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

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

Akshay Sanjeev Keloth

May 9, 2025

Outline

- The past and bowtie
- Background of the algorithm
- Overview of the algorithm
- Results and comparison

We are in 2009

- 2005: First NGS.
- Human genome and human microbiome projects
- 2010s: Real-time sequencing(PacBio and Oxford Nanopore)
- High throughput and large datasets made available.

Pre-existing methods(SOAP and Maq) were computationally expensive and not accessible for many. Bowtie was developed into this world!

Bowtie uses BWT indexing and EXACTMATCH algorithm with specific modifications.

Burrows- Wheeler Transform[1]: Forward Transform

Let $T = \texttt{BANANA}$. BWT(T) will be:

$$\begin{bmatrix} \$ & B & A & N & A & N & A \\ A & \$ & B & A & N & A & N \\ N & A & \$ & B & A & N & A \\ A & N & A & \$ & B & A & N \\ N & A & N & A & \$ & B & A \\ A & N & A & N & A & \$ & B \\ B & A & N & A & N & A & \$ \end{bmatrix}$$

[1] Burrows, Michael; Wheeler, David J. (May 10, 1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, archived from the original on January 5, 2003

Burrows- Wheeler Transform[1]: Forward Transform

Let $T = \texttt{BANANA}$. BWT(T) will be:

$$\begin{bmatrix} \$ & B & A & N & A & N & A \\ A & \$ & B & A & N & A & N \\ N & A & \$ & B & A & N & A \\ A & N & A & \$ & B & A & N \\ N & A & N & A & \$ & B & A \\ A & N & A & N & A & \$ & B \\ B & A & N & A & N & A & \$ \end{bmatrix} \rightarrow \begin{bmatrix} \$ & B & A & N & A & N & A \\ A & \$ & B & A & N & A & N \\ A & N & A & \$ & B & A & N \\ A & N & A & N & A & \$ & B \\ B & A & N & A & N & A & \$ \\ N & A & \$ & B & A & N & A \\ N & A & N & A & \$ & B & A \end{bmatrix}$$

$$BWT(T) \rightarrow \texttt{ANNB\$AA}$$

[1] Burrows, Michael; Wheeler, David J. (May 10, 1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, archived from the original on January 5, 2003

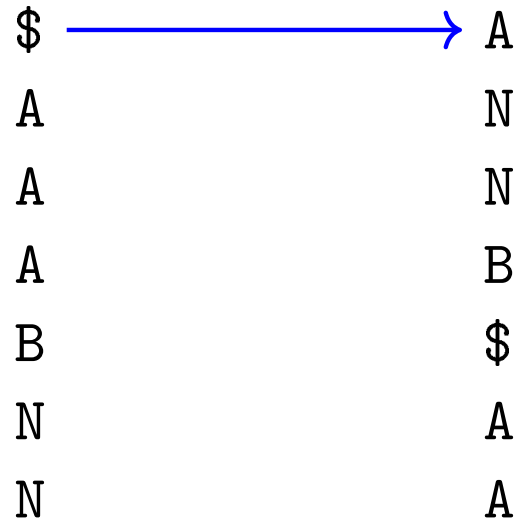
BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:

$	A
A	N
A	N
A	B
B	$
N	A
N	A

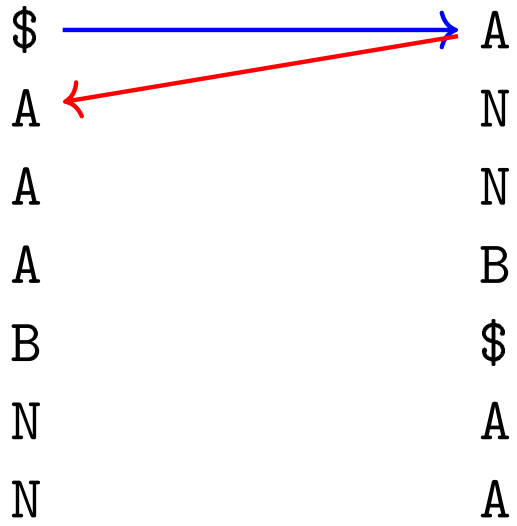
BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:



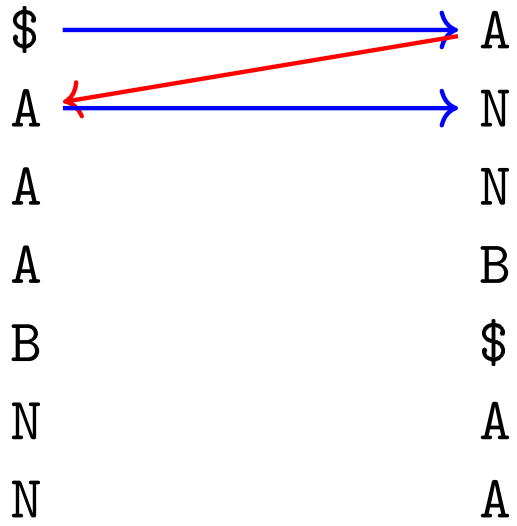
BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:



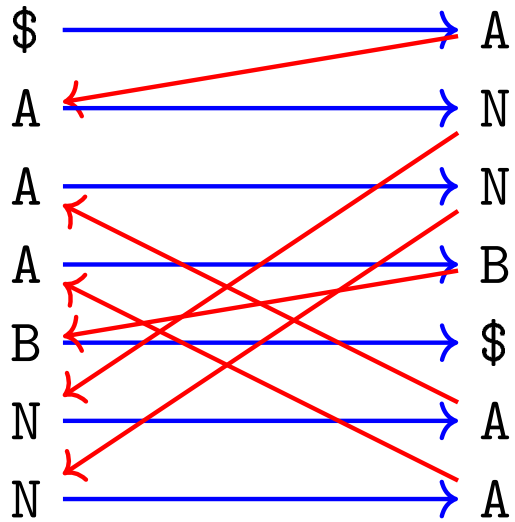
BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:



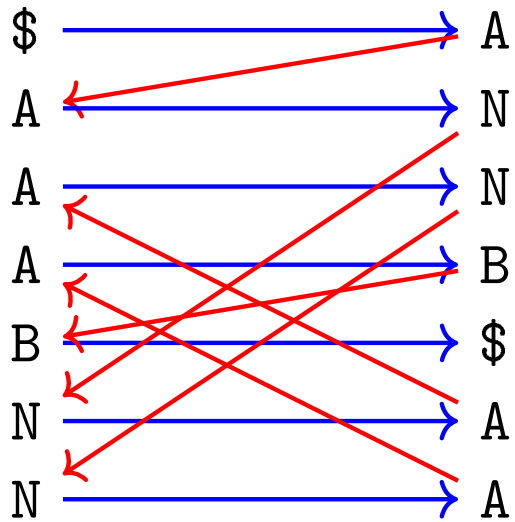
BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:



BWT: Reverse Transform(UNPERMUTE)

BWT is reversible:



follow the tip of blue arrows to get back the initial message:

BANANA$

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

$	$\mathtt{A}_1$
$\mathtt{A}_1$	$\mathtt{N}_1$
$\mathtt{A}_2$	$\mathtt{N}_2$
$\mathtt{A}_3$	$\mathtt{B}_1$
$\mathtt{B}_1$	$
$\mathtt{N}_1$	$\mathtt{A}_2$
$\mathtt{N}_2$	$\mathtt{A}_3$

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

0	$\mathtt{\$}$	$\mathtt{A}_1$	1
1	$\mathtt{A}_1$	$\mathtt{N}_1$	5
2	$\mathtt{A}_2$	$\mathtt{N}_2$	6
3	$\mathtt{A}_3$	$\mathtt{B}_1$	4
4	$\mathtt{B}_1$	$\mathtt{\$}$	0
5	$\mathtt{N}_1$	$\mathtt{A}_2$	2
6	$\mathtt{N}_2$	$\mathtt{A}_3$	3

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

top→	0	\$	$\mathtt{A}_1$	1
	1	$\mathtt{A}_1$	$\mathtt{N}_1$	5
	2	$\mathtt{A}_2$	$\mathtt{N}_2$	6
	3	$\mathtt{A}_3$	$\mathtt{B}_1$	4
	4	$\mathtt{B}_1$	\$	0
	5	$\mathtt{N}_1$	$\mathtt{A}_2$	2
bot→	6	$\mathtt{N}_2$	$\mathtt{A}_3$	3

[2]Paolo Ferragina and Giovanni Manzini (2000). "Opportunistic Data Structures with Applications". Proceedings of the 41st Annual Symposium on Foundations of Computer Science. p.390.

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

	0	$	$\mathtt{A}_1$	1
	1	$\mathtt{A}_1$	$\mathtt{N}_1$	5
	2	$\mathtt{A}_2$	$\mathtt{N}_2$	6
	3	$\mathtt{A}_3$	$\mathtt{B}_1$	4
	4	$\mathtt{B}_1$	$	0
top $\mathtt{N}_1 \rightarrow$	5	$\mathtt{N}_1$	$\mathtt{A}_2$	2
bot $\mathtt{N}_2 \rightarrow$	6	$\mathtt{N}_2$	$\mathtt{A}_3$	3

[2] Paolo Ferragina and Giovanni Manzini (2000). "Opportunistic Data Structures with Applications". Proceedings of the 41st Annual Symposium on Foundations of Computer Science. p.390.

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

	0	$	$\mathtt{A}_1$	1
	1	$\mathtt{A}_1$	$\mathtt{N}_1$	5
top $\mathtt{A}_2\mathtt{N}_1\rightarrow$	2	$\mathtt{A}_2$	$\mathtt{N}_2$	6
bot $\mathtt{A}_3\mathtt{N}_2\rightarrow$	3	$\mathtt{A}_3$	$\mathtt{B}_1$	4
	4	$\mathtt{B}_1$	$	0
	5	$\mathtt{N}_1$	$\mathtt{A}_2$	2
	6	$\mathtt{N}_2$	$\mathtt{A}_3$	3

[2]Paolo Ferragina and Giovanni Manzini (2000). "Opportunistic Data Structures with Applications". Proceedings of the 41st Annual Symposium on Foundations of Computer Science. p.390.

EXACTMATCH Algoritham using FM Inedx[2]

We have, $T = \mathtt{B}_1\mathtt{A}_1\mathtt{N}_1\mathtt{A}_2\mathtt{N}_2\mathtt{A}_3$.
Pattern to find, $P = \mathtt{NAN}$

	0	$\mathtt{\$}$	$\mathtt{A}_1$	1
	1	$\mathtt{A}_1$	$\mathtt{N}_1$	5
	2	$\mathtt{A}_2$	$\mathtt{N}_2$	6
	3	$\mathtt{A}_3$	$\mathtt{B}_1$	4
	4	$\mathtt{B}_1$	$\mathtt{\$}$	0
	5	$\mathtt{N}_1$	$\mathtt{A}_2$	2
top/bot: $\mathtt{N}_2\mathtt{A}_2\mathtt{N}_1 \rightarrow$	6	$\mathtt{N}_2$	$\mathtt{A}_3$	3

[2] Paolo Ferragina and Giovanni Manzini (2000). "Opportunistic Data Structures with Applications". Proceedings of the 41st Annual Symposium on Foundations of Computer Science. p.390.

Exact vs Inexact Matching

- EXACTMATCH algorithm is insufficient for DNA short read alignment.
- So, Bowtie algorithm follows a specified alignment policy:
  - Each character in a read has a numeric quality value(m_i), with lower values indicating a higher likelihood of a sequencing error.
  - We allows a limited number of mismatches, while trying to minimize $\sum_i m_i$, where i covers all the mismatches and performs a greedy search.

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

Excessive Backtracking

If a particular suffix doesn't occur in the text, the algorithm can backtrack. Backtracking involves **potentially substituting a different base at an already-matched query position, introducing a mismatch, and then resuming the search.**

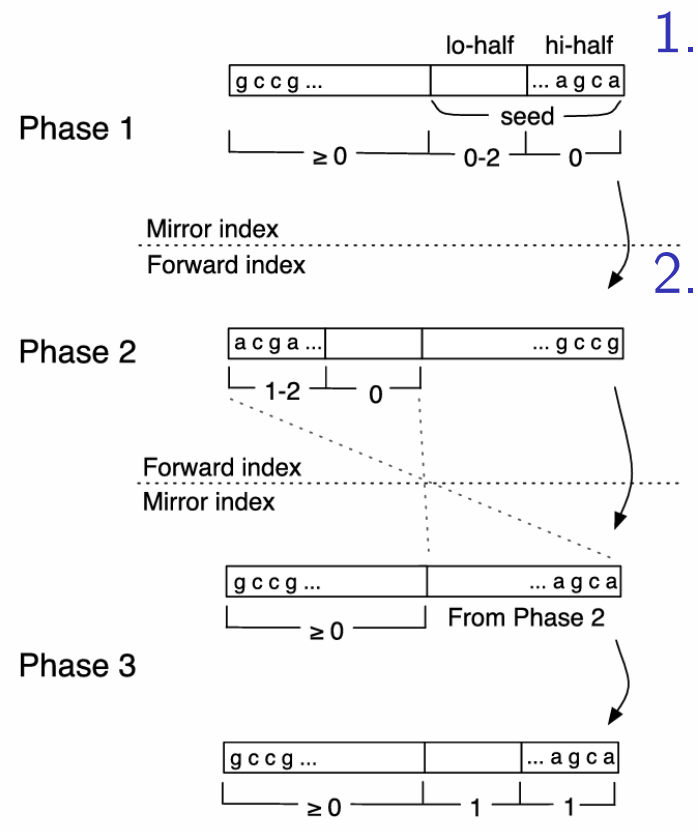
Excessive backtracking occurs when too many alternative paths are explored during the alignment process due to mismatches. Bowtie uses double indexing to deal with this problem. Excessive backtracking is significant only when a read has many low-quality positions and does not align or aligns poorly to the reference.

Mismatch Cases

- The high-quality base-pairs at 3' is called Seed(default: 28).
- Seed is further divided into hi-half and lo-half each with 14bps.
- Assuming the default mismatches(2), a reportable alignment can occur in 4 different ways.
  1. No mismatches in seed
  2. No mismatches in hi-half, one or two mismatches in lo-half
  3. No mismatches in lo-half, one or two mismatches in hi-half
  4. O ne mismatch in hi-half, one mismatch in lo-half

Phased Maq-like Search

Bowtie uses a 3 phase approach:



1. Phase 1 uses the mirror index and invokes the aligner to find alignments for cases 1 and 2.
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 align ments. Finally, phase 3 invokes the aligner to find alignments for case 4.

Summary of Results

- **Speed:** Bowtie is significantly faster (*Table 1*). When aligning 8.84 million 35-bp reads to the human genome, Bowtie on a server achieved 33.8 million reads/CPU hour, which is 351× faster than SOAP (0.10 million reads/hour) and 107× faster than Maq (0.27 million reads/hour).

Summary of Results

- **Memory:** For the human genome, its memory footprint is approximately 1.3 GB. This small footprint allows Bowtie to run on a typical desktop computer with 2 GB of RAM, something SOAP could not do.

Summary of Results

- **Sensitivity:** Bowtie's default sensitivity is comparable to SOAP's (≈67%) and slightly less than Maq's (Bowtie 71.9% vs Maq 74.7% in *Table 1*). The slight difference is mainly because Maq's algorithm is more flexible, sometimes allowing three mismatches in the seed, and also due to Bowtie's backtrack ceiling. Most of this can be manually fine-tuned.

Summary of Results

- **Read Length:** Maq scales better overall to longer reads than Bowtie or SOAP. But, Bowtie in SOAP-like `-v 2` mode scales very well. Bowtie in its default Maq-like mode is slower for 76-bp reads compared to shorter reads but still significantly faster than Maq.

Summary of Results

- **Parallel Performance:** Bowtie supports parallel processing. On a four-core server, using four threads provided a speedup of 3.12× compared to single thread. The memory footprint doesn't increase substantially with more threads as the index is shared(*Table 4*).

Summary of Results

- **Index Building:** Unlike tools that build index-systems during alignment, Bowtie creates a permanent index that can be reused. Even though indexing the human genome takes hours, depending on the memory target, including index construction time, Bowtie still outperforms competing tools when aligning large datasets(*Table 5*)

Perfomance Results

Bowtie alignment performance versus SOAP and Maq

	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	-	67.4
SOAP		91 h 57 m 35 s	91 h 47 m 46 s	0.10	13,619	351×	67.3
Bowtie	PC	16 m 41 s	17 m 57 s	29.5	1,353	-	71.9
Maq		17 h 46 m 35 s	17 h 53 m 7 s	0.49	804	59.8×	74.7
Bowtie	Server	17 m 58 s	18 m 26 s	28.8	1,353	-	71.9
Maq		32 h 56 m 53 s	32 h 58 m 39 s	0.27	804	107×	74.7

Table 1

Perfomance Results

Bowtie alignment performance versus SOAP and Maq

	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	-	67.4
SOAP		91 h 57 m 35 s	91 h 47 m 46 s	0.10	13,619	351×	67.3
Bowtie	PC	16 m 41 s	17 m 57 s	29.5	1,353	-	71.9
Maq		17 h 46 m 35 s	17 h 53 m 7 s	0.49	804	59.8×	74.7
Bowtie	Server	17 m 58 s	18 m 26 s	28.8	1,353	-	71.9
Maq		32 h 56 m 53 s	32 h 58 m 39 s	0.27	804	107×	74.7

Table 1

Bowtie alignment performance versus Maq with filtered read set

	Platform	CPU time	Wall clock time	Reads mapped per hour (millions)	Peak virtual memory footprint (megabytes)	Bowtie speed up	Reads aligned (%)
Bowtie	PC	16 m 39 s	17 m 47 s	29.8	1,353	-	74.9
Maq		11 h 15 m 58 s	11 h 22 m 2 s	0.78	804	38.4×	78.0
Bowtie	Server	18 m 20 s	18 m 46 s	28.3	1,352	-	74.9
Maq		18 h 49 m 7 s	18 h 50 m 16 s	0.47	804	60.2×	78.0

Table 2

Perfomance Results-II

Varying read length using Bowtie, Maq and SOAP

Length	Program	CPU time	Wall clock time	Peak virtual memory footprint (megabytes)	Bowtie speed-up	Reads aligned (%)
36 bp	Bowtie	6 m 15 s	6 m 21 s	1,305	-	62.2
	Maq	3 h 52 m 26 s	3 h 52 m 54 s	804	36.7×	65.0
	Bowtie -v 2	4 m 55 s	5 m 00 s	1,138	-	55.0
	SOAP	16 h 44 m 3 s	18 h 1 m 38 s	13,619	216×	55.1
50 bp	Bowtie	7 m 11 s	7 m 20 s	1,310	-	67.5
	Maq	2 h 39 m 56 s	2 h 40 m 9 s	804	21.8×	67.9
	Bowtie -v 2	5 m 32 s	5 m 46 s	1,138	-	56.2
	SOAP	48 h 42 m 4 s	66 h 26 m 53 s	13,619	691×	56.2
76 bp	Bowtie	18 m 58 s	19 m 6 s	1,323	-	44.5
	Maq 0.7.1	4 h 45 m 7 s	4 h 45 m 17 s	1,155	14.9×	44.9
	Bowtie -v 2	7 m 35 s	7 m 40 s	1,138	-	31.7

Table 3

Perfomance Results-III

Bowtie parallel alignment performance

	CPU time	Wall clock time	Reads mapped per hour (millions)	Peak virtual memory footprint (megabytes)	Speedup
Bowtie, one thread	18 m 19 s	18 m 46 s	28.3	1,353	-
Bowtie, two threads	20 m 34 s	10 m 35 s	50.1	1,363	1.77×
Bowtie, four threads	23 m 9 s	6 m 1 s	88.1	1,384	3.12×

Table 4

Perfomance Results-III

Bowtie parallel alignment performance

	CPU time	Wall clock time	Reads mapped per hour (millions)	Peak virtual memory footprint (megabytes)	Speedup
Bowtie, one thread	18 m 19 s	18 m 46 s	28.3	1,353	-
Bowtie, two threads	20 m 34 s	10 m 35 s	50.1	1,363	1.77×
Bowtie, four threads	23 m 9 s	6 m 1 s	88.1	1,384	3.12×

Table 4

Bowtie index building performance

Physical memory target (GB)	Actual peak memory footprint (GB)	Wall clock time
16	14.4	4 h 36 m
8	5.84	5 h 5 m
4	3.39	7 h 40 m
2	1.39	21 h 30 m

Table 5

Summary

- Bowtie's speed and small memory footprint are due to its use of the Burrows-Wheeler-FM index in combination with the novel, quality-aware, backtracking algorithm introduced here. Double indexing is used to avoid the performance penalty of excessive backtracking.
- Bowtie exhibits a large performance advantage over both Maq and SOAP when mapping reads to the human genome.
- Unlike many other short-read aligners, Bowtie creates a per manent index of the reference that may be re-used across alignment runs.

Bowtie 2

- Bowtie 2 supports gapped alignments, enabling mapping read with in-dels or structural variations.
- Bowtie 2 is optimized for aligning longer reads (100 bp or more), which are common in newer sequencing technologies. Bowtie 1 was designed for very short reads (25–50 bp) and struggled with longer ones.
- Bowtie 2 allows for local alignments. This is particularly useful for RNA-seq or metagenomic data. Bowtie 1 required the entire read to align end-to-end.

Thank you!