Mapping Reads

Dr. Hubert Rehrauer

Sequence Alignment

- Optimization problem: Find the alignment with the highest score
- Global alignment (end-to-end): Needleman-Wunsch
- Local alignment: Smith-Waterman

Score matrix:

	a	c	g	t
a	+2	-2	-1	-2
c	-2	+2	-2	-1
g	-1	-2	+2	-2
t	-2	-1	-2	+2

Gap score: -3

```
c  c  t  a  g  t  t  c  a  t
      |  |  x  |  x  |  |
t  g  t  a  a  t  -  c  a  c
      +2 +2 -1 +2 -3 +2 +2
```

Alignment score = 6

Local alignments (Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap** (typically, mismatch penalties are set according to evolutionary knowledge)

Match: +1; Mismatch: -1; Gap: -2

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0					
T	0					
G	0					
A	0					
C	0					

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + w(-,b_j) & \boxed{\text{Insertion}} \end{cases}, \quad 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0	1	1			
T	0					
G	0					
A	0					
C	0					

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + w(-,b_j) & \boxed{\text{Insertion}} \end{cases}, \quad 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0	1	1	0	0	0
T	0	0	0	2	0	1
G	0					
A	0					
C	0					

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + w(-,b_j) & \boxed{\text{Insertion}} \end{cases}, \quad 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0	1	1	0	0	0
T	0	0	0	2	0	1
G	0	0	0	0	3	1
A	0	1	1	0	1	2
C	0	0	0	0	0	0

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + w(-,b_j) & \boxed{\text{Insertion}} \end{cases}, \quad 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Traceback:

Start with maximum score (here, 3)

Follow path that gives multiple score

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0	1	1	0	0	0
T	0	0	0	2	0	1
G	0	0	0	0	3	1
A	0	1	1	0	1	2
C	0	0	0	0	0	0

$$H(i,j) = \max \begin{Bmatrix} 0 & \\ H(i-1,j-1) + \ w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + \ w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + \ w(-,b_j) & \boxed{\text{Insertion}} \end{Bmatrix}, \ 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Traceback:

Start with maximum score (here, 3)

Follow path that gives multiple score

This gives:

```
AATGT-
-ATGAC
```

	.	A	A	T	G	T
.	0	0	0	0	0	0
A	0	1	1	0	0	0
T	0	0	0	2	0	1
G	0	0	0	0	3	1
A	0	1	1	0	1	2
C	0	0	0	0	0	0

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \boxed{\text{Match/Mismatch}} \\ H(i-1,j) + w(a_i,-) & \boxed{\text{Deletion}} \\ H(i,j-1) + w(-,b_j) & \boxed{\text{Insertion}} \end{cases}, \quad 1 \le i \le m, 1 \le j \le n$$

http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Local alignments (Smith-Waterman)

Other considerations:

Natural extension is to have a different *gap opening* and a *gap extension* penalty (former generally being larger)

GO penalty=-2; GE penalty=-1; Mismatch=-1; Match=1

With above penalties, which is the best scoring alignment?

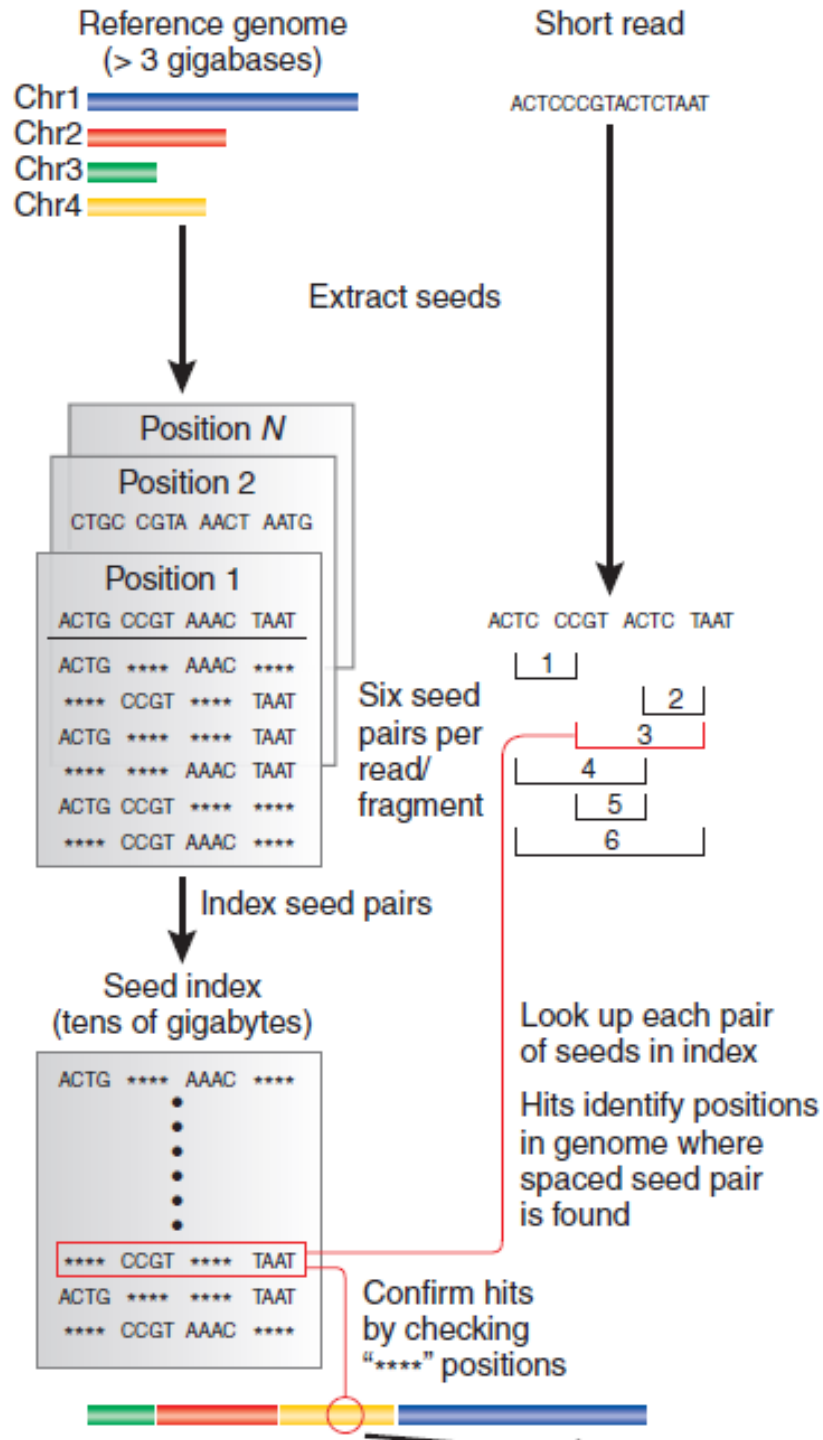
```
AT-C-GT        ATC--GT        AT-C--GT
ATTTTGT        ATTTTGT        ATT-TTGT
```

Alignment Tools

- Dynamic programming is slow
- Speed-up with heuristics
  - e.g. exactly align short subsequences and extend these alignments
    - e.g. BLAST / BLAT
  - no longer guaranteed to find the best alignments
  - exact matches are found by index lookup

Index Genome: Spaced Seeds

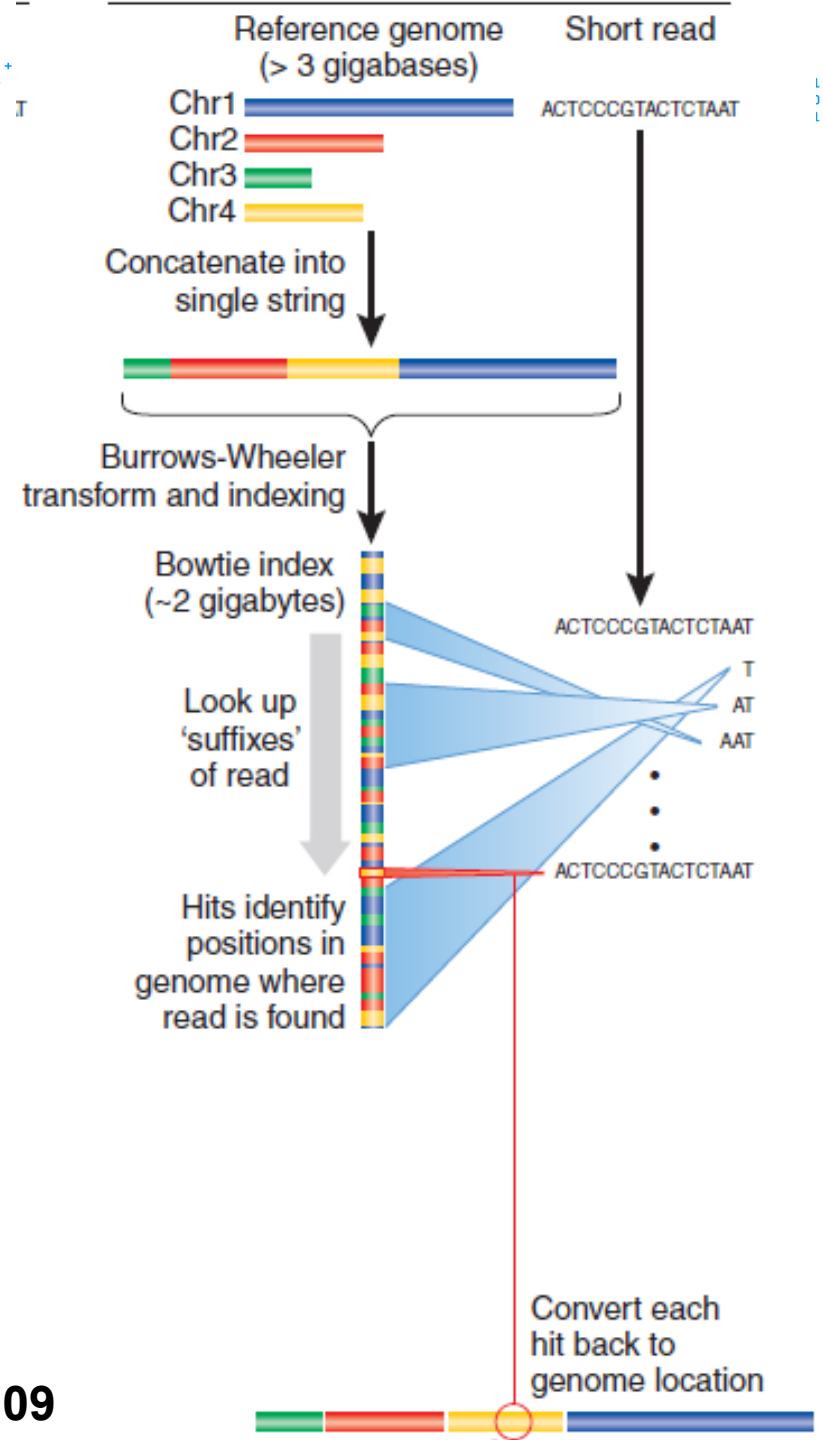
- Tags and tag-sized pieces of reference are cut into small "seeds."
- Pairs of spaced seeds are stored in an index.
- Look up spaced seeds for each tag.
- For each "hit," confirm the remaining positions.



Trapnell etal., Nature Biotechnology, 2009

Index Genome: Burrows-Wheeler Transform

- Store entire reference genome.
- Align tag base by base from the end.
- When tag is traversed, all active locations are reported.
- If no match is found, then back up and try a substitution.



Trapnell etal., Nature Biotechnology, 2009

The Burrows-Wheeler transform (1994; 1983)

c a c a a c g $

c	a	c	a	a	c	g	$
a	c	a	a	c	g	$	c
c	a	a	c	g	$	c	a
a	a	c	g	$	c	a	c
a	c	g	$	c	a	c	a
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c
$	c	a	c	a	a	c	g

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

g c c a a $ a c

The “Last-First mapping” property

The relative ordering of a particular character (say c) in column 1 is the same as that in the last column

c_1 a c_2 a a c_3 g $

$	c_1	a	c_2	a	a	c_3	g
a	a	c_3	g	$	c_1	a	c_2
a	c_2	a	a	c_3	g	$	c_1
a	c_3	g	$	c_1	a	c_2	a
c_2	a	a	c_3	g	$	c_1	a
c_1	a	c_2	a	a	c_3	g	$
c_3	g	$	c_1	a	c_2	a	a
g	$	c_1	a	c_2	a	a	c_3

The “Last-First mapping” property

The relative ordering of a particular character (say c) in column 1 is the same as that in the last column

c_1 a c_2 a a c_3 g $

$	c_1	a	c_2	a	a	c_3	g
a	a	c_3	g	$	c_1	a	c_2
a	c_2	a	a	c_3	g	$	c_1
a	c_3	g	$	c_1	a	c_2	a
c_2	a	a	c_3	g	$	c_1	a
c_1	a	c_2	a	a	c_3	g	$
c_3	g	$	c_1	a	c_2	a	a
g	$	c_1	a	c_2	a	a	c_3

Proof:

Suppose c X and c Y are cyclic permutations of the input T.
Suppose c X < c Y (in lexicographical ordering)
Then X c < Y c (in lexicographical ordering)
The LF-mapping property follows.

BWT is reversible

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

g c c a a $ a c → $ a a a c c c g

BWT is reversible

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

g$ ca ca aa ac $c ac cg

$c aa ac ac ca ca cg g$

BWT is reversible

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Lookup AAC

$	c	a	c	a	a	c	g
a	a	c	g	$	c	a	c
a	c	a	a	c	g	$	c
a	c	g	$	c	a	c	a
c	a	a	c	g	$	c	a
c	a	c	a	a	c	g	$
c	g	$	c	a	c	a	a
g	$	c	a	c	a	a	c

Range ← range of last character in 1st column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range ← LF-mapping of first and last match

Comparison

Spaced seeds

- Requires ~50Gb of memory.
- Runs 30-fold slower.
- More straightforward to program.
- Examples:
  - MAQ
  - Shrimp
- More tolerant to
  - sequence variations
  - sequencing errors

Burrows-Wheeler

- Requires <2Gb of memory.
- Runs 30-fold faster.
- More complicated to program.
- Examples:
  - bowtie
  - BWA
  - tophat (uses bowtie)
  - STAR

Alignment with Mismatches

- Mismatches can occur because of
  - sequencing error (error rate ~1/500)
  - mutation; (human mutation rate ~1/1e4)
  - → if reads are long (> 100nt) reads with mismatches will not be rare; more than ~10% of the reads may have a mismatch
- If there is a sequence mismatch the index lookup fails! How to find nevertheless an alignment?

BWT Alignment with Mismatches

- Strategies:
  - If the BWT lookup fails at position k, try all different bases at position k
    → drawback: computing effort grows exponentially with the number of mismatches
    → implemented e.g. in bowtie
  - Chop reads in segments (seeds) and align those mismatch-free and stitch seed alignments together
    → implemented e.g. in bowtie

Sequencing Quality: PHRED Scores

- PHRED is an accurate base-caller used for capillary traces (Ewing et al Genome Research 1998)
- Each called base is given a quality score Q
- Quality based on simple metrics (such as peak spacing) calibrated against a database of hand-edited data
- Q = 10 * log10(estimated probability that call is wrong)

  10 prob = 0.1

  20 prob = 0.01

  30 prob = 0.001

  [Q30 often used as a threshold for useful sequence data]

→ down-weight low-quality bases when computing the alignment score

Multiple Alignments

- A read may have multiple valid alignments with identical or similarly good alignment scores
- Aligners allow to choose different reporting strategies:
  - Randomly select one alignment from the top-scoring alignments
  - Report all alignments that are within *delta* of the top-scoring alignment; clip if more than *Nmax* alignments are found
  - Report only alignments if they are unique (no other alignment within *delta* of the alignment score)
  - Do not report anything if more than *Nmax* valid alignments are found
  - …
- Whether a read has a **unique alignment** depends on
  - the read sequence and the sequence homology of the organism
  - the search algorithm of the aligner
  - the reporting options strategy of the aligner

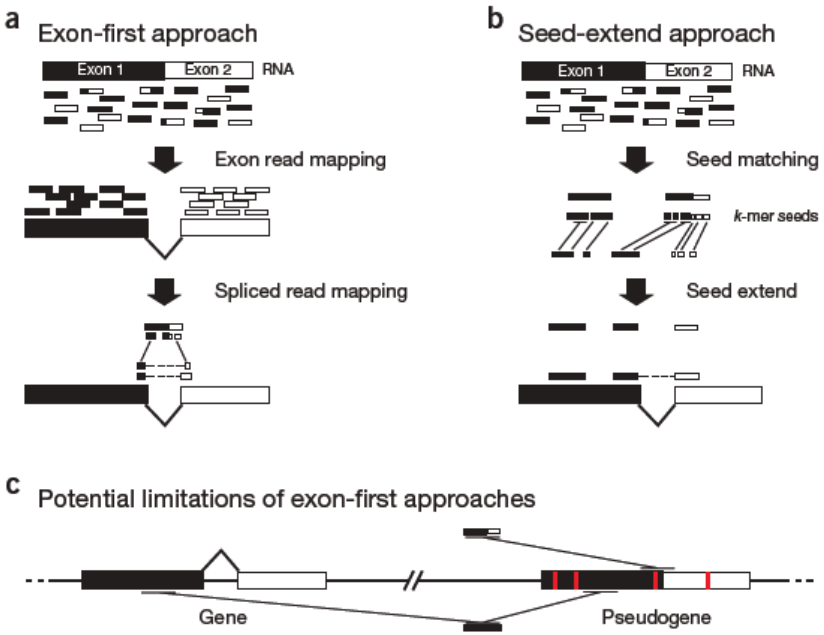
Low Quality Read Ends

- Read quality drops with the length of the reads
  → aligners clip ends if they do not align
  → **local alignment**

RNA-seq Mapping

- Mapping targets:
  - transcriptome
  - splice junction library
  - genome

- Mouse retina 60+60bp reads:
  - 41 of 91 Mio map to junctions

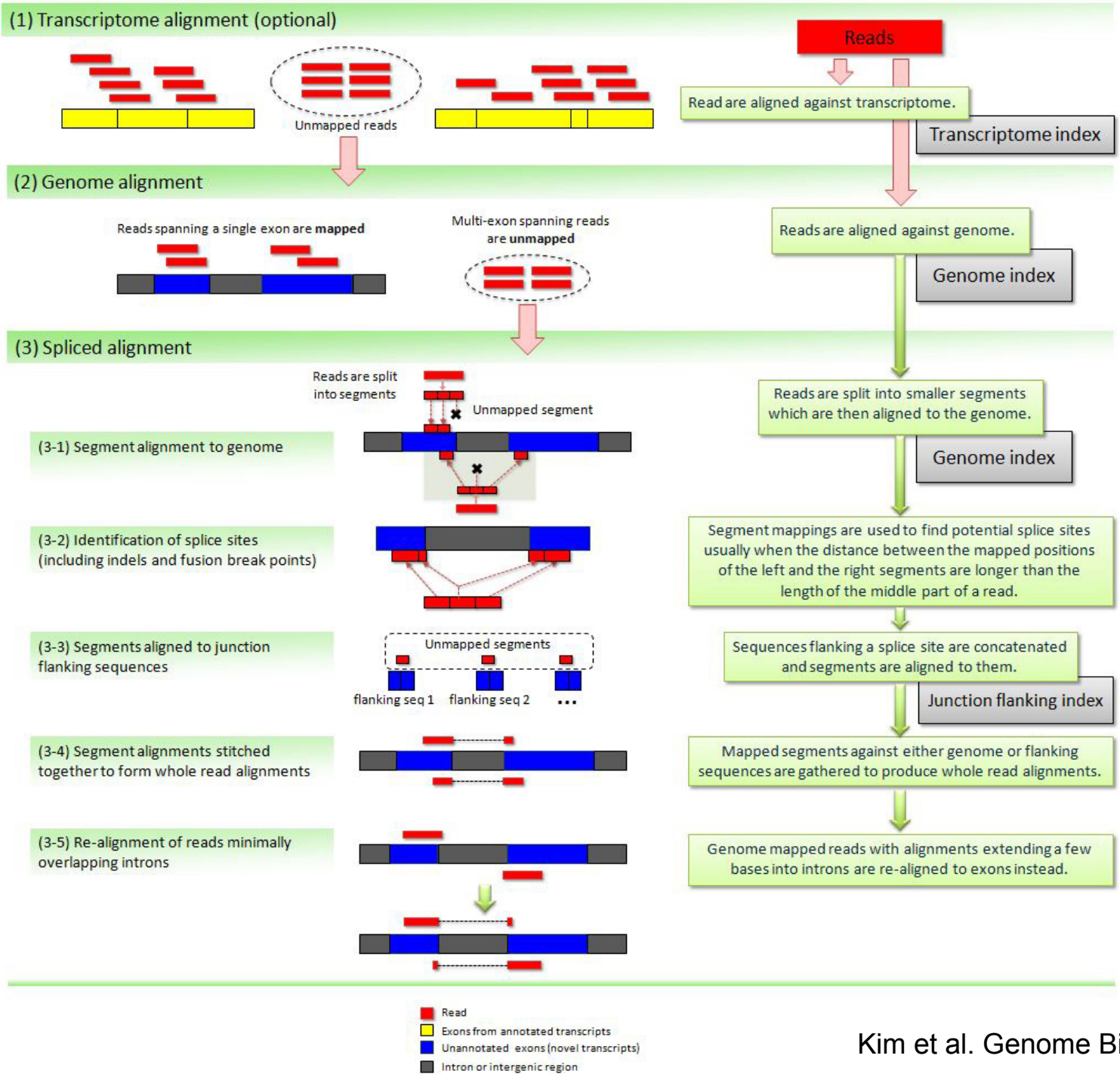


Strategies:

- Exon-first: fast
- Seed-extend:
  - good with polymorphisms
  - simultaneous spliced/ unspliced mapping

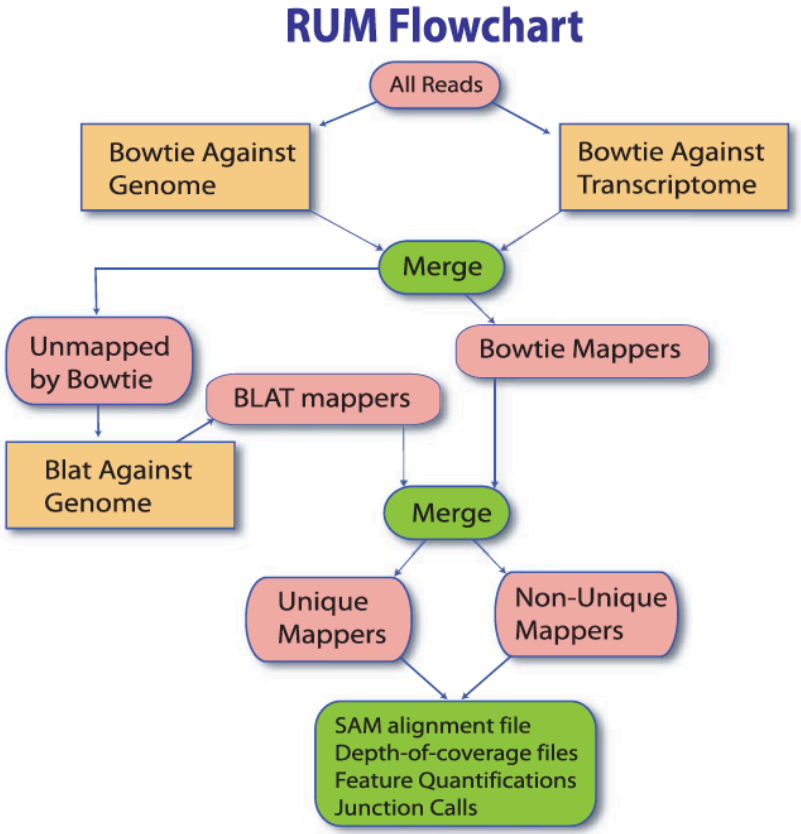
Garber etal., Nature Methods, 2011

Tophat2 pipeline



Kim et al. Genome Biology 2013

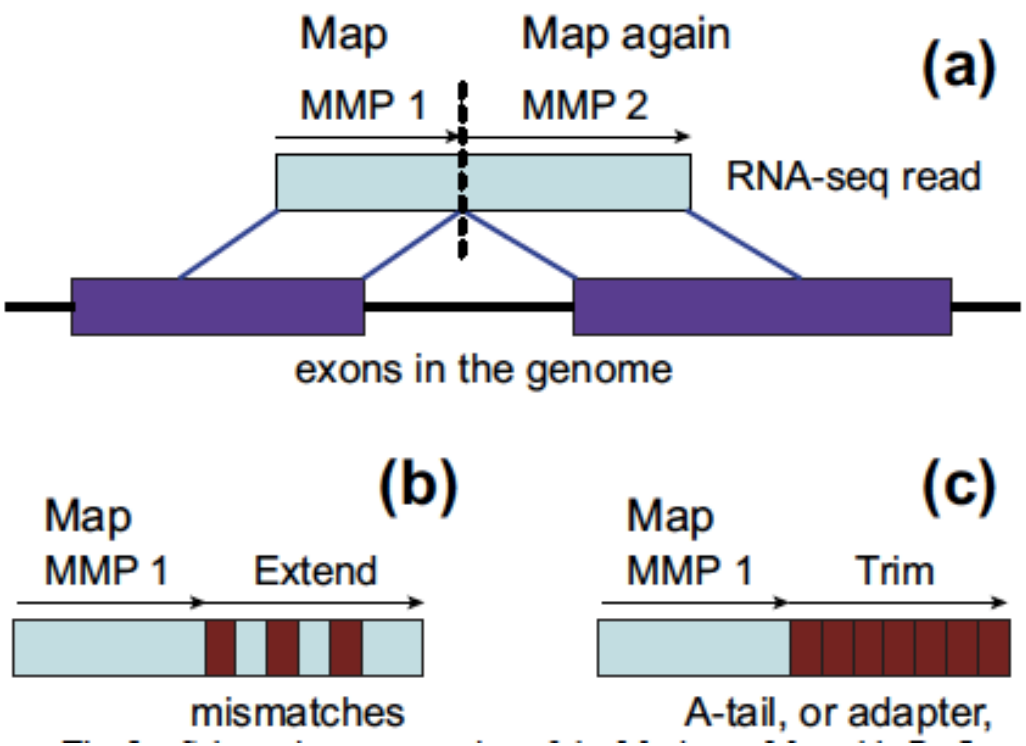
RUM: RNA-seq Unified Mapper



Grant etal., Bioinformatics, 2011

STAR: universal RNA-seq aligner

- Designed to align the non-contiguous sequences directly to the reference genome
- Steps:
  - Search Maximal Mappable Prefix (MMP)
  - clustering/stitching/scoring



Read Alignment Summary

How to map billions of reads?

- The major aligners use the Burrows-Wheeler-Transform to create an index of the reference. Even for the human genome the index fits into 3GB RAM.
- Reads are aligned by **index lookup** not by sequence comparison
- The lookup of a perfect match read is faster than loading the read and writing the alignment coordinates to the output file!
- If the lookup fails because of SNPs or sequencing errors, an actual sequence comparison is done –and this can take time!