Introduction to Read-Based Alignment

Michael C. Zody, Ph.D.
Workshop on Genomics
Cesky Krumlov
January 8, 2020
Aligning to a Reference

• Aligning sequences is a classic problem
  – Early bioinformatic problem
  – Very similar to older text matching problems

• Several algorithms exist
  – Tradeoffs of speed versus accuracy, sensitivity

• Sequencing throughput creates new problems
  – Short reads have less information than long seqs
  – Data volume requires faster processing per read
Example of alignment

Read:

TCAACTCTGCCAACACCTTCTCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAATGGCCAAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAATTACCATTGATGAGAAAAAAACATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGTTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCCTCCTCCAGGAAGCACTCCTGGGATTTCCCTCTTGGCCAACAAGATTCTGGAGGGCAGCTCCTCCA
ACATGCCCCAAACAGCTCTCTGCAGACATATCACATATCACATATCTTCCATACCATAACTGCCCATGCCCATACA
Example of alignment

Read:

TCAACTCTGCCAACACCTTCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAAATGGCAAAAATTAACATAGAAGGTGAGTAGAAACTTTAAATAATTACCATTGATGAGAAAGAAAAAATCATGCCACTGAAAAAGGCACCCGGTCCAGAGGTTTCTATGAGCGGGAAGTGGGTAGAAACTGTAGAAACCTTTCGAAT
TCAACTCTGC
CAACACCTTCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA
ACATGCCCCAAACAGCTCTCTGCAAGACATATCATATCATATCATATCATATCTTCCCATACATAACTGCATGCCATACAA
How Would You Find That?

- Brute force comparison
- Smith-Waterman
- Suffix Tree
- Burrows-Wheeler Transform
- Hashing/Minimizers
Brute Force Method

TCGATCC

?  
GACCTCATTGATCCCCACTG
Brute Force Method

TCGATCC
\[\times\]
GACCTCATA\textcolor{red}{TCGATCC}C\textcolor{red}{C}CACTG
TCGATCC

\[ \times \]

GACCTCATA\textcolor{red}{TCGATCC}C\textcolor{red}{CACTG}
Brute Force Method

TCGATCC

GACCTCATCGATCCCACTG
Brute Force Method

GACCTCA

TCGATCC

TCGATCC

TCGATCC

TCGATCC

TCGATCC

TCGATCC

GACCTCA

TCGATCC

TCGATCC
**Smith-Waterman**

Simplistic Scoring Scheme:
- +1 match if moving diagonally
- -1 mismatch if moving diagonally
- -1 gap if moving hor. or vert.

(no penalty for terminal gaps)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

^ G A C C T C A T C G A T C C C C A C T G
Suffix Tree

GACCTCA

TCGATCC

CACTG

G

G

TG

CTG

ACTG

CACTG

A

C

G

T

G

C

A

T

G

T

G

G
Suffix Tree

GACCTCA\textcolor{red}{T}CGATCC\textcolor{red}{C}CACTG
Burrows-Wheeler Transform

```
GACCTCA TCGATCCCACCTG$
ACCTCA TCGATCCCACCTG$
CCTCA TCGATCCCACCTG$GA
CTCATCGATCCCACCTG$GAC
TCATCGATCCCACCTG$GAC
CATCGATCCCACCTG$GACCT
ATCGATCCCACCTG$GACCTC
TCGATCCCACCTG$GACCTCA
CGATCCCACCTG$GACCTCAT
GATCCCACCTG$GACCTCATC
ATCCACCTG$GACCTCATCG
TCCACCTG$GACCTCATCGA
CCCACCTG$GACCTCATCGAT
CCACTG$GACCTCATCGATC
GACTG$GACCTCATCGATCC
ACTG$GACCTCATCGATCCC
CTG$GACCTCATCGATCCC
TG$GACCTCATCGATCCCC
TG$GACCTCATCGATCCCCA
G$GACCTCATCGATCCCAC
$GACCTCATCGATCCCACCT
```

```
ACCTCA TCGATCCCACCTG$G
ACTG$GACCTCA TCGATCCC
ATCCACCTG$GACCTCATCG
ATCGATCCCACCTG$GACCTC
CCACTG$GACCTCATCGAT
GATCCACCTG$GACCTCATC
ATCGATCCCACCTG$GACCTCA
CTG$GACCTCATCGATCC
TCCACCTG$GACCTCATCGA
CCCACCTG$GACCTCATCGAT
CCACTG$GACCTCATCGATC
GACTG$GACCTCATCGATCC
ACTG$GACCTCATCGATCCC
CTG$GACCTCATCGATCCC
TG$GACCTCATCGATCCCC
TG$GACCTCATCGATCCCCA
G$GACCTCATCGATCCCAC
$GACCTCATCGATCCCACCT
```
How Do We Use This To Align?

- Start with the transform column
- My read starts with a T, so I want rows with Ts in them
- This column gives me all the single nucleotide counts
- Sort the single nucleotide counts to get the alphabetically first column
- Now these two columns give me all the dinucleotide counts
- Sort those to get the alphabetically first two columns
- Now there is only one place my read can match
FM Index

- Start with the transform column
- Count all the characters, sort them, and store the count of lower characters

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

- This gives the positions of all the bases in the first column (because it’s sorted)
Take the query sequence TCGATCC

Start at the end and use the count table to look up the position of the last base in the first column

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>$</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>

The last column comes immediately before the first column

Find all the rows of the last column with the next to last base
• Take the query sequence TCGATCC

• The order of a given character in the last column matches the order of the same instance of that character in the first column

• The 3\textsuperscript{rd}-5\textsuperscript{th} Cs in the last column precede Cs in the first column, so we now want the 3\textsuperscript{rd}-5\textsuperscript{th} Cs in column 1

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>

• Now we take the next character and look for Ts in the last column (the 2\textsuperscript{nd} T)
FM Index

- Take the query sequence TCGATCC
- The second T is preceded by the 3rd A

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
**FM Index**

- Take the query sequence **TCGATCC**
- The third A is preceded by the $2^{nd}$ G

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>
• Take the query sequence TCGATCC
• The second G is preceded by the 6th C

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
• Take the query sequence TCGATCC
• The sixth C is preceded by the 3rd T

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
To find the position in the genome, we keep a separate index of positions for a sparse set of rows in the table and then just walk through the transform to the nearest indexed row.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
”Hashing” by Visual Example

Read:

TCAACTCTGCCAACAACCTTCTCCTCCAGGAAGCACTCCTGGATTCCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGTTTCATGAGCGGAAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTTCTCTCCACGGAAGCACTCCTGGAATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAAGCTCCTCCA
ACATGCCCCAAACAGCTCTGCAAGCATACTCATATCATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA
"Hashing" by Visual Example

Read:

TCAACTCTGCAACACCTCTCCCTCCAGGAAGCATCCTGGATTTCCCTCTTGCCAACAAGATTCTGGAGGGCAGCTCCTCCA

Genome:

ATAAATGGCACAATTTAAACTGAAGGTTGAGTAGAAACTTAAATAAACTAATTACCATTGAGAACTAAACTAAACTAA
TGCCACTGAAGAACAGCGGCTCCAGAGGGTTTCTCAGAGGGAACCTGATAGGACATTTCCCAATTCAACTCTGC
CAACACCTCCCTCCCTCAAGGAAGCCTCCTGGAATTTCCTTGGCCAACAGATTCTGAGGAGGACAGCTCCTCCA
ACATGCCCAACAGCTCTCTGCAGCATATCATATCATATCATATATCATATCTTCCATACCATAACTGCCATGCCATA
”Hashing” by Visual Example

Read:

TCAACTCTGCCAACACCTTCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGG

Genome:

ATAAAATGGCCAAAAATTAACTAGAAGGTGAGTGAAGAATTAAATAAACTAATTACCATTGATGAGAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTTGCCAACAGAGATTCTGGGAGG

ACATGCCCCAACAGCTCTGAGACATATCATATCATATCATATATCCATACCATAACTGCCATGCCATA
"Hashing" Explained

- Walk the reference and build a list of words of length \( k \) (k-mers) with their positions in the sequence
  - Exhaustive method is every k-mer
  - Can do non-overlapping, partially overlapping, etc.
  - The more k-mers you store, and the smaller \( k \) is, the more sensitive the method will be
  - The fewer k-mers you store, and the larger \( k \) is, the more efficient it will be

- To align, find all the k-mers in each read and look for them in the index (or “hash”) and find their locations, then use a modified Smith-Waterman to extend and score the match
“Seeding”

- Hashing is a way to seed, but not the only way
- One can use suffix trees or bwts to seed (in fact many aligners do this); however, it is only efficient if a single seed can be extended to most of the alignment cheaply
- For a while, there was a great deal of effort expended to develop better and more efficient seeding methods
Minimizers

• A minimizer (Roberts et al., 2004) is one efficient way to seed

• Minimizers are generated as follows:
  – Slide a window of size $w$ across the genome
  – For every position starting in $w$, determine the k-mer that starts at $w$
  – By some deterministic method, sort the k-mers in $w$
  – The lowest sort order k-mer is $w$ is the minimizer of $w$

• Any sequence containing a window $w$ identical to the window will produce the same minimizer, making it irrelevant to store other k-mers to match those regions

• By tuning $k$ and $w$, you can adjust sensitivity and efficiency
Minimizer Example

- Minimizers in a toy example with $k = 3$ and $w = 3$
- For all $w \leq k$, it is guaranteed every position will be covered by at least one minimizer.
- Although compression is small ($7/14$) in this toy example, it scales as compression ratio $= 2 / (w+1)$
Seed-Chain-Extend

- For long, noisy (or diverged) data, going straight from seeding to base pair resolution alignments may be inefficient.
- Instead, we can form an optimal chain of seeds.
- This uses a dynamic programming scheme similar to Smith-Waterman, but optimizes on minimum gap size.
- If our sequences are highly similar and our minimizers are dense, we may have the complete alignment from overlapping chained minimizers.
- Otherwise, we can add an extend step where we use a true Smith-Waterman global alignment between each adjacent pair of non-overlapping minimizers.
Common Short Read Aligners

- Seed and Smith-Waterman extend
  - Novoalign
- BWA align gap-free
  - Bowtie
- BWA align with gaps
  - BWA aln, Bowtie2
- BWA Seed and Smith-Waterman extend
  - BWA mem
- Seed-chain-extend
  - STAR, Blasr, minimap2