



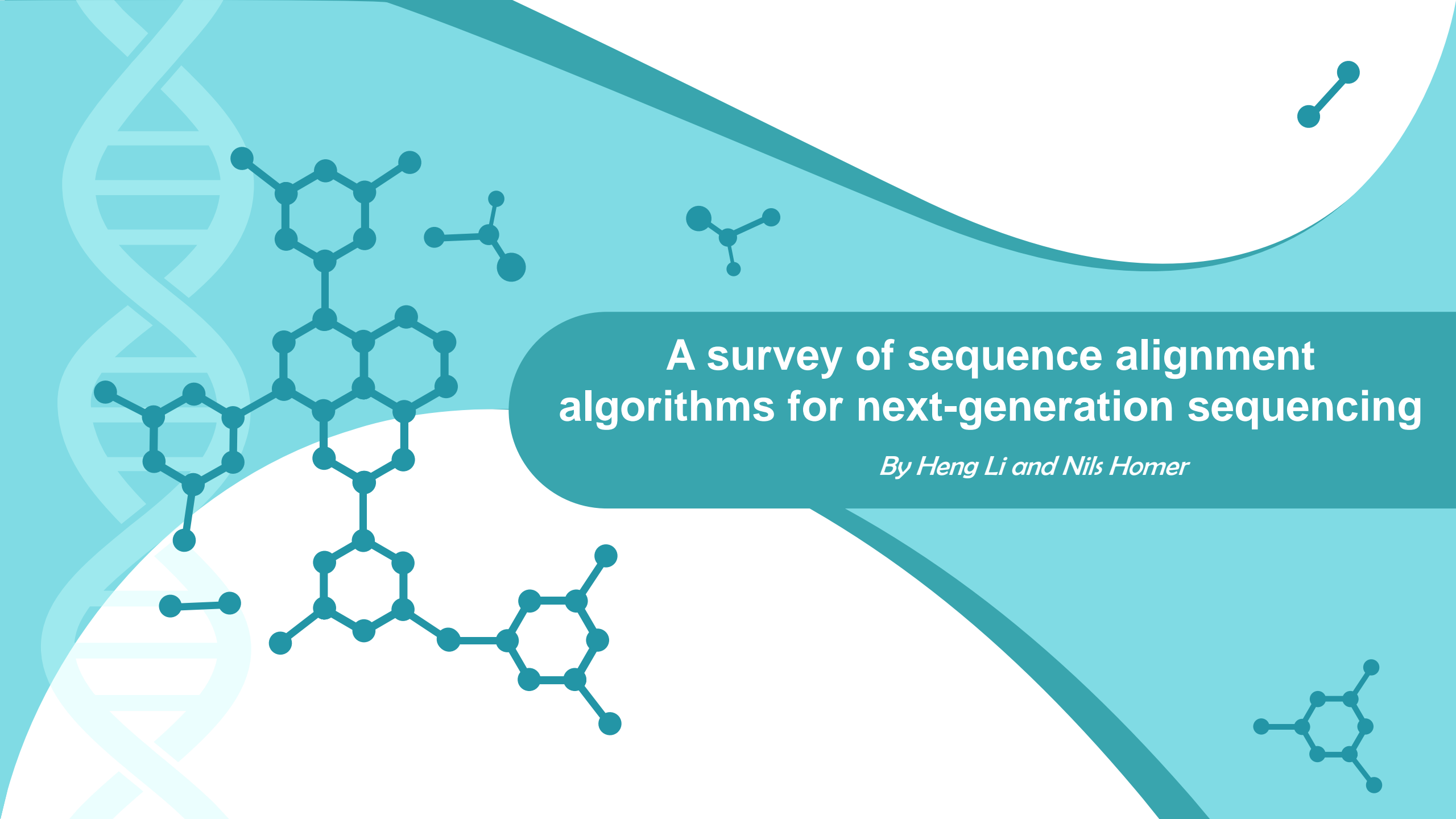
Papers to be Presented

**A survey of sequence alignment
algorithms for next-generation
sequencing**

&

**Fast and accurate short read
alignment with Burrows–Wheeler
transform**

By: Nabila Shahnaz Khan



A survey of sequence alignment algorithms for next-generation sequencing

By Heng Li and Nils Homer

Introduction

01

Sequence alignment is essential to nearly all the applications of new sequencing technologies.

02

The time complexity of comparing sequence reads to a reference is a challenge due to enormous amount of reads and large size of reference sequence

03

In this article, the authors systematically reviewed the current development of sequence alignment algorithms, introduced their practical applications and discussed future development of alignment

Alignment Algorithms

Based on Hash Tables

Exact Seed Match

BLAST

Spaced Seed Match

ELAND
SOAP
MAQ
SeqMap
RMAP

Gapped Seed Match

SHRiMP
RazerS
SSAHA2
BLAT

Based on Suffix Trees

Suffix Tree

MUMer
OASIS

Implicit Suffix Tree

Vmatch
Segemehl

FM-index

Bowtie
BWA
SOAP2
BWT-SW

Based on Merge Sorting

Slider,
SlideII

Algorithms based on hash tables

- Used to map a set of short query sequences against a long reference genome of the same species
- Follows the same seed-and-extend paradigm

STEPS followed in BLAST:

01

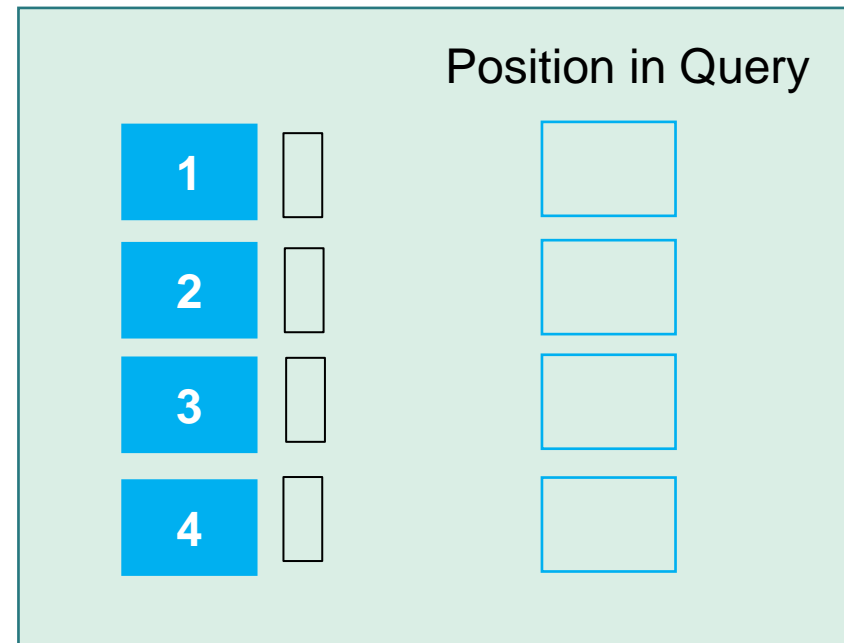
keeps the position of each k-mer subsequence of the query in a hash table with the k-mer sequence being the key; default 11-mer

Query Sequence:

GACAGAC

Database Sequence:

ACGGATTCCATAT



Hash Table

Algorithms based on hash tables

- Used to map a set of short query sequences against a long reference genome of the same species
- Follows the same seed-and-extend paradigm

STEPS followed in BLAST:

01

keeps the position of each k-mer subsequence of the query in a hash table with the k-mer sequence being the key; default 11-mer

Query Sequence:

GACAGAC

Database Sequence:

ACGGATTCCATAT

Position in Query		
1	GAC	1
2		
3		
4		

Hash Table

Algorithms based on hash tables

- Used to map a set of short query sequences against a long reference genome of the same species
- Follows the same seed-and-extend paradigm

STEPS followed in BLAST:

01

keeps the position of each k-mer subsequence of the query in a hash table with the k-mer sequence being the key; default 11-mer

Query Sequence:

GACAGAC

Database Sequence:

ACGGATTCCATAT

Position in Query		
1	GAC	1
2	ACA	2
3		
4		

Hash Table

Algorithms based on hash tables

- Used to map a set of short query sequences against a long reference genome of the same species
- Follows the same seed-and-extend paradigm

STEPS followed in BLAST:

01

keeps the position of each k-mer subsequence of the query in a hash table with the k-mer sequence being the key; default 11-mer

Query Sequence:
GACAGAC

Database Sequence:
ACGGATTCCATAT

Position in Query		
1	GAC	1, 5
2	ACA	2
3	CAG	3
4	AGA	4

Hash Table

Algorithms based on hash tables

02

scans the database sequences for k-mer exact matches, called seeds, by looking up the hash table

Database Sequence:
TGACAGATTC

Position in Query		
1	GAC	1, 5
2	ACA	2
3	CAG	3
4	AGA	4

Hash Table

3-mer	Coordinates	
GAC	2	1,5
ACA	3	2
CAG	4	3
AGA	5	4

Match Table

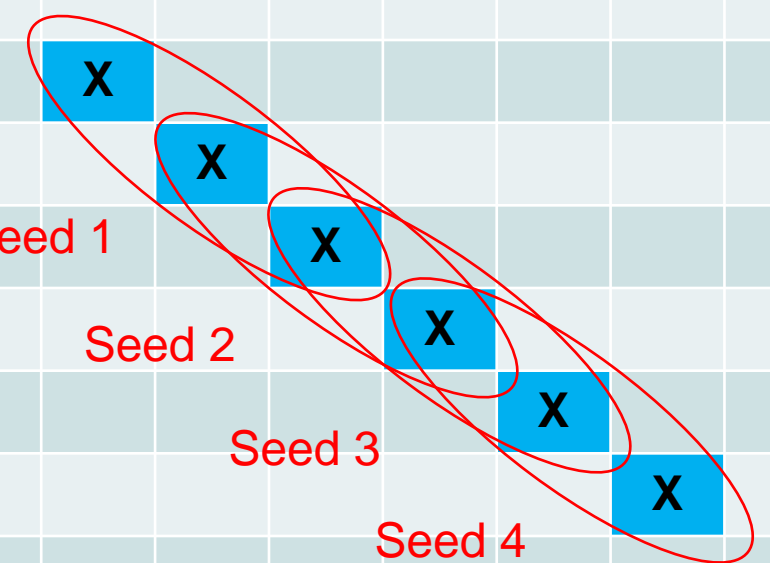
Seeds

Algorithms based on hash tables

03

BLAST extends and joins the seeds first without gaps and then refines them by a Smith–Waterman alignment. It outputs statistically significant local alignments as the final results

		T	G	A	C	A	G	A	T	T	C
	0	0	0	0	0	0	0	0	0	0	0
T	0										
G	0		X								
A	0			X							
C	0	Seed 1			X						
A	0			Seed 2		X					
G	0						X				
A	0				Seed 3			X			
T	0								X		
T	0										
C	0										



Improvements on seeds and seed extension

Improvement on seeding: spaced seed

- Seeding with non-consecutive matches improves sensitivity [1]
- A template '111010010100110111' requiring 11 matches at the '1' positions is 55% more sensitive than BLAST's default template '11111111111'
- Ex: ELAND, SOAP, MAQ, SeqMap, RMAP, ZOOM

Improvement on seeding: q-gram filter and multiple seed hits

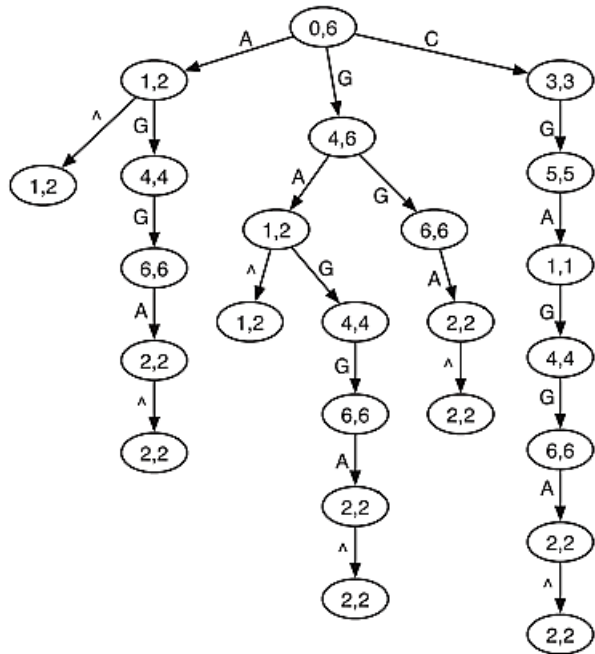
- q-gram filter provides a possible solution to building an index natively allowing gaps
- It is based on the observation that at the occurrence of a w -long query string with at most k differences (mismatches and gaps), the query and the w -long database substring share at least $(w + 1) - (k + 1)q$ common substrings of length q
- Ex: SHRiMP, RazerS, SSAHA2, BLAT

Improvements on seed extension

- Accelerating the standard Smith–Waterman with vectorization; Ex: CLC Genomics Workbench, SHRiMP
- Constraining dynamic programming around seeds already found in the seeding step

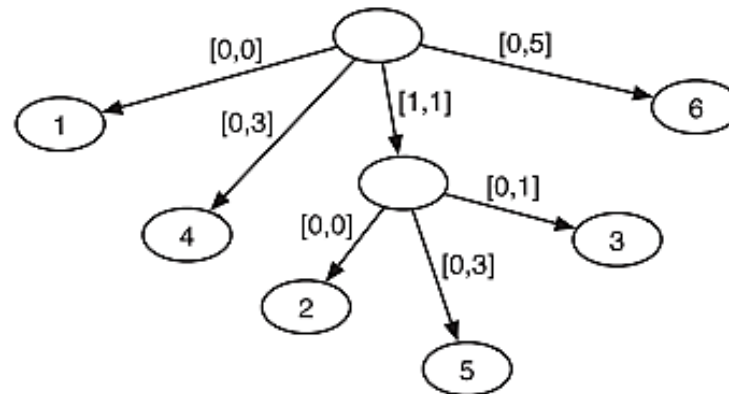
Algorithms based on suffix/prefix tries

- Reduces the inexact matching problem to the exact matching problem
- Involve two steps: identifying exact matches and building inexact alignments supported by exact matches
- Rely on a certain representation of suffix/prefix trie, such as suffix tree, enhanced suffix array and FM-index



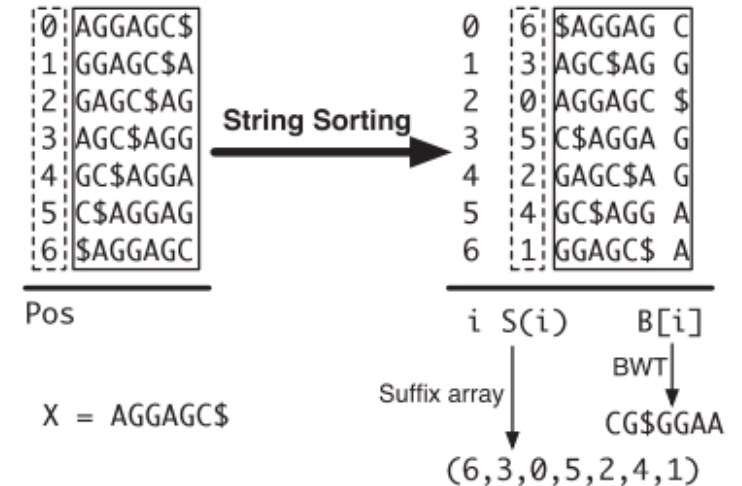
Prefix Trie

[O|Query| time, $O(L^2)$ space]



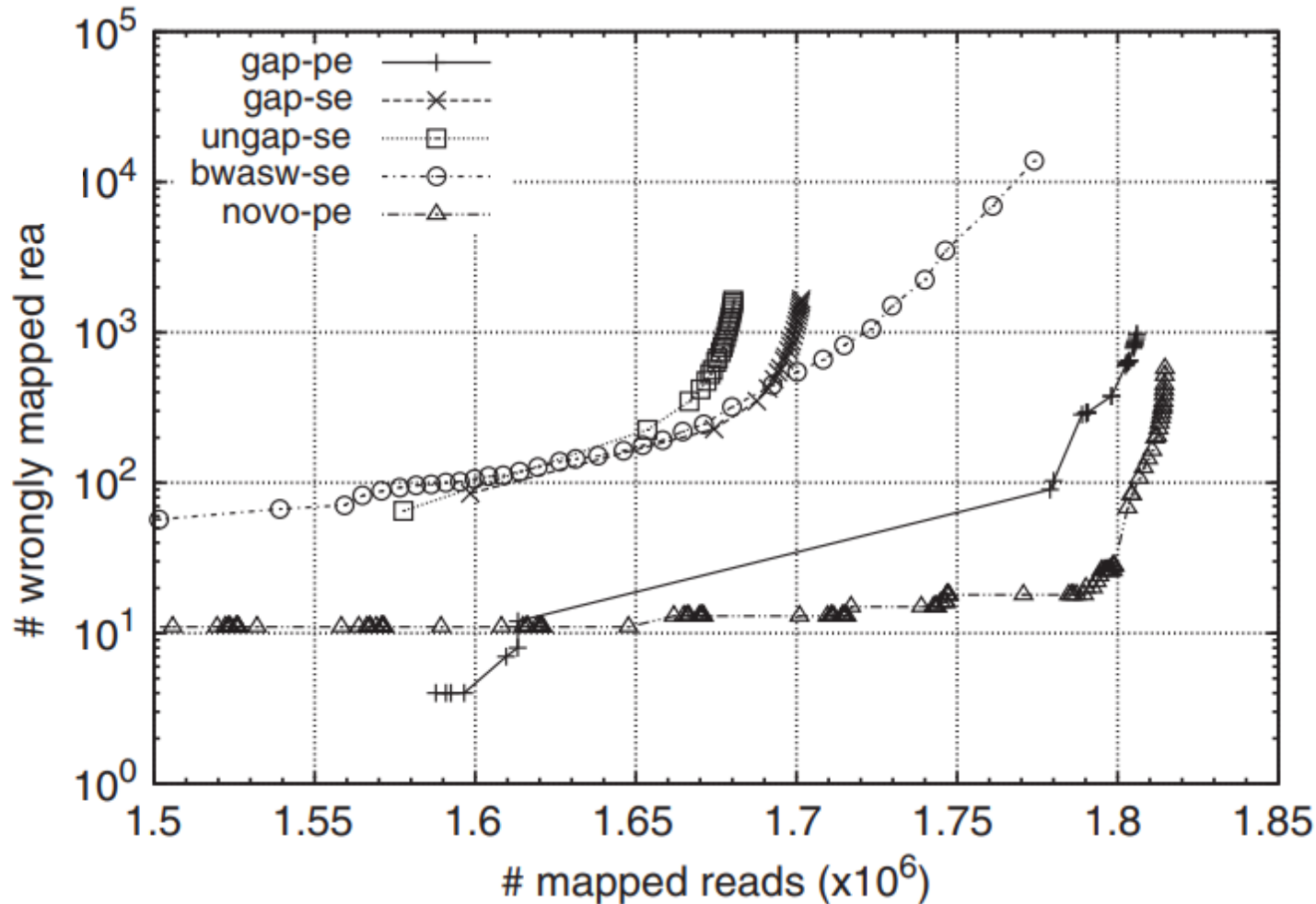
Prefix Tree
[O(L) space]

String: AGGAGC, length L



FM-index using BWT
[O(L) space, $4L \cdot \log L$ bits]

Other Considerations



- Effect of gapped alignment
- Role of paired-end and mate-pair mapping
- Using base quality in alignment

gap-pe = gapped paired-end (PE)
gap-se = gapped single-end (SE)
ungap-se = ungapped SE
bwasw-se = BWA-SW SE

Future Implications

- Cloud computing, with data uploaded and analyzed in a shared cloud
- Simultaneous alignment against multiple genomes



Fast and accurate short read alignment with Burrows–Wheeler transform

Heng Li and Richard Durbin

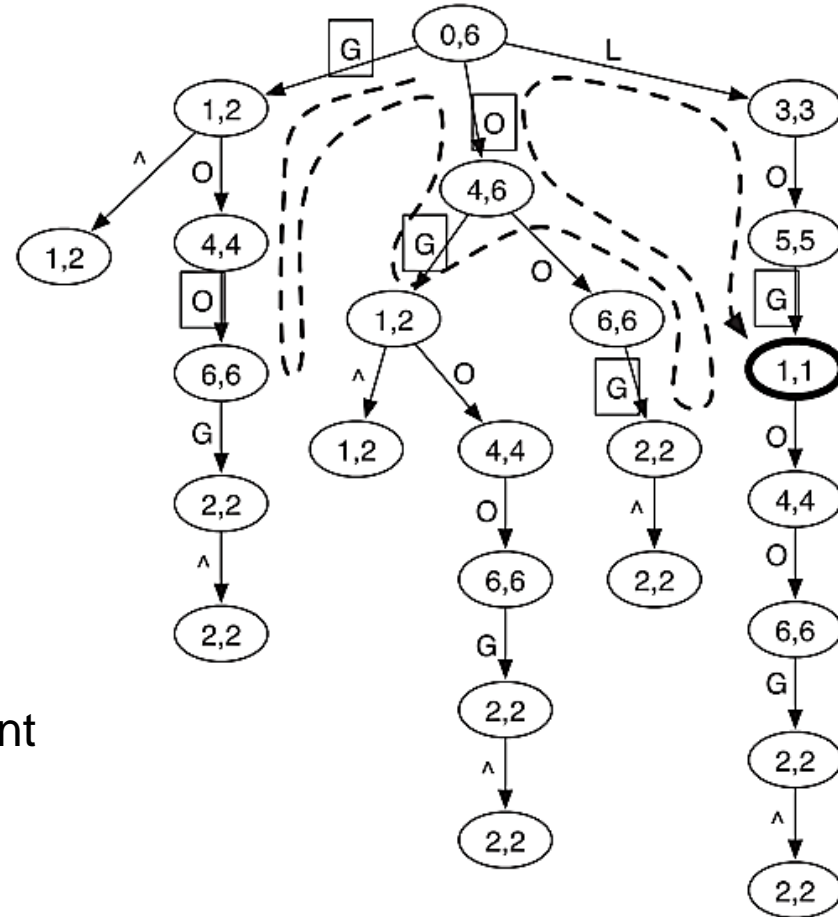
Introduction

- 01 BWA tool is based on backward search with BWT, allows mismatches and gaps
- 02 Efficiently aligns short sequencing reads against a large reference sequence using backward search with BWT. For inexact search, BWA outputs the distinct substrings that are less than k edit distance away from the query read
- 03 Input: Illumina sequence reads up to 100bp. Output: Aligned SAM format.

Prefix Trie and string matching

List of Prefixes
G
G O
G O O
G O O G
G O O G O
G O O G O L

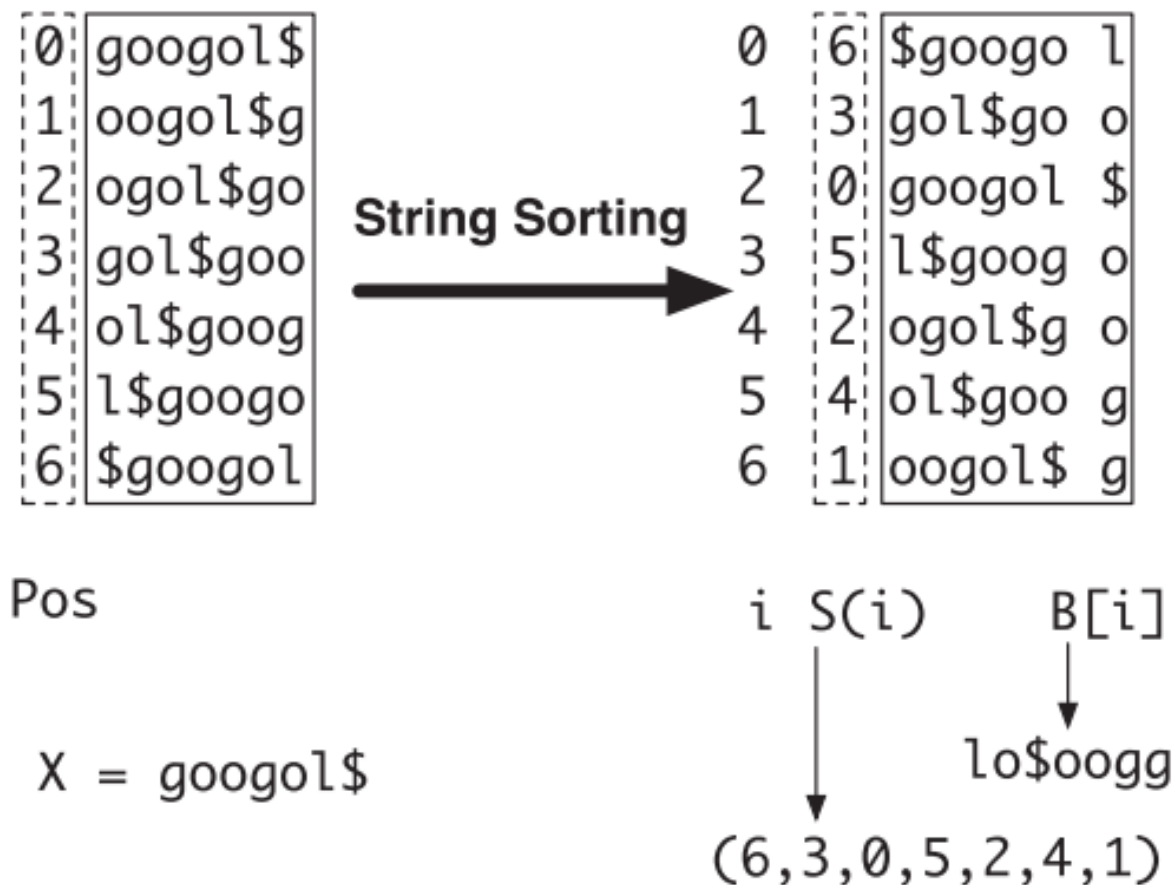
Needs $O(|W|)$ time to search
for a query W , space $O|X|$ with large coefficient



Prefix Tree of the string 'GOOGOL'

Burrows–Wheeler transform

- Suffix array S , $S(i)$ is the start position of the i -th smallest suffix.
- The BWT of X is defined as $B[i]=\$$ when $S(i)=0$ and $B[i]=X[S(i)-1]$ otherwise



Observation 1: All the suffixes that have a string W as prefix, are stored together in the BWM

Suffix array interval and sequence alignment

Range of interval of W in SA

$$\underline{R}(W) = \min \{k : W \text{ is the prefix of } X_{S(k)}\}$$

$$\overline{R}(W) = \max \{k : W \text{ is the prefix of } X_{S(k)}\}$$

- If W is an empty string, $\underline{R}(W)=1$ and $\overline{R}(W)=n-1$
- The set of positions of all occurrences of W in X is

$$\{S(k) : \underline{R}(W) \leq k \leq \overline{R}(W)\}$$

Observation 2: Sequence alignment is equivalent to searching for the SA intervals of substrings of X that match the query

BWA for Exact matching: backward search

C(a) \rightarrow number of symbols in $X[0, n-2]$ that are lexicographically smaller than $a \in \Sigma$

O(a, i) \rightarrow the number of occurrences of a in $B[0, i]$

0	6	\$googo l
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

$C(g) = 0$ $C(l) = 2$ $C(o) = 3$
--

$$O(o, i) = \begin{cases} 0 & , i = 0 \\ 1 & , i = 1, 2 \\ 2 & , i = 3 \\ 3 & , 4 \leq i \leq 6 \end{cases}$$

$$O(g, i) = \begin{cases} 0 & , 0 \leq i \leq 4 \\ 1 & , i = 5 \\ 2 & , i = 6 \end{cases}$$

$$O(l, i) = 1; \quad 0 \leq i \leq 6$$

BWA for Exact matching: backward search

- If W is a substring of X :

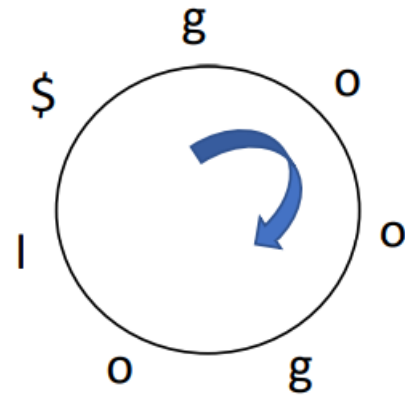
$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W)) - 1 + 1$$

$$\overline{R}(aW) = C(a) + O(a, \overline{R}(W))$$

and $\underline{R}(aW) \leq \overline{R}(aW)$ if and only if aW is a substring of X [2]

- This result makes it possible to test whether W is a substring of X and to count the occurrences of W in $O(|W|)$ time
- This procedure is called **backward search**

Example: $W = \text{'go'}$, $a = \text{'o'}$, $aW = \text{'ogo'}$



BWA for Exact matching: backward search

Example: $W = \text{'go'}$, $a = \text{'o'}$, $aW = \text{'ogo'}$

$X = \text{googol\$}$

0	6	\$googol
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

i $S(i)$ $B[i]$
 ↓ ↓
 (6,3,0,5,2,4,1)
 lo\$oogg

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W)) + 1$$

$$\bar{R}(aW) = C(a) + O(a, \bar{R}(W))$$

Here, $\underline{R}(W) = 1$, $O(\text{'o'}, 0) = 0$, $C(\text{'o'}) = 3$

So, $\underline{R}(aW) = 3 + 0 + 1 = 4$

BWA for Exact matching: backward search

Example: $W = \text{'go'}$, $a = \text{'o'}$, $aW = \text{'ogo'}$

$X = \text{googol\$}$

0	6	\$googol
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

i	S(i)	B[i]
	↓	↓
		lo\$oogg

(6,3,0,5,2,4,1)

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W) - 1) + 1$$

$$\overline{R}(aW) = C(a) + O(a, \overline{R}(W))$$

Here, $\underline{R}(W) \equiv 1$, $O(\text{'o'}, 0) = 0$, $C(\text{'o'}) = 3$

So, $\underline{R}(aW) = 3 + 0 + 1 = 4$

$\overline{R}(W) = 2$, $O(\text{'o'}, 2) = 1$

So, $\overline{R}(aW) = 3 + 1 = 4$

As, $\underline{R}(aW) \leq \overline{R}(aW)$; aW ['ogo'] is a substring of X

$S(4) = 2$

*** If $\underline{R}(aW) \leq \overline{R}(aW)$, aW is a substring of X [2]

BWA for Inexact matching

This recursive algorithm search for the SA intervals of substrings of X that match the query string W with no more than z differences (mismatches or gaps)

Precalculation:

Calculate BWT string B for reference string X

Calculate array $C(\cdot)$ and $O(\cdot, \cdot)$ from B

Calculate BWT string B' for the reverse reference

Calculate array $O'(\cdot, \cdot)$ from B'

Procedures:

$\text{INEXACTSEARCH}(W, z)$

$\text{CALCULATED}(W)$

return $\text{INEXRECUR}(W, |W| - 1, z, 1, |X| - 1)$

Reverse $X = \text{logoog\$}$

F						L
\$	l	o	g	o	o	g
g	\$	l	o	g	o	o
g	o	o	g	\$	l	o
l	o	g	o	o	g	\$
o	g	\$	l	o	g	o
o	g	o	o	g	\$	l
o	o	g	\$	l	o	g

B'

BWA for Inexact matching

$D(i)$ is the lower bound of the number of differences in $W[0,i]$. The better the D is estimated, the smaller the search space and the more efficient the algorithm becomes

If we set $D(i)=0$ for all i and disallow gaps, search space increases

CALCULATED $D(W)$

$k \leftarrow 1$

$l \leftarrow |X| - 1$

$z \leftarrow 0$

for $i=0$ **to** $|W| - 1$ **do**

$k \leftarrow C(W[i]) + O'(W[i], k - 1) + 1$

$l \leftarrow C(W[i]) + O'(W[i], l)$

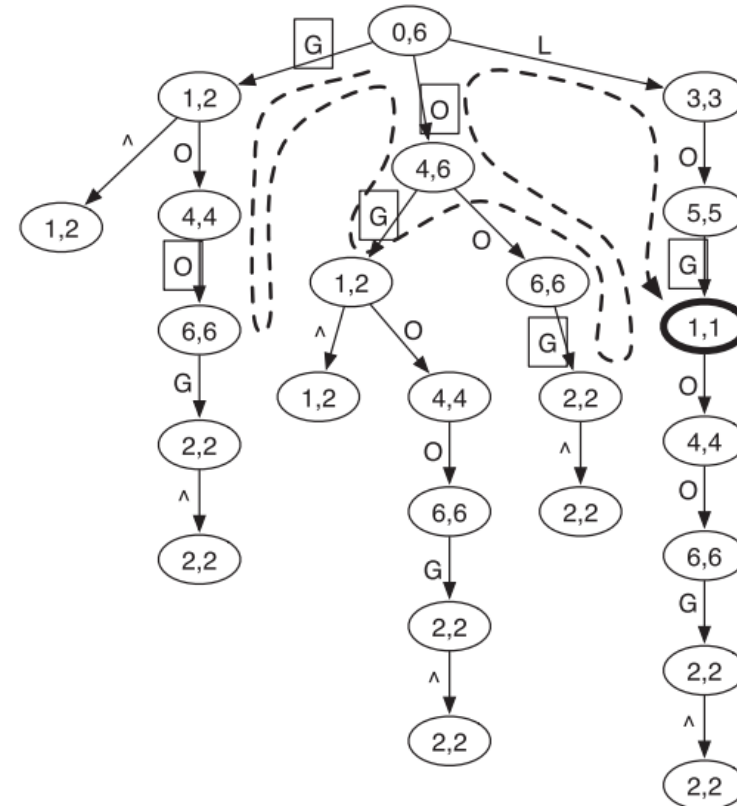
if $k > l$ **then**

$k \leftarrow 1$

$l \leftarrow |X| - 1$

$z \leftarrow z + 1$

$D(i) \leftarrow z$



Searching for $W = lol$ in $X = googol\$$ allowing 1 mismatch

BWA for Inexact matching

CALCULATED(W)

$k \leftarrow 1$

$l \leftarrow |X| - 1$

$z \leftarrow 0$

for $i=0$ **to** $|W|-1$ **do**

$k \leftarrow C(W[i]) + O'(W[i], k-1) + 1$

$l \leftarrow C(W[i]) + O'(W[i], l)$

if $k > l$ **then**

$k \leftarrow 1$

$l \leftarrow |X| - 1$

$z \leftarrow z + 1$

$D(i) \leftarrow z$

CALCULATED ('lol'):

for $i = 0$,

$k = 3 + O'('L', 0) + 1 = 3 + 0 + 1 = 4$

$l = C('L') + O'('L', 6) = 3 + 1 = 4$

So, $D[0] = 0$

Similarly, we can show, $D[1] = D[2] = 1$

Searching for $W = \text{lol}$ in $X = \text{googol\$}$ allowing 1 mismatch

BWA for Inexact matching

INEXRECUR(W, i, z, k, l)

if $z < D(i)$ **then**

return \emptyset

if $i < 0$ **then**

return $\{[k, l]\}$

$I \leftarrow \emptyset$

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

for each $b \in \{A, C, G, T\}$ **do**

$k \leftarrow C(b) + O(b, k-1) + 1$

$l \leftarrow C(b) + O(b, l)$

if $k \leq l$ **then**

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i, z-1, k, l)$

if $b = W[i]$ **then**

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z, k, l)$

else

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

return I

$W = \text{lol}$

0	6	\$googo l
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

mismatch = 0

Searching for $W = \text{lol}$ in $X = \text{googol\$}$ allowing 1 mismatch

BWA for Inexact matching

INEXRECUR(W, i, z, k, l)

if $z < D(i)$ **then**

return \emptyset

if $i < 0$ **then**

return $\{[k, l]\}$

$I \leftarrow \emptyset$

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

for each $b \in \{A, C, G, T\}$ **do**

$k \leftarrow C(b) + O(b, k-1) + 1$

$l \leftarrow C(b) + O(b, l)$

if $k \leq l$ **then**

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i, z-1, k, l)$

if $b = W[i]$ **then**

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z, k, l)$

else

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

return I

$W = \text{LOL}$

0	6	\$googo l
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

mismatch = 0

Searching for $W = \text{lol}$ in $X = \text{googol\$}$ allowing 1 mismatch

BWA for Inexact matching

INEXRECUR(W, i, z, k, l)

if $z < D(i)$ **then**

return \emptyset

if $i < 0$ **then**

return $\{[k, l]\}$

$I \leftarrow \emptyset$

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

for each $b \in \{A, C, G, T\}$ **do**

$k \leftarrow C(b) + O(b, k-1) + 1$

$l \leftarrow C(b) + O(b, l)$

if $k \leq l$ **then**

*

$I \leftarrow I \cup \text{INEXRECUR}(W, i, z-1, k, l)$

if $b = W[i]$ **then**

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z, k, l)$

else

$I \leftarrow I \cup \text{INEXRECUR}(W, i-1, z-1, k, l)$

return I

$W = \text{lol}$

0	6	\$googo l
1	3	gol\$go o
2	0	googol \$
3	5	l\$goog o
4	2	ogol\$g o
5	4	ol\$goo g
6	1	oogol\$ g

mismatch = 1

Output= gol

Searching for $W = \text{lol}$ in $X = \text{googol\$}$ allowing 1 mismatch

Concluding Remarks

Pros:

Memory Efficient:

- Only needs to store BWT array and a small fraction of O and S array, S is calculated using inverse compressed suffix array Ψ^{-1}
- The memory requirement is doubled for calculating D
- In all, the alignment procedure uses $4n + n \lceil \log_2 n \rceil / 8$ bits, $2n$ for B, $n \lceil \log n \rceil / 32$ for O, $n \lceil \log n \rceil / 32$ for S

Time Efficient:

Because exact repeats are collapsed on one path on the prefix trie, no need to align the reads against each copy of the repeat

Gapped alignment for single-end reads

Important when reads get longer and tend to contain indels

Cons:

Performance is degraded on long reads especially when the sequencing error rate is high

BWA always requires the full read to be aligned but longer reads are more likely to be interrupted

Reference

- [1] Ma B, Tromp J, Li M. PatternHunter: faster and more sensitive homology search. Bioinformatics
BWT of the reverse genome to calculate the bound, the memory required for calculating intervals is doubled
- [2] Ferragina,P. and Manzini,G. (2000) Opportunistic data structures with applications. In Proceedings of the
41st Symposium on Foundations of Computer Science (FOCS 2000), IEEE Computer Society, pp. 390–398.

Thank You

