Bowtie

Langmead, Trapnell, Pop, Salzberg 2009

CS 4390/5390
Fall 2019

Software

Open Access

Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

Ben Langmead, Cole Trapnell, Mihai Pop and Steven L Salzberg

Address: Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA.

Correspondence: Ben Langmead. Email: langmead@cs.umd.edu

Published: 4 March 2009

Genome ***Biology*** 2009, **10:**R25 (doi:10.1186/gb-2009-10-3-r25)

The electronic version of this article is the complete one and can be found online at http://genomebiology.com/2009/10/3/R25

Received: 21 October 2008
Revised: 19 December 2008
Accepted: 4 March 2009

FM-Index

The "BWT Index" discussed previously is also called the "FM Index"

- Originally defined by Ferragina and Manzini in 2000/2005

Reminder that the BWT/FM-index is:

- A data structure for a sting *T* containing
  - $BWT_{T\$}$ encoded as a wavelet tree
  - an integer array *C* continuing the counts of each character from Σ in T

Refresher on BWTs

Can be constructed using the last character of the lexicographic order of all cyclic rotations of the text

Refresher on BWTs

Can be constructed using the last character of the lexicographic order of all cyclic rotations of the text

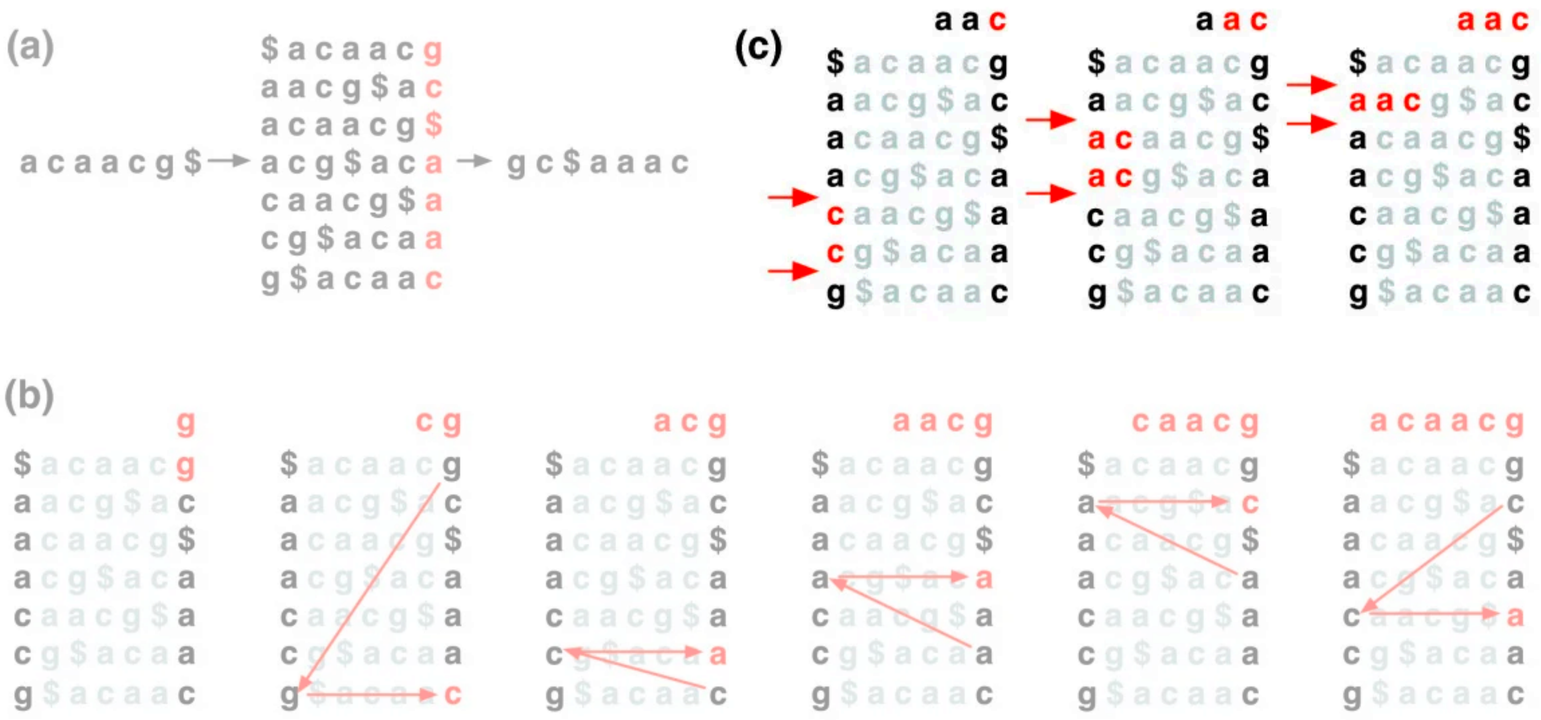
Encodes the original text, which can be recovered by a walk in the sequence

Refresher on BWTs

Can be constructed using the last character of the lexicographic order of all cyclic rotations of the text

Encodes the original text, which can be recovered by a walk in the sequence

Searching for patterns is done back to front using similar techniques to sequence recovery



Using a BWT to Align Reads

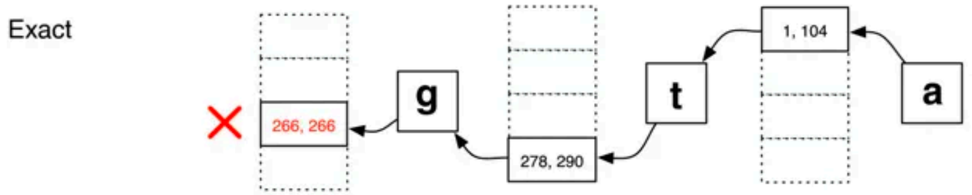
The BWT and FM-Index are insufficient for aligning reads since it doesn't allow for errors

Previously mentioned some method to overcome this

Bowtie assumes all changes are single point changes (i.e. mismatches only)

- They use a backtracking search to find matching locations
- The quality scores are used to prioritize alignments
- Other speed-ups are included to ensure all matching locations are found

Backtracking



Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made

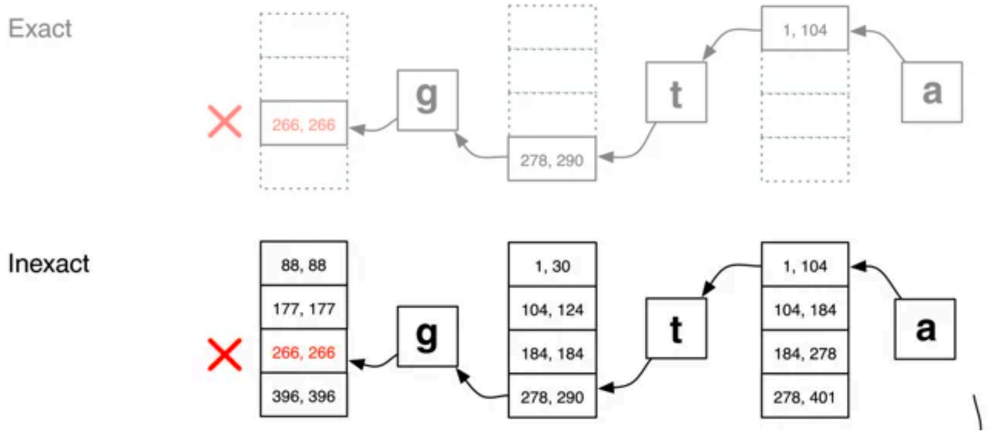
Backtracking

Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made



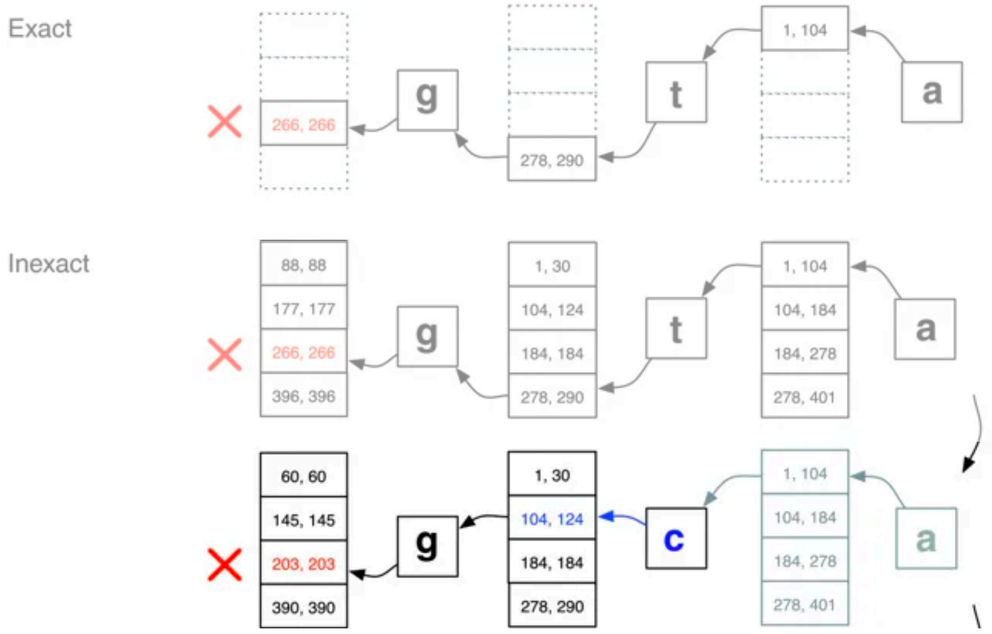
Backtracking

Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made



Backtracking

Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made

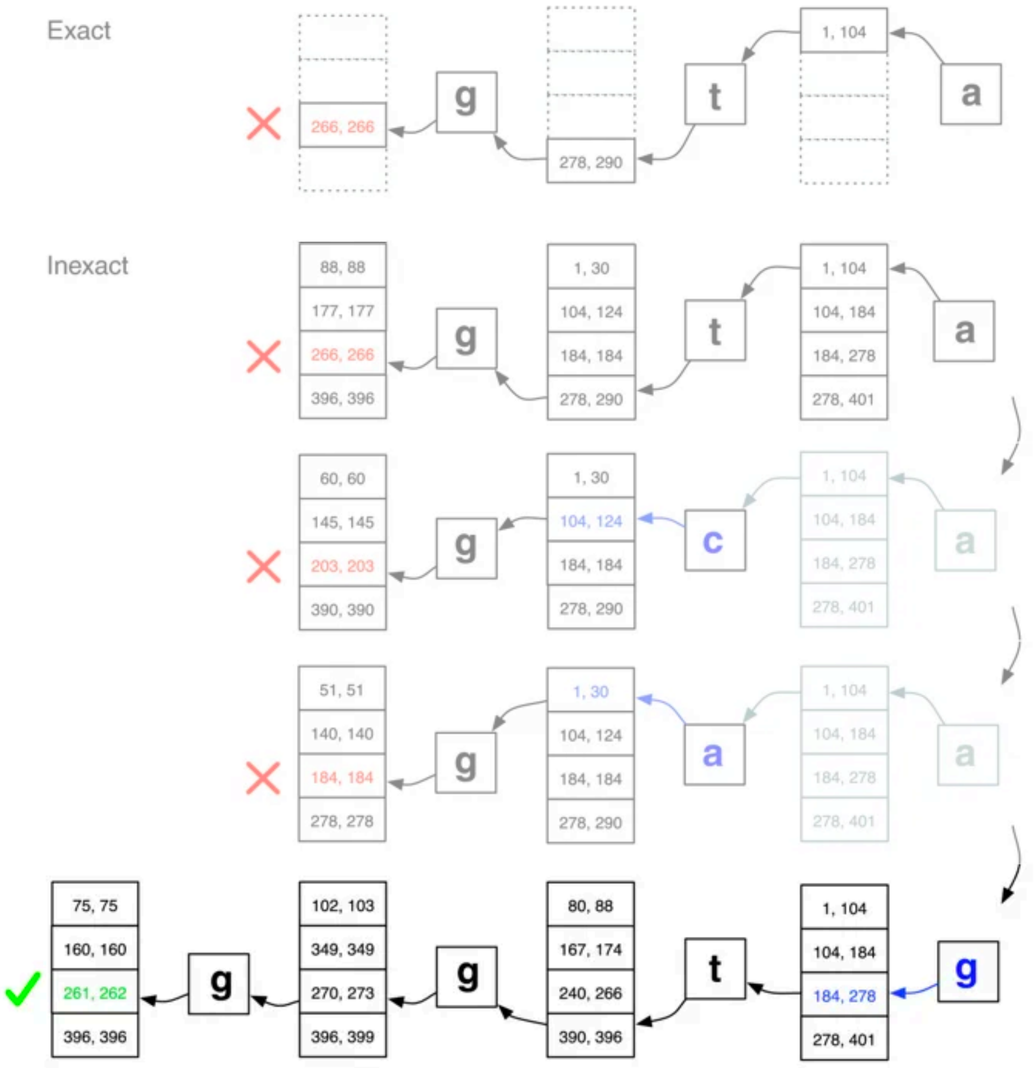
Backtracking

Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made



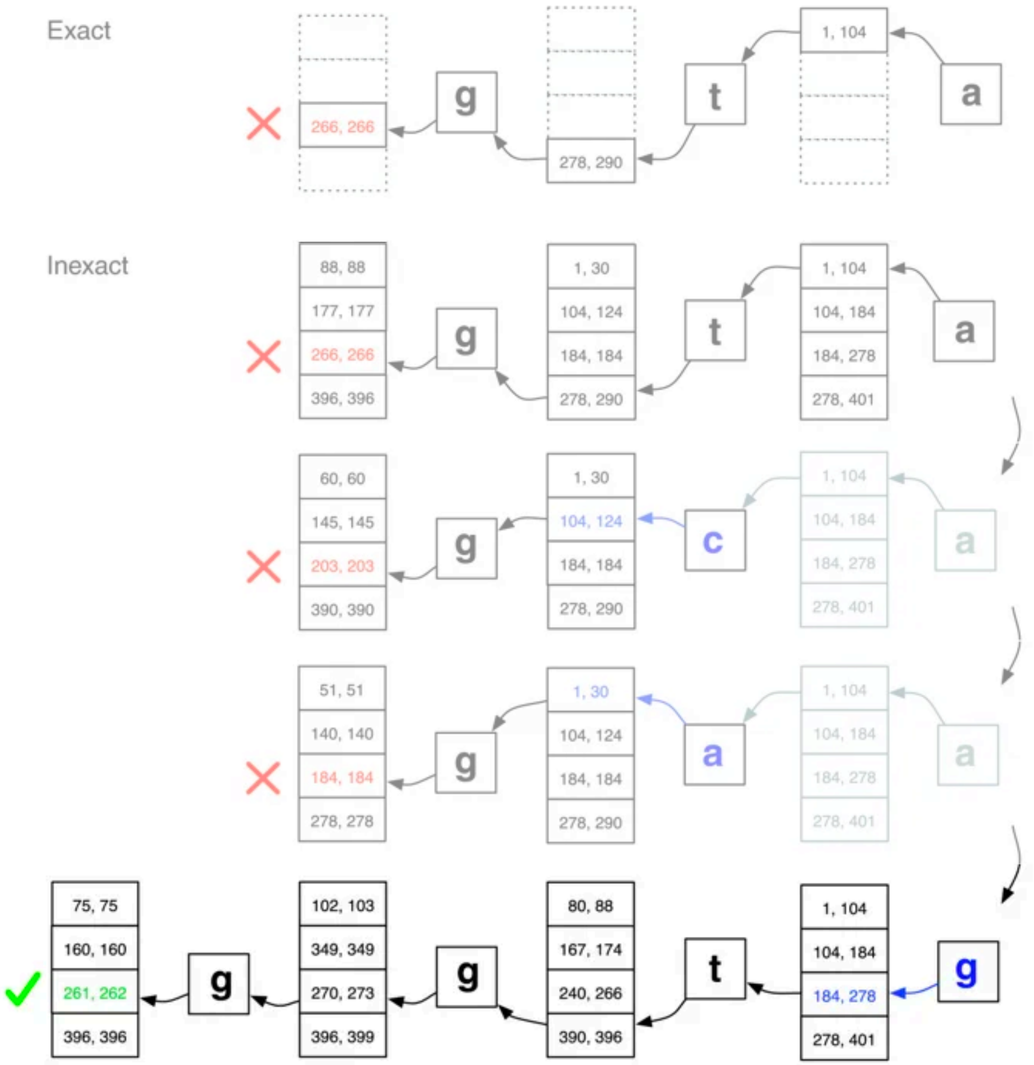
Backtracking

Start by matching the exact sequence

If the algorithm reaches a point with no matches swap out characters already matched and restart search from that there

When ties occur, start with the character with the lowest quality score, keep the rest in a stack

Keep track of how many changes are made



"*Bowtie conducts a quality-aware, greedy, randomized, depth-first search through the space of possible alignments.*"

Backtracking Options

The user specifies the sum of the quality scores that can be changed

- this means that a mapping can have lots of low quality replacements, or
- one medium quality change

Bowtie outputs the first valid alignment by default (within the specified constraints)

- can be modified to complete the backtracking and return the "best" alignment
- 2x-3x slower to do this

User can specify a number of alignments to consider

- default is to use only one
- might want the two best alignments
- 2 alignments is ~2x slower than using only 1

Excessive Backtracking

In low quality reads, lots of time may be spent backtracking since there are many possible changes at low quality positions.

They mitigate this by creating two indexes (as we saw previously), one for the forward and one for the reverse of the string

- the backtracking is performed somewhat simultaneously on both index as we will see next

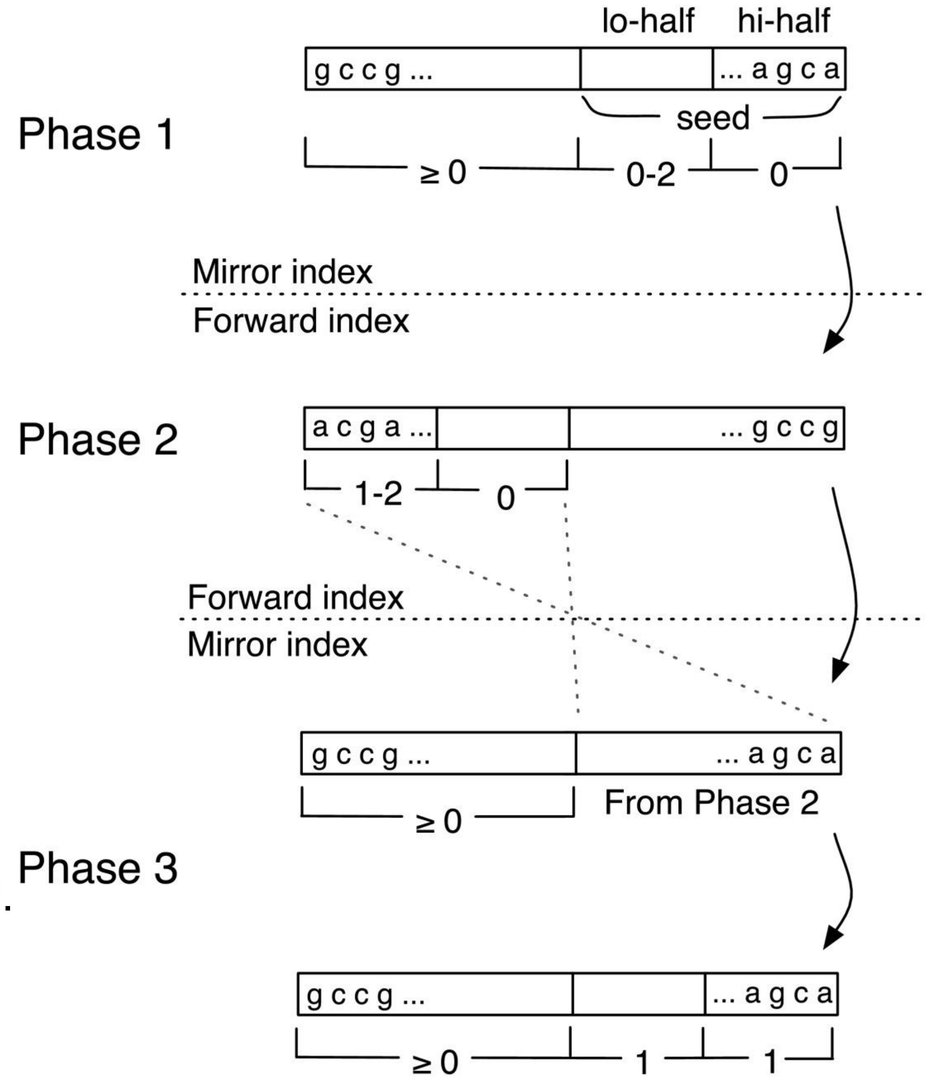
One other step they take is to concentrate on the "high-quality" end of a read (the first 28 characters read) which is most reliable

Phased Search

Split the seed (first 28 bases) into two parts, hi-half and lo-half

Assume we're allowing 2 changes in the seed, a good alignment will have either:

1. no mismatches
2. no mismatches in hi-half, 1 or 2 mismatches in lo-half
3. 1 or 2 mismatches in hi-half, no mismatches in lo-half
4. 1 mismatch in hi-half, 1 mismatch in lo-half

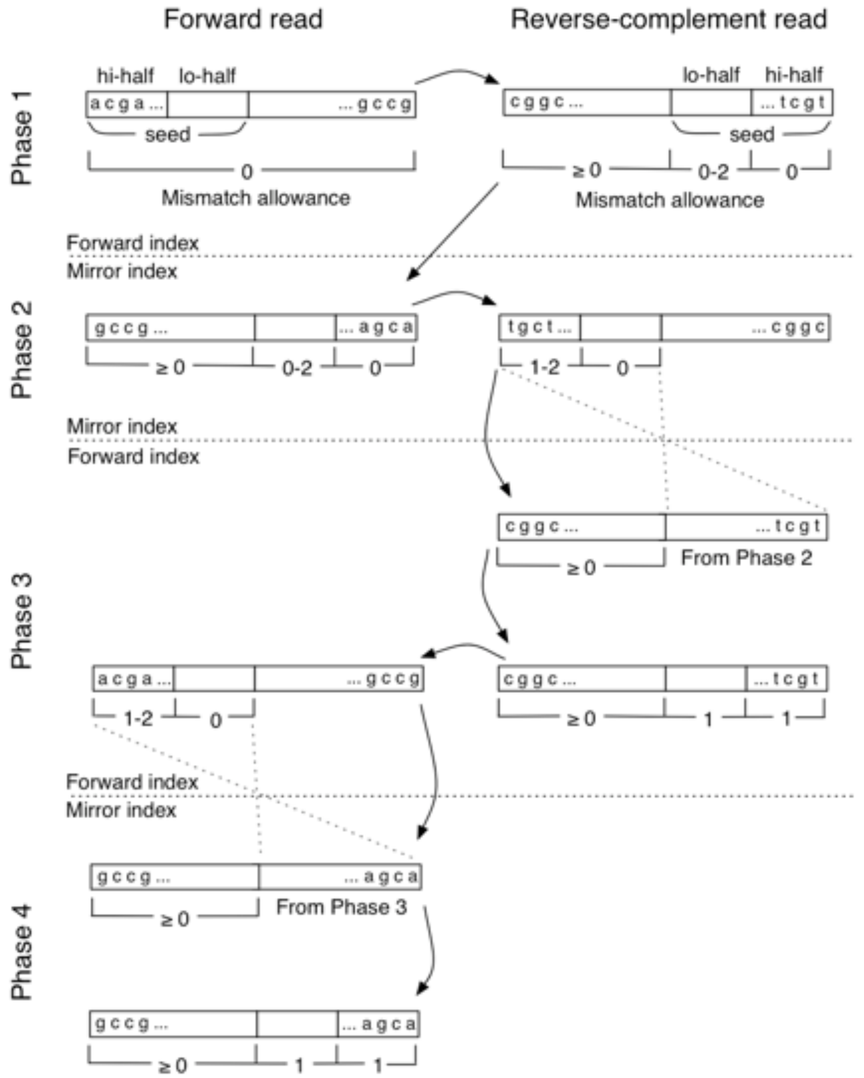


Phase 1 uses the mirror index and invokes the aligner to find alignments for cases 1 & 2.
Phases 2 and 3 cooperate to find alignments for case 3:
Phase 2 finds partial alignments with mismatches only in the hi-half, and phase 3 attempts to extend those partial alignments into full alignments.
Finally, phase 3 invokes the aligner to find alignments for case 4.

Phased search with reverse strand

Since both the read and its reverse complement are possibilities for ma match, need to consider both.

Phases 2-4 here map to phases 1-3 previously.



Performance

Maq (Li, Ruan, Durban 2008), SOAP (Li, Li, Kristiansen, Wang 2008) the leading competitors at the time

Both used hashing to find potential mapping locations

Performance

	Platform	CPU time	Wall clock time	Reads mapped per hour (millions)	Peak virtual memory footprint (megabytes)	Bowtie speed-up	Reads aligned (%)
Bowtie -v 2	Server	**15 m 7 s**	**15 m 41 s**	**33.8**	**1,149**	351×	**67.4**
SOAP		91 h 57 m 35 s	91 h 47 m 46 s	0.10	13,619		67.3
Bowtie	PC	**16 m 41 s**	**17 m 57 s**	**29.5**	**1,353**	59.8×	**71.9**
MAQ		17 h 46 m 35 s	17 h 53 m 7 s	0.49	804		74.7
Bowtie	Server	**17 m 58 s**	**18 m 26 s**	**28.8**	**1,353**	107×	**71.9**
MAQ		32 h 56 m 53 s	32 h 58 m 39 s	0.27	804		74.7

Take Aways

Bowtie was (at the time) the fastest short read aligniner

Used a one-time index based on a BWT that could be reused (novel at the time)

Is able to run on a standard PC

When first published didn't use mate-pair information

Bowtie2

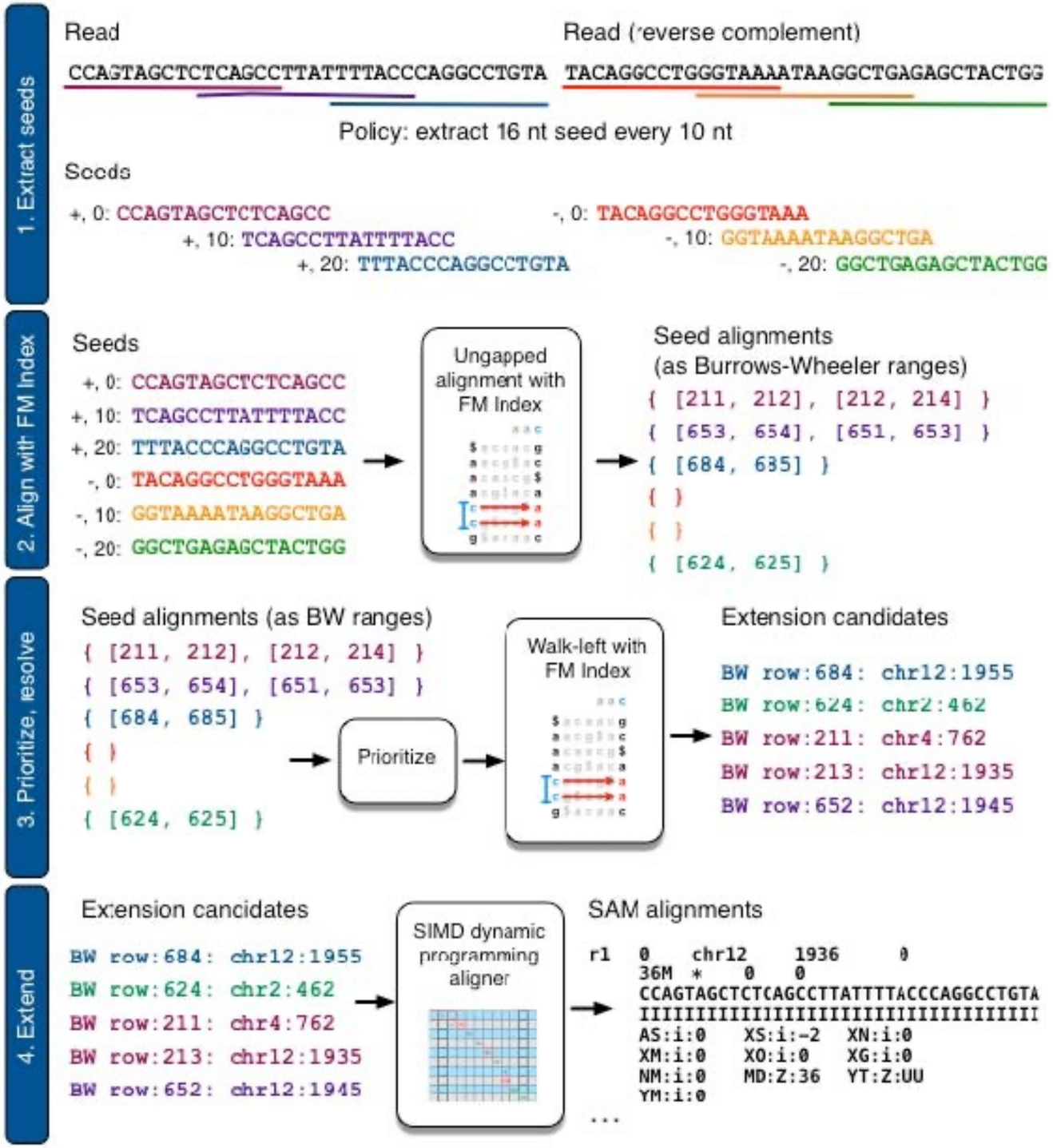
Brief Communication | Published: 04 March 2012

Fast gapped-read alignment with Bowtie 2

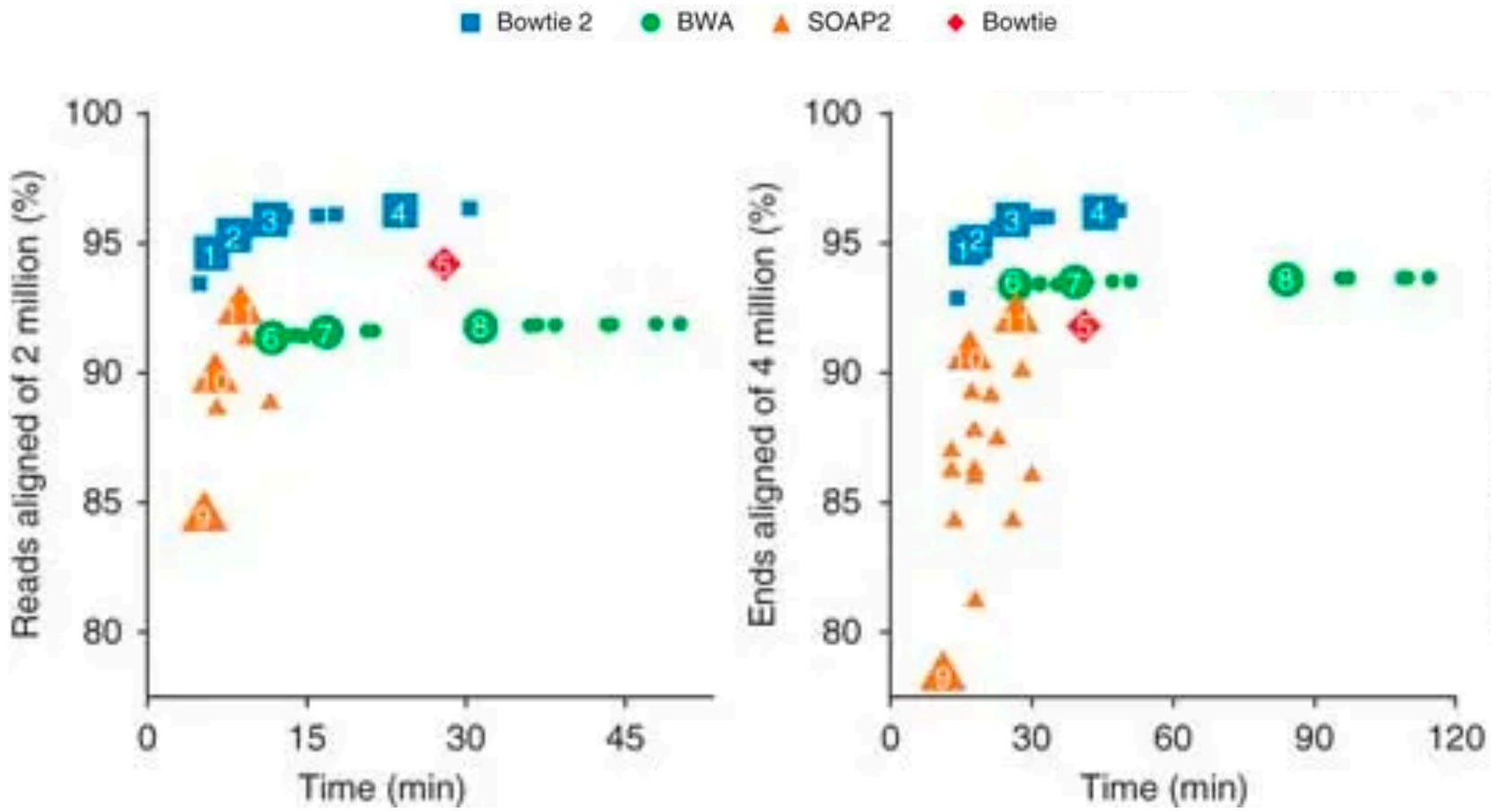
Ben Langmead & Steven L Salzberg

Nature Methods **9**, 357–359(2012) | Cite this article

Bowtie2


1. Extract seeds
Read
CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
Read (reverse complement)
TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG
Policy: extract 16 nt seed every 10 nt
Seeds
+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG
2. Align with FM Index
Seeds
+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG
Ungapped alignment with FM Index
Seed alignments (as Burrows-Wheeler ranges)
{ [211, 212], [212, 214] }
{ [653, 654], [651, 653] }
{ [684, 685] }
{ }
{ }
{ [624, 625] }
3. Prioritize, resolve
Seed alignments (as BW ranges)
{ [211, 212], [212, 214] }
{ [653, 654], [651, 653] }
{ [684, 685] }
{ }
{ }
{ [624, 625] }
Prioritize
Walk-left with FM Index
Extension candidates
BW row:684: chr12:1955
BW row:624: chr2:462
BW row:211: chr4:762
BW row:213: chr12:1935
BW row:652: chr12:1945
4. Extend
Extension candidates
BW row:684: chr12:1955
BW row:624: chr2:462
BW row:211: chr4:762
BW row:213: chr12:1935
BW row:652: chr12:1945
SIMD dynamic programming aligner
SAM alignments
r1 0 chr12 1936 0
36M * 0 0
CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
AS:i:0 XS:i:-2 XN:i:0
XM:i:0 XO:i:0 XG:i:0
NM:i:0 MD:Z:36 YT:Z:UU
YM:i:0
...

Bowtie2


Bowtie 2
BWA
SOAP2
Bowtie
Reads aligned of 2 million (%)
100
95
90
85
80
0
15
30
45
Time (min)
Ends aligned of 4 million (%)
100
95
90
85
80
0
30
60
90
120
Time (min)