Short (and Long) Read Alignment

Simon Rasmussen

27626: Next Generation Sequencing analysis
CBS - DTU
Generalized NGS analysis

Data size

Assembly: Alignment / de novo

Application specific: Variant calling, count matrix, ...

Compare samples / methods

Answer?
Alignment/Mapping

• Assemble your reads by aligning them to a closely related reference genome

• High sequence similarity between individuals makes this possible

Reads

Genome
Sounds easy?

- Some pitfalls:
  - Divergence between sample and reference genome
  - Repeats in the genome
  - Recombination and re-arrangements
  - Poor reference genome quality
  - Read errors
  - Regions not in the ref. genome
Alignment approaches

- Short reads: global read alignment (or glocal for little longer reads)
- Long reads: local alignments (more likely to have gap/indels)
- Exact string match
- Creating hash tables (maybe you know these from Perl/Python/Java/C, ...)
- BWT + suffix arrays
Simplest solution

- Exact string matches:
  
  Reference: ACGTGC GGACGCTGAACGTGACG
  Read: GTG GTG G-TG GTG

- We need to allow mismatches/indels (Smith-Waterman, Needleman-Wunsch)

- One of the world’s fastest computer (*K computer* - RIKEN)

- 20 mill reads 100 nt reads vs. human genome ~ 1 month

- We search each read vs. the entire reference

Complexity: $O(n*m)$
How about BLAST?

• Everybody uses BLAST
• Everybody will believe your BLAST hits (almost)

What we can learn: Reducing the search space

• However BLAST
  • finds local alignments - not always what we want for short reads
  • and other stuff (alignment scores, output format, speed)
• Not practical for short reads!
Smart solution

1. Use algorithm to quickly find *possible* matches

Drastically reduced search space

2. Allow us to perform slow/precise alignment for possible matches (Smith-Waterman)

3.2 Gb

× possible matches

1 best match
Hash based algorithms

Lookups in hashes are fast!

1. Index the reference using $k$-mers.
2. Search reads vs. hash $k$-mers
3. Perform alignment of entire read around seed
4. Report best alignment

Also known as Seed and extend

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTGCGTGTGA</td>
<td>Chr1_pos1234; Chr2_pos567</td>
</tr>
<tr>
<td>ACTGCGTGTGC</td>
<td>Chr7_posX</td>
</tr>
<tr>
<td>ACTGCGTGTGT</td>
<td>Chr7_posZ; ...</td>
</tr>
</tbody>
</table>
Spaced seeds

• Key/k-mer is called a seed

• BLAST uses \( k=11 \) and all must be matches

• Smarter: Spaced seeds (only care about ‘1’ in seed)

• Higher sensitivity

• One can use several seeds
Multiple seeds & drawbacks

- One could require multiple short seeds
  - Instead of extending around each seed, extend around positions with several seed matches (SHRiMP)

- Drawbacks of hash-based approaches:
  - Lots(!) of RAM to keep index in memory (hg ~48Gb!)
  - Poor hashing may lead to slow alignment
Burrows-Wheeler Transform

- Hash based aligners require lots of memory and are only reasonable fast
- Can we make it better/faster?
- Burrows Wheeler Transformation (BWT), Suffix Arrays and FM-index
  - BWT originally created for compression (implemented in bzip2)
The concepts

- Burrows-Wheeler Transform (BWT)
  - A reversible transformation of the genome
- Suffix Array is “array of integers giving the starting positions of suffixes of a string in lexicographical order”
- Ferragina and Manzini (FM) index
  - Allows us to recreate parts of the Suffix Array on the fly
- First create BWT of the genome (\(\text{index}\))
BWT: Create index

Genome

1. Create all possible transformations of the string
2. Sort the strings lexicographically
   (create BWT matrix and Suffix Array)

T = AGGAGC$

Marks end of string, lexicographically smallest

| 0  | AGGAGC$          | 6  | $ | A | G | G | A | G | C |
| 1  | GGAGC$A          | 3  | A | G | C | $ | A | G | G |
| 2  | GAGC$AG          | 0  | A | G | G | A | G | C | $ |
| 3  | AGC$AGG          | 5  | C | $ | A | G | G | A | G |
| 4  | GC$AGGA          | 2  | G | A | G | C | $ | A | G |
| 5  | C$AGGAG          | 4  | G | C | $ | A | G | G | A |
| 6  | $AGGAGC          | 1  | G | G | A | G | C | $ | A |
### BWT

**This one we need later**

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$ A G G A G C</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>A G C $ A G G</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>A G G A G C $</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>C $ A G G A G</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>G A G C $ A G</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>G C $ A G G A</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>G G A G C $ A</td>
<td>A</td>
</tr>
</tbody>
</table>

**SA** BWT matrix

\[ F = \text{First column} \]
\[ L = \text{Last column} \]
**BWT: T-rank**

\[ T = \text{AGGAGC}\$ \]

*T-ranking*: \# of times the base occurred previously in \( T \)

\[
\begin{align*}
A_0 & \quad G_0 & \quad G_1 & \quad A_1 & \quad G_2 & \quad C_0 & \quad $ & \\
A & \quad G & \quad C & \quad $ & \quad A & \quad G & \quad G & \\
A & \quad G & \quad G & \quad A & \quad G & \quad C & \quad $ & \\
C & \quad $ & \quad A & \quad G & \quad G & \quad A & \quad G & \\
G & \quad A & \quad G & \quad C & \quad $ & \quad A & \quad G & \\
G & \quad C & \quad $ & \quad A & \quad G & \quad G & \quad A & \\
G & \quad G & \quad A & \quad G & \quad C & \quad $ & \quad A & \\
\end{align*}
\]
**BWT: T-rank**

$T = \text{AGGAGC}\$

*T-ranking:* # of times the base occurred previously in $T$

\[
\begin{align*}
T & = \text{AGGAGC}\$
\end{align*}
\]

Notice that individual base-rank is the same in $F$ and $L$.

Rank will always be the same in $F$ and $L$.
BWT: $B$-rank

$B$-ranking: Ranked based on occurrence in $F/L$

<table>
<thead>
<tr>
<th>$F$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>$\text{C}_0$</td>
</tr>
<tr>
<td>$A_0$</td>
<td>$\text{G}_0 \text{ A}_0 \text{ G}_1$</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$\text{G}_2 \text{ G}_0 \text{ A}_0 \text{ G}_1 \text{ C}_0$</td>
</tr>
<tr>
<td>$C_0$</td>
<td>$$ $\text{A}_1 \text{ G}_2 \text{ G}_0 \text{ A}_0$</td>
</tr>
<tr>
<td>$G_0$</td>
<td>$\text{A}_0 \text{ G}_1 \text{ C}_0 $ $\text{A}_1$</td>
</tr>
<tr>
<td>$G_1$</td>
<td>$\text{C}_0 $ $\text{A}_1 \text{ G}_2 \text{ G}_0$</td>
</tr>
<tr>
<td>$G_2$</td>
<td>$\text{G}_0 \text{ A}_0 \text{ G}_1 \text{ C}_0 $</td>
</tr>
</tbody>
</table>

$T = \text{AGGAGC}\$ |

$B$-rank: $\text{A}_1 \text{ G}_2 \text{ G}_0 \text{ A}_0 \text{ G}_1 \text{ C}_0 \$ |

$T$-rank: $\text{A}_0 \text{ G}_0 \text{ G}_1 \text{ A}_1 \text{ G}_2 \text{ C}_0 \$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 G_0 G_2 A_1 \]

Reversed:
\[ A_1 G_2 G_0 A_0 G_1 C_0 \]
\[ T = \text{AGGAGGC}$ \]

\( F \) can be computed = 2\( x \) A, 1\( x \) C, 3\( x \) G

We therefore only need to store \( L \)
We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A G G

Find all rows that start with “G”
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A G G

Find all rows that starts with “G”

Simple because $F$ is sorted
## BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

**Read** = “AGG”

Start from last base:

A \textcolor{red}{G} \textcolor{red}{G}

In this range, match second last base in $L$
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:

A  G  G

In this range, match second last base in $L$
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A  G  G

Use LF mapping to get to F
**BWT: Lookups**

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base: 

\[ \text{AGG} \]

Match next base (A) in L
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:

\[
\text{AGG}
\]

Luckily there was a match - else no alignment
We can look up where a read matches in our genome.

Read = “AGG”

Start from last base:
A G G

Use LF mapping to get to F
We can look up where a read matches in our genome

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>C</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>C</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>$</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>$</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>A</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>A</td>
<td>G</td>
<td>C</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A   G   G

This is our match!
BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>A_1 \ G_2 \ G_0 \ A_0 \ G_1 \ C_0</td>
</tr>
<tr>
<td>A_0</td>
<td>G_1 \ C_0 \ $ \ A_1 \ G_2 \ G_0</td>
</tr>
<tr>
<td>A_1 \ G_2 \ G_0</td>
<td>A_0 \ G_1 \ C_0 \ $</td>
</tr>
<tr>
<td>C_0 \ $</td>
<td>A_1 \ G_2 \ G_0 \ A_0 \ G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0 \ G_1 \ C_0 \ $ \ A_1 \ G_2</td>
</tr>
<tr>
<td>G_1 \ C_0 \ $</td>
<td>A_1 \ G_2 \ G_0 \ A_0</td>
</tr>
<tr>
<td>G_2 \ G_0 \ A_0 \ G_1 \ C_0 \ $</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A G G

This is our match!

But where is this in our genome?
# BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>SA</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$ A_1 G_2 G_0 A_0 G_1 C_0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A_0 G_1 C_0 $ A_1 G_2 G_0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>A_1 G_2 G_0 A_0 G_1 C_0 $</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C_0 $ A_1 G_2 G_0 A_0 G_1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>G_0 A_0 G_1 C_0 $ A_1 G_2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>G_1 C_0 $ A_1 G_2 G_0 A_0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>G_2 G_0 A_0 G_1 C_0 $ A_1</td>
<td></td>
</tr>
</tbody>
</table>

Read = “AGG”
BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>SA</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$</td>
<td>A₁ G₂ G₀ A₀ G₁ C₀</td>
</tr>
<tr>
<td>3</td>
<td>A₀ G₁ C₀ $</td>
<td>A₁ G₂ G₀</td>
</tr>
<tr>
<td>0</td>
<td>A₁ G₂ G₀</td>
<td>A₀ G₁ C₀ $</td>
</tr>
<tr>
<td>5</td>
<td>C₀ $</td>
<td>A₁ G₂ G₀ A₀ G₁</td>
</tr>
<tr>
<td>2</td>
<td>G₀ A₀ G₁ C₀ $</td>
<td>A₁ G₂</td>
</tr>
<tr>
<td>4</td>
<td>G₁ C₀ $</td>
<td>A₁ G₂ G₀ A₀</td>
</tr>
<tr>
<td>1</td>
<td>G₂ G₀ A₀ G₁ C₀ $</td>
<td>A₁</td>
</tr>
</tbody>
</table>

Read = “AGG”

Our read aligns at pos 0

T = AGGAGC$

pos 0
BWT for alignment

- Entire SA is 12Gb for human genome
- FM-index
  - We only store certain parts of the array
  - We can calculate missing parts on the fly
- Human genome can be effectively indexed and searched using 3Gb RAM!
Implementation in BWA

- Burrows Wheeler Aligner (BWA) can use:
  - `bwa aln`: First ~30nt of read as seed
    - Extend around positions with seed match
  - `bwa mem`: Multiple short seeds across the read
    - Extend around positions with several seed matches
Genome: GTAC$

All possible transformations

Lexicographically sorted

The BWT is: __________
Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors

Unmapped read can be “rescued” by a good aligning mate
Coverage

- Coverage/depth is how many times that your data covers the genome (on average)

- Example:
  - $N$: Number of reads: 5 mill
  - $L$: Read length: 100
  - $G$: Genome size: 5 Mbases
  - $C = \frac{5 \times 100}{5} = 100X$
  - On average there are 100 reads covering each position in the genome

\[
C = N \times \frac{L}{G}
\]
Actual depth

- We aligned reads to the genome - how much do we actually cover?
- Avg. depth ~ 90X
- Range from 0-250X
- Only 50% of the genome was covered with reads
SAM/BAM format

- Sequence Alignment / Map format
- BAM = Binary SAM and zipped - *always* convert to BAM
- Two sections
  - Header: All lines start with “@”
  - Alignments: All other lines