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:

TCAACTCTGCCAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAGCTCCTCCA
ACATGCCCCCAACAGCTCTCTGCAGACATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA

Example of alignment

Read:

TCAACTCTGCCAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

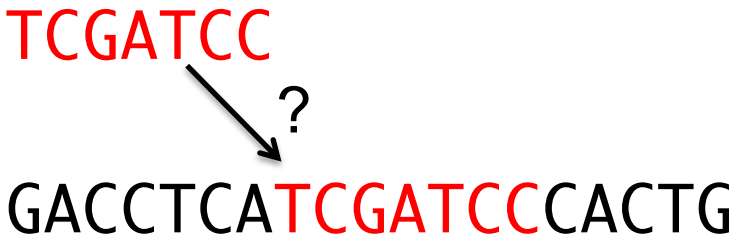
Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAGCTCCTCCA
ACATGCCCCCAACAGCTCTCTGCAGACATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA

How Would You Find That?

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

Brute Force Method



Brute Force Method

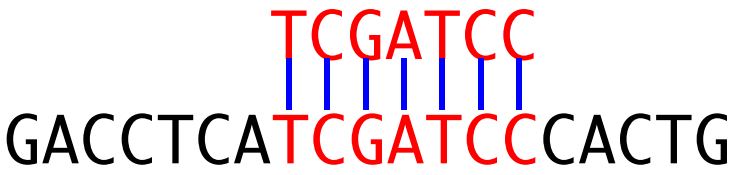
TCGATCC
x
GACCTCATCGATCCCACTG

Brute Force Method

```
  TCGATCC
  x
GACCTCATCGATCCCACTG
```

Brute Force Method

Brute Force Method



Smith-Waterman

Simplistic Scoring Scheme:

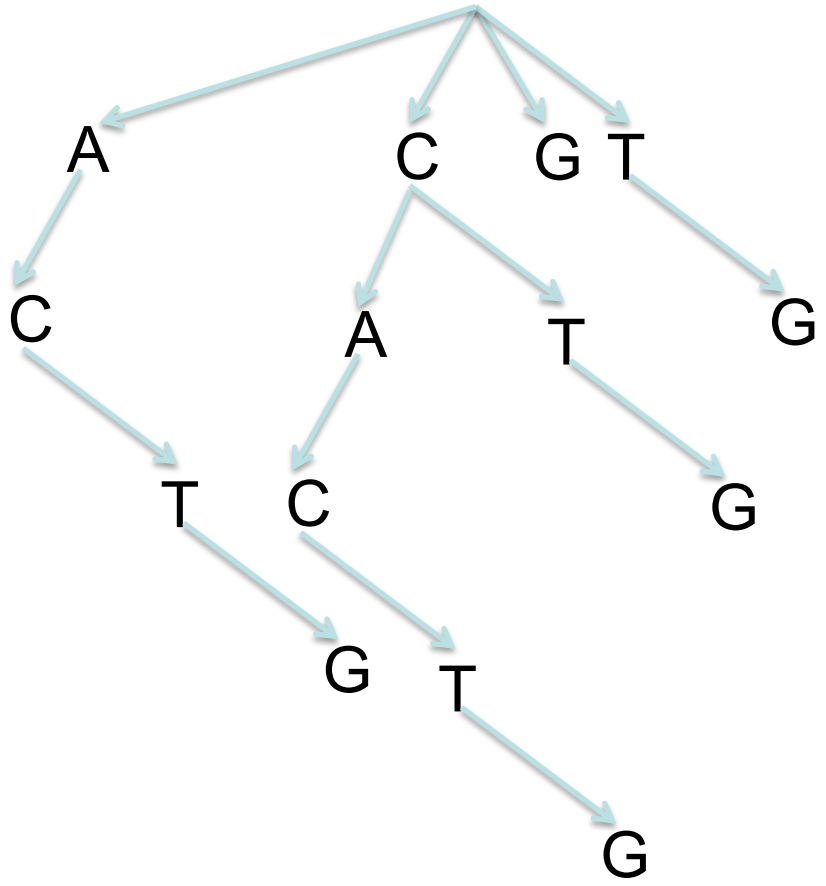
+1 match	if moving diagonally	take the
-1 mismatch	if moving diagonally	highest of
-1 gap	if moving hor. or vert.	these

(no penalty for terminal gaps)

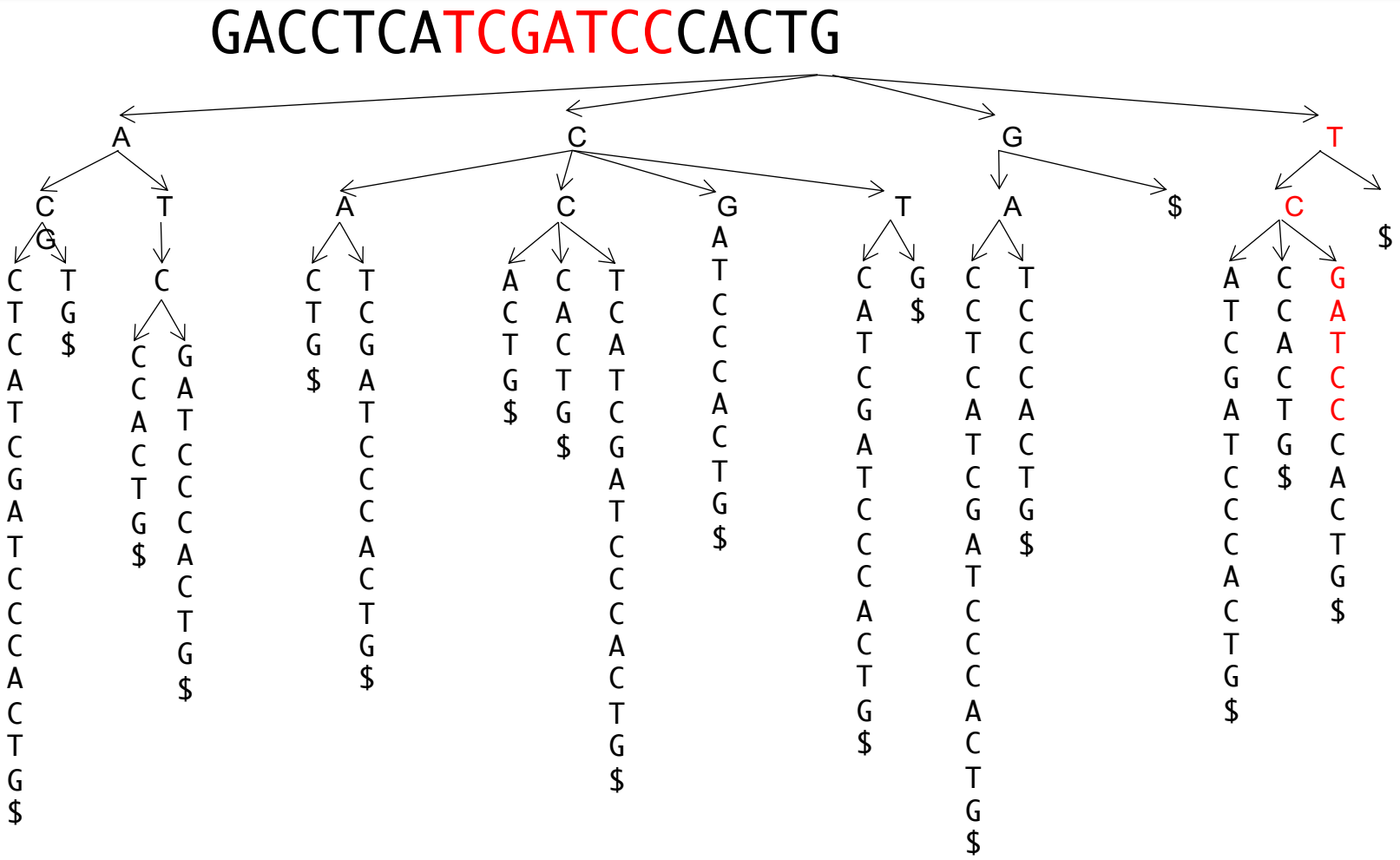
	0	-1	-2	0	2	1	1	0	1	3	3	2	3	5	7	6	5	4	3	2
	0	-1	-1	1	1	0	1	0	2	4	3	2	4	6	5	4	3	2	1	0
	0	-1	0	0	-1	0	-1	-1	3	2	1	1	5	4	3	2	1	0	1	0
A	0	0	1	0	-1	-2	0	2	1	0	2	4	3	2	1	0	1	0	-1	-2
G	0	1	0	-1	-2	-1	1	1	0	1	3	2	1	1	1	0	-1	-2	-1	0
C	0	-1	0	-1	0	0	2	1	0	2	1	0	0	2	1	0	-1	0	0	0
T	0	-1	-1	-1	-1	1	0	-1	1	0	-1	-1	1	0	-1	-1	-1	-1	1	0
^	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	^	G	A	C	C	T	C	A	T	C	G	A	T	C	C	C	A	C	T	G

Suffix Tree

GACCTCATCGATCCCACTG
G
TG
CTG
ACTG
CACTG



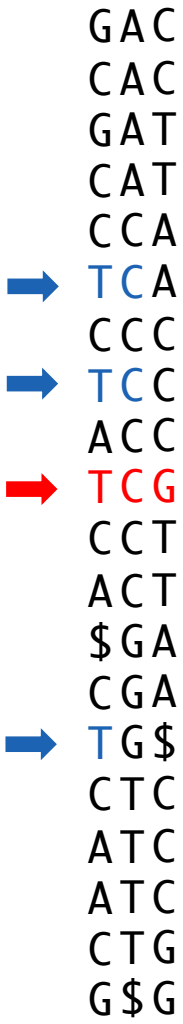
Suffix Tree


GACCTCATCGATCCCACTG
A
C
G
T
C
G
T
C
A
C
G
T
$
C
T
C
$
CTCATCGATCCCACTG$
TG$
CCACTG$
GATCCCACTG$
CTG$
TCGATCCCACTG$
ACTG$
CACTG$
TCATCGATCCCACTG$
GATCCCACTG$
CATCGATCCCACTG$
G$
CCTCATCGATCCCACTG$
TCCCACTG$
ATCGATCCCACTG$
CCACTG$
GATCCCACTG$

Burrows-Wheeler Transform

GACCTCATCGATCCCACTG$	ACCTCATCGATCCCACTG$G
ACCTCATCGATCCCACTG$G	ACTG$GACCTCATCGATCCC
CCTCATCGATCCCACTG$GA	ATCCCACTG$GACCTCATCG
CTCATCGATCCCACTG$GAC	ATCGATCCCACTG$GACCTC
TCATCGATCCCACTG$GACC	CACTG$GACCTCATCGATCC
CATCGATCCCACTG$GACCT	CATCGATCCCACTG$GACCT
ATCGATCCCACTG$GACCTC	CCACTG$GACCTCATCGATC
TCGATCCCACTG$GACCTCA	CCCACTG$GACCTCATCGAT
CGATCCCACTG$GACCTCAT	CCTCATCGATCCCACTG$GA
GATCCCACTG$GACCTCATC	CGATCCCACTG$GACCTCAT
ATCCCACTG$GACCTCATCG	CTCATCGATCCCACTG$GAC
TCCCACTG$GACCTCATCGA	CTG$GACCTCATCGATCCCA
CCCACTG$GACCTCATCGAT	GACCTCATCGATCCCACTG$
CCACTG$GACCTCATCGATC	GATCCCACTG$GACCTCATC
CACTG$GACCTCATCGATCC	G$GACCTCATCGATCCCACT
ACTG$GACCTCATCGATCCC	TCATCGATCCCACTG$GACC
CTG$GACCTCATCGATCCCA	TCCCACTG$GACCTCATCGA
TG$GACCTCATCGATCCCAC	TCGATCCCACTG$GACCTCA
G$GACCTCATCGATCCCACT	TG$GACCTCATCGATCCCAC
$GACCTCATCGATCCCACTG	$GACCTCATCGATCCCACTG

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

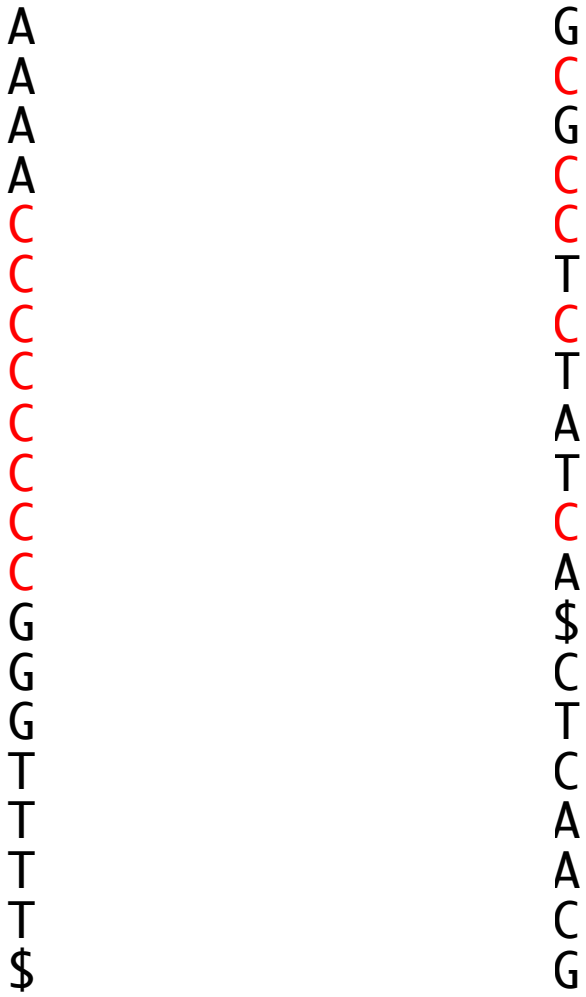
GCGCCTCTATCA$CTCAACG

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

A	C	G	T	$
0	4	12	15	19

- This gives the positions of all the bases in the first column (because it's sorted)

FM Index



- 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*

A	C	G	T	$
0	4	12	15	19

- The last column comes immediately before the first column
- Find all the rows of the last column with the next to last base

FM Index

AC	G
AC	C
AT	G
AT	C
CA	C
CA	T
CC	C
CC	T
CC	A
CG	T
CT	C
CT	A
GA	$
GA	C
G$	T
TC	C
TC	A
TC	A
TG	C
$G	G

- 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 3rd-5th Cs in the last column precede Cs in the first column, so we now want the 3rd-5th Cs in column 1

A	C	G	T	$
0	4	12	15	19

- Now we take the next character and look for Ts in the last column (the 2nd T)

FM Index

ACC	G
ACT	C
ATC	G
ATC	C
CAC	C
CAT	T
CCA	C
CCC	T
CCT	A
CGA	T
CTC	C
CTG	A
GAC	$
GAT	C
G$G	T
TCA	C
TCC	A
TCG	A
TG$	C
$GA	G

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

A	C	G	T	$
0	4	12	15	19

FM Index

ACCT	G
ACTG	C
ATCC	G
ATCG	C
CACT	C
CATC	T
CCAC	C
CCCA	T
CCTC	A
CGAT	T
CTCA	C
CTG$	A
GACC	$
GATC	C
G$GA	T
TCAT	C
TCCC	A
TCGA	A
TG$G	C
$GAC	G

- Take the query sequence TCGATCC
- The third A is preceded by the 2nd G

A	C	G	T	$
0	4	12	15	19

FM Index

ACCTC	G
ACTG$	C
ATCCC	G
ATCGA	C
CACTG	C
CATCG	T
CCACT	C
CCCAC	T
CCTCA	A
CGATC	T
CTCAT	C
CTG$G	A
GACCT	$
GATCC	C
G$GAC	T
TCATC	C
TCCCA	A
TCGAT	A
TG$GA	C
$GACC	G

- Take the query sequence TCGATCC
- The second G is preceded by the 6th C

A	C	G	T	$
0	4	12	15	19

FM Index

ACCTCA	G
ACTG$G	C
ATCCCA	G
ATCGAT	C
CACTG$	C
CATCGA	T
CCACTG	C
CCCACT	T
CCTCAT	A
CGATCC	T
CTCATC	C
CTG$GA	A
GACCTC	$
GATCCC	C
G$GACC	T
TCATCG	C
TCCCAC	A
TCGATC	A
TG$GAC	C
$GACCT	G

- Take the query sequence TCGATCC
- The sixth C is preceded by the 3rd T

A	C	G	T	$
0	4	12	15	19

FM Index

ACCTCAT	G
ACTG$GA	C
ATCCCAC	G
ATCGATC	C
CACTG$G	C
CATCGAT	T
CCACTG$	C
CCCACTG	T
CCTCATC	A
CGATCCC	T
CTCATCG	C
CTG$GAC	A
GACCTCA	$
GATCCCA	C
G$GACCT	T
TCATCGA	C
TCCCACT	A
TCGATCC	A
TG$GACC	C
$GACCTC	G

- Take the query sequence TCGATCC
- And we're done

A	C	G	T	$
0	4	12	15	19

- 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

”Hashing” by Visual Example

Read:

TCAACTCTGCCAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAGCTCCTCCA
ACATGCCCCCAACAGCTCTCTGCAGACATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA

”Hashing” by Visual Example

Read:

TCAACTCTGCCAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAGCTCCTCCA
ACATGCCCCCAACAGCTCTCTGCAGACATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA

”Hashing” by Visual Example

Read:

TCAACTCTGCCAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCA

Genome:

ATAAAATGGCCAAAATTAACTAGAAGGTGAGTAGAAACTTAAATAAACTAATTACCATTGATGAGAAAAAAAATC
TGCCACTGAAAAAGGCACCCGGTCCAGAGGGTTTCATGAGCGGGAACTGTAGAAACCTTTCGAATTCAACTCTGC
CAACACCTTCCTCCTCCAGGAAGCACTCCTGGATTTCCCTCTTGCCAACAAGATTCTGGGAGGGCAGCTCCTCCA
ACATGCCCCCAACAGCTCTCTGCAGACATATCATATCATATCATATCTTCCATACCATAACTGCCATGCCATACA

”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

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	2	3	1	0	3	2	1	0	1	2	3	3	1	0	1
	2	3	**1**	**0**	**3**										
		3	1	**0**	**3**	**2**									
			1	**0**	**3**	**2**	1								
				0	**3**	**2**	1	0							
					3	2	**1**	**0**	**1**						
						2	1	**0**	**1**	**2**					
							1	**0**	**1**	**2**	3				
								0	**1**	**2**	3	3			
									1	**2**	**3**	3	1		
										2	**3**	**3**	1	0	
											3	3	**1**	**0**	**1**

- Minimizers in a toy example with k = 3 and w= 3
- For all w ≤ k, it is guarantee 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