FM-index

Jared T. Simpson and Richard Durbin

Sudhanva Kamath

April 26, 2022

Indian Institute of Science, Bangalore

Table of contents

Introduction

Introduction

Introduction

- de Bruijn Graphs
- Overlap Graphs
    - Larger Computation Costs
- String Graphs
    - ‘Transitive’ Edges
    - Bottleneck - Compute All-Pairs Overlap

Complexity of Computing Overlaps

- Naive Method
    - $\Omega(N^2)$

Complexity of Computing Overlaps

- Naive Method
  - $\Omega(N^2)$
- Generalized Suffix Tree
  - $O(N + C^2)$ (C is number of overlaps)

Complexity of Computing Overlaps

- Naive Method
  - $\Omega(N^2)$
- Generalized Suffix Tree
  - $O(N + C^2)$ (C is number of overlaps)
- FM-Index
  - $O(N + C)$

Definitions

Strings

- A *string* X is a sequence of symbols from the alphabet Σ.
- All strings are terminated by a special \$ character, which is lexicographically smaller than all characters in Σ.
- The *length* of a read is denoted $|X|$.
- $X[i] = a_i$, for $a_i \in \Sigma, X[|X|] = \$$.
- $X[i,j]$ denotes the substring $X[i], X[i+1], \ldots, X[j]$, when $i < j$.
- The *suffix* of a string is any substring $X[k, |X|]$.
- The *prefix* of a string is any substring $X[1,k]$.
- The reverse of a string X is the string $X' = X[|X|-1], X[|X|-2], \ldots, X[1]\$$.

Genomes

- A *genome* is a long string from the alphabet $\{A, C, G, T\}$.
- A *read* is a substring of a genome.
- $\overline{X}$ is used to denote the *reverse complement* of a read X. This is also called the Watson-Crick Complement.
- $\mathcal{R}$ - an indexed collection of reads.

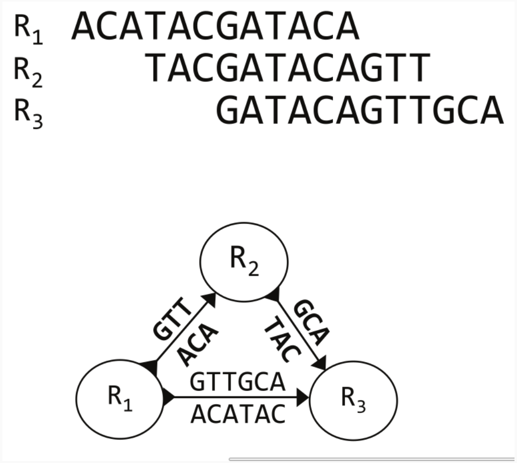
Genomes

- A *genome* is a long string from the alphabet $\{A, C, G, T\}$.
- A *read* is a substring of a genome.
- $\overline{X}$ is used to denote the *reverse complement* of a read X. This is also called the Watson-Crick Complement.
- $\mathcal{R}$ - an indexed collection of reads.
- *Overlaps* occur between a pair of reads $r_i, r_j \in \mathcal{R}$, when a suffix of r_i is equal to the prefix of r_j. Overlaps can occur between a read and the reverse complement of another read as well.
- τ - Minimum acceptable threshold for an overlap.

Overlap Graphs

- An *overlap graph* of a set of indexed reads $\mathcal{R}$ is graph where the vertex set is $\mathcal{R}$ and the edge set is the set of overlaps between pairs of reads.
- τ - Minimum acceptable threshold for an overlap.

Overlap Graphs



Source: Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010 Jun 15;26(12):i367-73. doi: 10.1093/bioinformatics/btq217. PMID: 20529929; PMCID: PMC2881401.

String Graphs

- An *string graph* can be constructed from an overlap graph by deleting some specific nodes and edges.

String Graphs

- An *string graph* can be constructed from an overlap graph by deleting some specific nodes and edges.
- Any read which is ‘contained’ in another read is not retained in the string graph.

String Graphs

- An *string graph* can be constructed from an overlap graph by deleting some specific nodes and edges.
- Any read which is ‘contained’ in another read is not retained in the string graph.
- Any edge representing a ‘transitive’ overlap is not retained in the string graph.

Read Overlaps Formalized

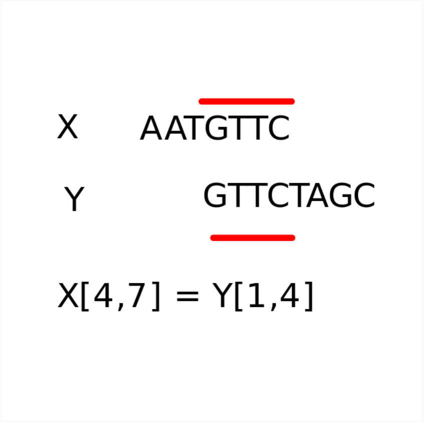
- Let X and Y be two reads. Let $X[s_{xy}, e_{xy}] = Y[s_{yx}, e_{yx}]$.
- $X[s_{xy}, e_{xy}]$ is called the ‘matched’ portion of X and the rest is called the ‘unmatched’ portion of X.
- If $s_{xy} = 1$ and $e_{xy} = |X|$, the read X fully overlaps with Y and is said to be *contained* in Y.
- If X and Y are both contained in the other, then X and Y are identical reads. Convention: The read with the higher index in $\mathcal{R}$ is contained in the read with the lower index.

Contained Read Example

- $X = AATGTGC, Y = ATGT$
  $X[2,5] = Y[1,4]$, Y is contained in X.
- $X = ATGT, Y = ACAT$
  $\overline{X[1,4]} = Y[1,4]$. X and Y are contained in each other.
  The read with the higher index is marked as a contained read.

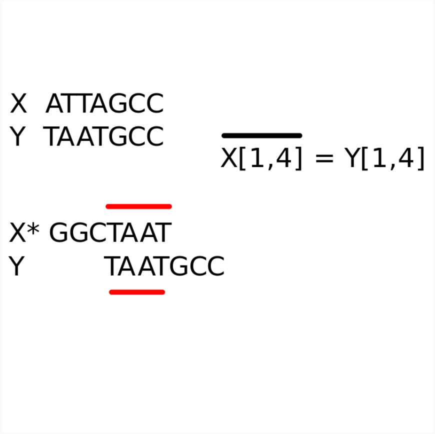
Read Overlaps Formalized

X and Y are said to have a *proper overlap*, if neither X nor Y are contained in the other, and either $X[s_{xy}, e_{xy}]$ is a prefix of X, with $s_{xy} = 1$, and $Y[s_{yx}, e_{yx}]$ is a suffix of Y, with $e_{yx} = |Y|$, or vice versa.



Read Overlaps Formalized

Reads from opposite strands of DNA they can still form an overlap. $\overline{X[s_{xy}, e_{xy}]} = Y[s_{yx}, e_{yx}]$, and both of $X[s_{xy}, e_{xy}]$ and $Y[s_{yx}, e_{yx}]$ are prefixes or suffixes.



String Graphs

- The vertex set of a string graph is the set of non-contained reads in $\mathcal{R}$.
- Each edge in a string graph is a bidirected edge which represents a proper overlap between two reads.
- $\overline{X[s_{xy}, e_{xy}]} = Y[s_{yx}, e_{yx}]$, and both of $X[s_{xy}, e_{xy}]$ and $Y[s_{yx}, e_{yx}]$ are prefixes or suffixes.
- A $4-$tuple is associated to each edge: $(type_{xy}, type_{yx}, label_{xy}, label_{yx})$.

type of an Edge

The *type* of an edge is a 2-tuple $(type_{xy}, type_{yx})$ used to indicate the type of overlap between the two reads.

type of an Edge

The *type* of an edge is a 2-tuple $(type_{xy}, type_{yx})$ used to indicate the type of overlap between the two reads.

$$type_{xy} = \begin{cases} B & \text{if } s_{xy} = 1 \\ E & \text{if } e_{xy} = |X| \end{cases}$$

$$type_{yx} = \begin{cases} B & \text{if } s_{yx} = 1 \\ E & \text{if } e_{yx} = |Y| \end{cases}$$

label **of an Edge**

The *label* of an edge are two strings $label_{xy}$ and $label_{yx}$. $label_{xy}$ is used to indicate the unmatched portion of the read Y. The concatenation of X and $label_{xy}$ is the assembly of X and Y.

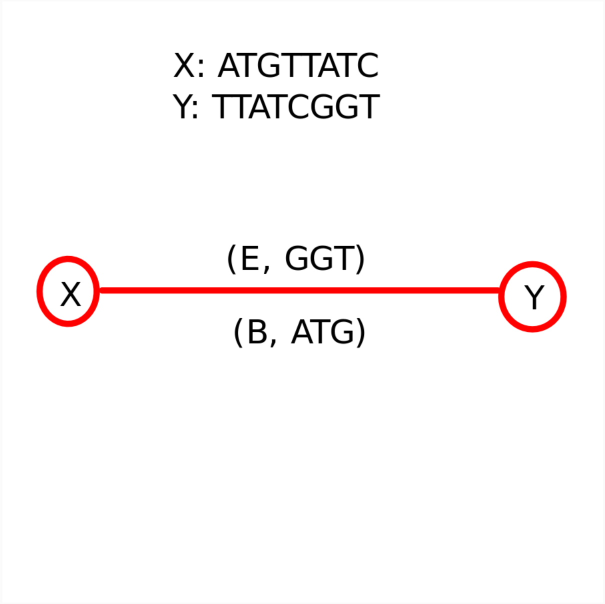
label of an Edge

The *label* of an edge are two strings $label_{xy}$ and $label_{yx}$. $label_{xy}$ is used to indicate the unmatched portion of the read Y. The concatenation of X and $label_{xy}$ is the assembly of X and Y.

$$label_{xy} = \begin{cases} Y[e_{yx}+1, |Y|] & \text{if } s_{yx} = 1 \\ Y[1, s_{yx}-1] & \text{if } e_{yx} = |Y| \end{cases}$$

$$label_{yx} = \begin{cases} X[e_{xy}+1, |X|] & \text{if } s_{xy} = 1 \\ X[1, s_{xy}-1] & \text{if } e_{xy} = |X| \end{cases}$$

Edge of type (E, B)



Reverse Complement Overlaps

If a read X overlaps has a suffix-prefix overlap with the reverse complement of Y, then

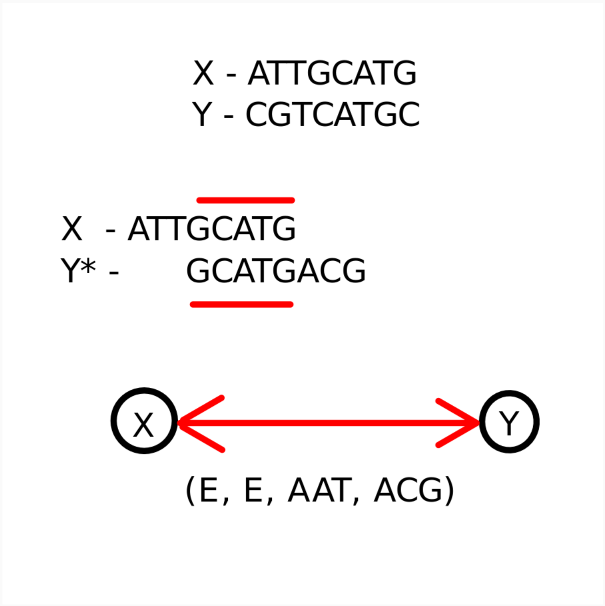
- $type_{xy} = type_{yx}$

Reverse Complement Overlaps

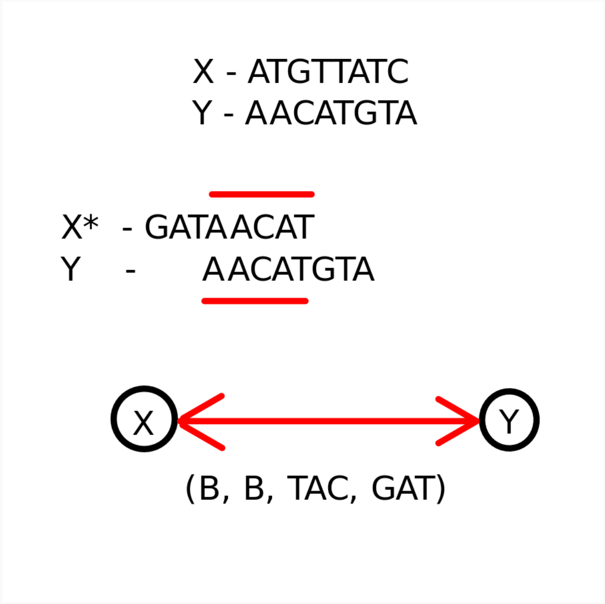
If a read X overlaps has a suffix-prefix overlap with the reverse complement of Y, then

- $type_{xy} = type_{yx}$
- $label_{xy}$ is the reverse complement of the unmatched portion of Y.
- $label_{yx}$ is the reverse complement of the unmatched portion of X.

Edge of type (E, E)



Edge of type (B, B)



Transitive and Irreducible Edges

- Suppose there are three reads, X, Y, Z such that the edges $X \leftrightarrow Y, Y \leftrightarrow Z$, and $X \leftrightarrow Z$ represent proper overlaps between the reads.
- Let $type_{xy} = type_{xz}$. Both Y and Z overlap at the same end of X. There is a substring of X which is a prefix of Y or Z but a substring of both Y and Z.

Transitive and Irreducible Edges

- Suppose there are three reads, X, Y, Z such that the edges $X \leftrightarrow Y, Y \leftrightarrow Z$, and $X \leftrightarrow Z$ represent proper overlaps between the reads.
- Let $type_{xy} = type_{xz}$. Both Y and Z overlap at the same end of X. There is a substring of X which is a prefix of Y or Z but a substring of both Y and Z.
- Let $X \rightarrow Y \rightarrow Z$ be a path in the graph, then the string corresponding to this path is an assembly of the three reads X, Y, and Z. This string is the same as the string corresponding to the path $X \rightarrow Z$. The edge $X \leftrightarrow Z$ is called a *transitive edge*.
- Edges which aren't transitive are called *irreducible edges*.

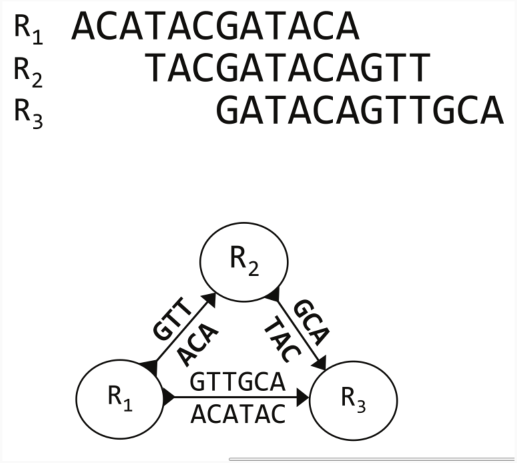
Transitive and Irreducible Edges

- Transitive edges can be safely deleted without losing any information about the assembly of the reads in the string graph.

Transitive and Irreducible Edges

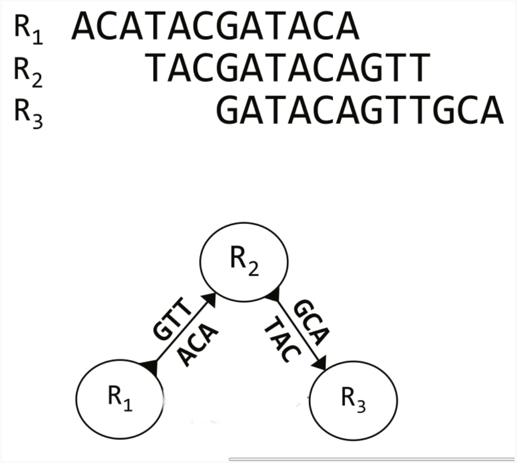
- Transitive edges can be safely deleted without losing any information about the assembly of the reads in the string graph.
- The length of the overlap between X and Y is larger than the length of the overlap between X and Z.
- $label_{xy}$ is shorter than $label_{xz}$. The label of the irreducible edge has to be shorter than the label of the transitive edge.

Overlap Graph



Source: Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010 Jun 15;26(12):i367-73. doi: 10.1093/bioinformatics/btq217. PMID: 20529929; PMCID: PMC2881401.

String Graph



Source: Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010 Jun 15;26(12):i367-73. doi: 10.1093/bioinformatics/btq217. PMID: 20529929; PMCID: PMC2881401.

String Graph Construction Pipeline

Reads$\rightarrow$Indexing$\rightarrow$Overlaps$\rightarrow$Assembly$\rightarrow$Contigs

Data Structures

Suffix Array

- The *suffix array* of a string X, denoted SA_X, is a permutation of $\{1, 2, \ldots, |X|\}$ such that $SA_X[i] = j$ if, and only if, $X[j, |X|]$ is the ith lexicographically smallest suffix of X.

Suffix Array

- The *suffix array* of a string X, denoted SA_X, is a permutation of $\{1, 2, \ldots, |X|\}$ such that $SA_X[i] = j$ if, and only if, $X[j, |X|]$ is the ith lexicographically smallest suffix of X.
- For a pattern Q, all the indices of suffixes starting with Q will occur in an interval in SA_X. Suppose $[l, u]$ denote the range of indices in which Q appears at the beginning of the suffixes of X in SA_X. Then, $[l, u]$ is called the *suffix array interval* corresponding to Q in SA_X. Using binary search, this interval can be found in $O(|Q| \log |X|)$ time.
- FM-Index enables us to do better.

Suffix Array Example

Suffix Array of CAAGTGAGTGA$

i	suffix(*i*)
12	$
11	A$
2	AAGTGAGTGA$
7	AGTGA$
3	AGTGAGTGA$
1	CAAGTGAGTGA$
10	GA$
6	GAGTGA$
8	GTGA$
4	GTGAGTGA$
9	TGA$
5	TGAGTGA$

Burrows-Wheeler Transform

The Burrows-Wheeler Transformation of a string X, denoted B_X, is a permutation of the symbols of X such that

$$B_X[i] = \begin{cases} X[SA_X[i] - 1] & \text{if } SA_X[i] > 1 \\ \$ & \text{if } SA_X[i] = 1 \end{cases}$$

BWT Example

Suffix Array and Burrows-Wheeler Transform of CAAGTGAGTGA$

Rank	i	$B[rank]$	suffix(i)
1	12	A	$
2	11	G	A$
3	2	C	AAGTGAGTGA$
4	7	G	AGTGA$
5	3	A	AGTGAGTGA$
6	1	$	CAAGTGAGTGA$
7	10	T	GA$
8	6	T	GAGTGA$
9	8	A	GTGA$
10	4	A	GTGAGTGA$
11	9	G	TGA$
12	5	G	TGAGTGA$

FM-Index

- The FM-Index is an augmentation of the BWT of a string to enable fast location of the suffix array interval of a pattern Q.
- Let $C_X(a)$ denote the number of symbols in X that are lexicographically smaller than or equal to a. Let $Occ_X(a, i)$ denote the number of occurrences of a in $B_X[1, i]$.

FM-Index

- The FM-Index is an augmentation of the BWT of a string to enable fast location of the suffix array interval of a pattern Q.
- Let $C_X(a)$ denote the number of symbols in X that are lexicographically smaller than or equal to a. Let $Occ_X(a, i)$ denote the number of occurrences of a in $B_X[1, i]$.
- If $[l, u]$ is known to be the suffix array interval of a pattern S, then the suffix array interval of aS can be determined using the following equations:

$$l = C_X(a) + Occ_X(a, l-1) \tag{1}$$

$$u = C_X(a) + Occ_X(a, u) - 1 \tag{2}$$

Using the FM-Index of CAAGTGAGTGA$

Pattern: AGTG, we have located *GTG* at $[9, 10]$.

Rank	i	$B[rank]$	suffix(i)
1	12	A	$
2	11	G	A$
3	2	C	AAGTGAGTGA$
4	7	G	AGTGA$
5	3	A	AGTGAGTGA$
6	1	$	CAAGTGAGTGA$
7	10	T	GA$
8	6	T	GAGTGA$
9	8	A	GTGA$
10	4	A	GTGAGTGA$
11	9	G	TGA$
12	5	G	TGAGTGA$

Using the FM-Index of CAAGTGAGTGA$

Pattern: AGTG, we have located *GTG* at $[9, 10]$.

Rank	i	$B[rank]$	suffix(i)
1	12	A	$
2	11	G	A$
3	2	C	AAGTGAGTGA$
4	7	G	AGTGA$
5	3	A	AGTGAGTGA$
6	1	$	CAAGTGAGTGA$
7	10	T	GA$
8	6	T	GAGTGA$
9	8	A	GTGA$
10	4	A	GTGAGTGA$
11	9	G	TGA$
12	5	G	TGAGTGA$

$l = C(A) + Occ(A, 9 - 1) = 2 + 2 = 4$
$u = C(A) + Occ(A, 10) - 1 = 2 + 4 - 1 = 5$

UpdateBackward($[l, u], a$)

```
1. $l \leftarrow C_X(a) + Occ_X(a, l-1)$
2. $u \leftarrow C_X(a) + Occ_X(a, u) - 1$
3. **return** $[l, u]$
```

UpdateBackward$([l, u], a)$

```
1. $l \leftarrow C_X(a) + Occ_X(a, l-1)$
2. $u \leftarrow C_X(a) + Occ_X(a, u) - 1$
3. **return** $[l, u]$
```

Each call to the UpdateBackward procedure takes $O(1)$ time.

To reduce space requirements for Occ_X, we can store only $Occ_X(a, i)$ for i divisible by d. The remaining values can be calculated using the Burrows Wheeler Transform.

Pattern Search: BackwardSearch(Q)

```
1. $i \leftarrow |Q|$
2. $l \leftarrow C_X(Q[i])$
3. $u \leftarrow C_X(Q[i]+1)-1$
4. $i \leftarrow i-1$
```

Pattern Search: BackwardSearch(Q)

```
1. $i \leftarrow |Q|$
2. $l \leftarrow C_X(Q[i])$
3. $u \leftarrow C_X(Q[i]+1)-1$
4. $i \leftarrow i-1$
5. **while** $l \leq u$ and $i \geq 1$ **do**
    5.1 $[l,u] \leftarrow$ UpdateBackward($[l,u], Q[i]$)
    5.2 $i \leftarrow i-1$
```

Pattern Search: BackwardSearch(Q)

```
1. $i \leftarrow |Q|$
2. $l \leftarrow C_X(Q[i])$
3. $u \leftarrow C_X(Q[i] + 1) - 1$
4. $i \leftarrow i - 1$
5. **while** $l \leq u$ and $i \geq 1$ **do**
    5.1 $[l, u] \leftarrow \text{UpdateBackward}([l, u], Q[i])$
    5.2 $i \leftarrow i - 1$
6. **return** $[l, u]$
```

Pattern Search: BackwardSearch(Q)

```
1. $i \leftarrow |Q|$
2. $l \leftarrow C_X(Q[i])$
3. $u \leftarrow C_X(Q[i] + 1) - 1$
4. $i \leftarrow i - 1$
5. **while** $l \leq u$ and $i \geq 1$ **do**
    5.1 $[l, u] \leftarrow$ UpdateBackward($[l, u], Q[i]$)
    5.2 $i \leftarrow i - 1$
6. **return** $[l, u]$
```

Time Complexity: $O(|Q|)$

Generalized Suffix Array of an Indexed set of Strings

- Let $\mathcal{T}$ be an indexed set of strings, with $|\mathcal{T}| = m$.
- Let $\$_1, \$_2, \ldots, \$_m$ be sentinel string terminator characters. $\$_i$ is a string terminator for string $\mathcal{T}_i$.
- Each string terminator is ordered by its index and are lexicographically smaller than all symbols in Σ.

Generalized Suffix Array of an Indexed set of Strings

- Let $\mathcal{T}$ be an indexed set of strings, with $|\mathcal{T}| = m$.
- Let $\$_1, \$_2, \ldots, \$_m$ be sentinel string terminator characters. $\$_i$ is a string terminator for string $\mathcal{T}_i$.
- Each string terminator is ordered by its index and are lexicographically smaller than all symbols in Σ.
- $SA_{\mathcal{T}}[i] = (j, k)$ if, and only if $\mathcal{T}_j[k, |\mathcal{T}_j|]$ is the ith lexicographically smallest suffix among all strings in $\mathcal{T}$.

BWT and FM-Index of an Indexed set of Strings

The Burrows Wheeler Transform of an indexed collection of strings $\mathcal{T}$ can be defined as follows. If $SA_{\mathcal{T}}[i] = (j, k)$,

$$B_{\mathcal{T}}[i] = \begin{cases} \mathcal{T}_j[k-1] & \text{if } k > 1 \\ \$ & \text{if } k = 1 \end{cases}$$

BWT and FM-Index of an Indexed set of Strings

The Burrows Wheeler Transform of an indexed collection of strings $\mathcal{T}$ can be defined as follows. If $SA_{\mathcal{T}}[i] = (j, k)$,

$$B_{\mathcal{T}}[i] = \begin{cases} \mathcal{T}_j[k-1] & \text{if } k > 1 \\ \$ & \text{if } k = 1 \end{cases}$$

Since $B_{\mathcal{T}}$ is a permutation of the symbols in $\mathcal{T}$, the definitions of $C_{\mathcal{T}}$ and $Occ_{\mathcal{T}}$ are unchanged and the procedures UpdateBackward and BackwardSearch are unchanged.

Constructing String Graphs

String Graph Construction Pipeline

Reads$\rightarrow$$\underline{\text{Indexing}}$$\rightarrow$$\underline{\text{Overlaps}}$$\rightarrow$Assembly$\rightarrow$Contigs

Building FM-Index of $\mathcal{R}$

- Concatenate all strings in $\mathcal{R}$ into a single string $S = R_1 R_2 \cdots R_m$.
- Compute the generalized suffix array of $S = R_1 R_2 \cdots R_m$.
- Compute the BWT and the FM-Index of S.

Building FM-Index of $\mathcal{R}$

- Concatenate all strings in $\mathcal{R}$ into a single string $S = R_1 R_2 \cdots R_m$.
- Compute the generalized suffix array of $S = R_1 R_2 \cdots R_m$.
- Compute the BWT and the FM-Index of S.
- Compute the FM-Index of $\mathcal{R}' = \{R'_1, R'_2, \ldots, R'_m\}$. This is the set of reversed strings.
- Compute the *lexicographic index* of $\mathcal{R}$ - This is a permutation of the indices of the strings in $\mathcal{R}$ which sorts the reads in $\mathcal{R}$ in lexicographic order.

Detecting All Overlaps of type (E, B)

- Let X be a read. Let P_k be the k length suffix of X.
- Perform BackwardSearch(P_k) using the FM-Index of $\mathcal{R}$ to determine the suffix array interval $[l, u]$ corresponding to P_k.
- $[l, u]$ contain reads which have a substring matching P_k. We must isolate those substrings which are also prefixes. HOW?

Detecting All Overlaps of type (E, B)

- Let X be a read. Let P_k be the k length suffix of X.
- Perform BackwardSearch(P_k) using the FM-Index of $\mathcal{R}$ to determine the suffix array interval $[l, u]$ corresponding to P_k.
- $[l, u]$ contain reads which have a substring matching P_k. We must isolate those substrings which are also prefixes. HOW?
- Which elements in this range correspond to $B_{mathcalR}[i] = \$$?
- Compute $[l_\$, u_\$]$ using UpdateBackward($[l, u], \$$).
- $[l_\$, u_\$]$ corresponds to the string $\$P_k$ in the FM-Index.
- Use the lexicographic index of $\mathcal{R}$ to find which reads are in this range.

FindOverlaps(X, τ)

```
1. $i \leftarrow |X|$
2. $l \leftarrow C_{\mathcal{R}}(X[i])$
3. $u \leftarrow C_{\mathcal{R}}(X[i] + 1) - 1$
4. $i \leftarrow i - 1$
```

FindOverlaps(X, τ)

```
1. $i \leftarrow |X|$
2. $l \leftarrow C_{\mathcal{R}}(X[i])$
3. $u \leftarrow C_{\mathcal{R}}(X[i] + 1) - 1$
4. $i \leftarrow i - 1$
5. **while** $l \leq u$ and $i \geq 1$ **do**
    5.1 **if** $|X| - i + 1 \geq \tau$
        5.1.1 $[l_\$, u_\$] \leftarrow$ UpdateBackward($[l, u], \$$)
        5.1.2 **if** $l_\$ \leq u_\$$ OutputOverlaps($X, [l_\$, u_\$]$)
    5.2 $[l, u] \leftarrow$ UpdateBackward($[l, u], X[i]$)
    5.3 $i \leftarrow i - 1$
```

FindOverlaps(X, τ)

```
1. $i \leftarrow |X|$
2. $l \leftarrow C_{\mathcal{R}}(X[i])$
3. $u \leftarrow C_{\mathcal{R}}(X[i]+1)-1$
4. $i \leftarrow i-1$
5. **while** $l \leq u$ and $i \geq 1$ **do**
    5.1 **if** $|X|-i+1 \geq \tau$
        5.1.1 $[l_\$, u_\$] \leftarrow$ UpdateBackward($[l, u], \$$)
        5.1.2 **if** $l_\$ \leq u_\$$ OutputOverlaps($X, [l_\$, u_\$]$)
    5.2 $[l, u] \leftarrow$ UpdateBackward($[l, u], X[i]$)
    5.3 $i \leftarrow i-1$
6. **if** $l \leq u$ OutputContained($X, [l, u]$)
```

Overlaps of type (E, E)

- Type (E, E) edges correspond to a suffix of X is matching a reverse complemented suffix of Y.
- Call FindOverlaps on the complement on X and use the FM-Index of $\mathcal{R}'$.

Overlaps of type (B, B)

- Type (B, B) edges correspond to a prefix of X is matching a reverse complemented prefix of Y.
- Call FindOverlaps on the reverse complement on X and use the FM-Index of $\mathcal{R}$.

Complexity of FindOverlaps

- Let c_i is the number of overlaps for read R_i. FindOverlaps makes at most $|R_i|$ calls to UpdateBackward and a total of c_i iterations in OutputOverlaps.

Complexity of FindOverlaps

- Let c_i is the number of overlaps for read R_i. FindOverlaps makes at most $|R_i|$ calls to UpdateBackward and a total of c_i iterations in OutputOverlaps.
- The time complexity FindOverlaps for one read X is $O(|X| + c_X)$.
- Total time complexity is $O(N + C)$.

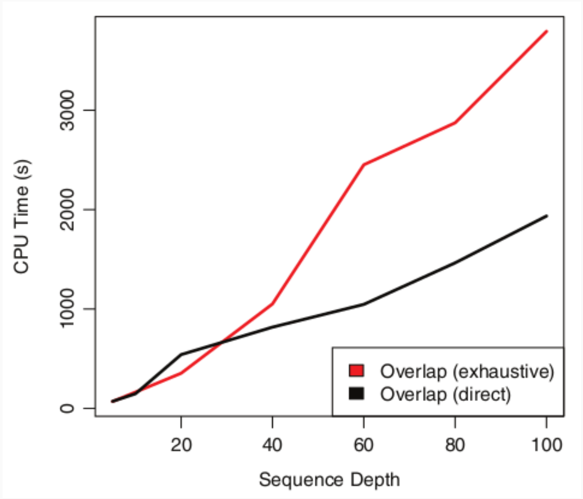
Experiments

- Indexing of Reads
- Computing Overlaps
- Assembly of String Graph, Transitive Reduction, Contigs

Experiments

- E. coli read data with mean sequence depth from 5x to 100x.
- $\tau = 27$.
- Aim: Test computational complexity as a function of sequence depth.

Experiments



Source: Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index.

References

Eugene W. Myers, The fragment assembly string graph, Bioinformatics, Volume 21, Issue suppl_2, Pages ii79-ii85, https://doi.org/10.1093/bioinformatics/bti1114

Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010 Jun 15;26(12):i367-73. doi: 10.1093/bioinformatics/btq217. PMID: 20529929; PMCID: PMC2881401.