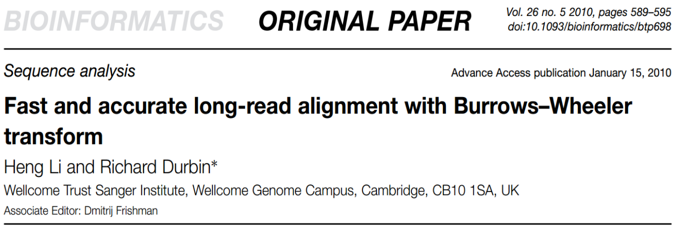

BIOINFORMATICS ORIGINAL PAPER Vol. 26 no. 5 2010, pages 589–595
doi:10.1093/bioinformatics/btp698

Sequence analysis

Advance Access publication January 15, 2010

Fast and accurate long-read alignment with Burrows–Wheeler transform

Heng Li and Richard Durbin*
Wellcome Trust Sanger Institute, Wellcome Genome Campus, Cambridge, CB10 1SA, UK
Associate Editor: Dmitrij Frishman

Fast and accurate long-read alignment with Burrows-Wheeler transform

Dongfang Wang

January 14, 2014

Outline

Differences between short-read and long-read alignment

Short-read alignment

- align full-length read
- efficient for ungapped alignment or limited gaps

Long-read alignment

- Find local matches (more fragile to structural variations and misassemblies)
- Permissive about alignment gaps

Outline

Suffix array and BWT

1. The alphabet of nucleosides $\Sigma = (A, C, D, T)$, and a symbol \$ smaller than all in Σ
2. Nucleoside sequence $X = a_1 \cdots a_{n-1} \quad a_{n-1} = \$$

$$\begin{array}{ll} X[i] & : \text{ the } i\text{–th symbol} \\ X[i,j] := a_i \cdots a_j & \text{subsequence of } X \\ X_i := X[i, n-1] & \text{a suffix of } X \end{array}$$

3. Suffix array S: integers (0..n-1) such that $S(i) = j$ iff X_j is the i–th smallest suffix
4. Burrows-Wheeler transform B: a permutation of X

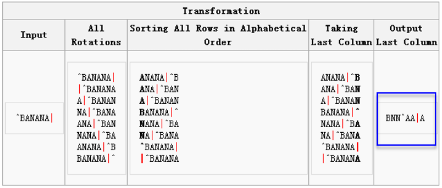
$$B[i] = \$ \text{ if } S(i) = 0$$
$$B[i] = X[S(i) - 1]$$

An example

Sequence:

1	2	3	4	5	6	7	8
^	B	A	N	A	N	A	|

Burrows-Wheeler Transform:



Transformation				
Input	All Rotations	Sorting All Rows in Alphabetical Order	Taking Last Column	Output Last Column
^BANANA|	^BANANA|	ANANA|^B	ANANA|^B	BNN^AA|A
	|^BANANA	ANA|^BAN	ANA|^BAN	
	A|^BANAN	A|^BANAN	A|^BANAN	
	NA|^BANA	BANANA|^	BANANA|^	
	ANA|^BAN	NANA|^BA	NANA|^BA	
	NANA|^BA	NA|^BANA	NA|^BANA	
	ANANA|^B	^BANANA|	^BANANA|	
	BANANA|^	|^BANANA	|^BANANA	

Suffix Array:

^BANANA|	7
BANANA|	4
ANANA|	1
NANA|	5
ANA|	2
NA|	6
A|	3
|	8

1	2	3	4	5	6	7	8
3	5	7	2	4	6	1	8
B	N	N	^	A	A	|	A

Suffix array interval

Given a sequence W:

$$\begin{aligned}\underline{R}(W) &= \min\{k : W \text{ is the prefix of } X_{S(k)}\}\\ \overline{R}(W) &= \max\{k : W \text{ is the prefix of } X_{S(k)}\}\end{aligned}$$

- Suffix array interval $[\underline{R}(W), \overline{R}(W)]$
- The positions of all the occurences of W : $\{S(k) : \underline{R}(W) \leq k \leq \overline{R}(W)\}$

$$\begin{aligned} C(a) &= \#\{0 \leq j \leq n-2 : X[j] < a\} \\ O(a,i) &= \#\{0 \leq j \leq i : B[j] < a\} \end{aligned}$$

Ferragina and Manzini(2000):

$$\begin{aligned} \underline{R}(aW) &= C(a) + O(a, \underline{R}(W) - 1) + 1 \\ \overline{R}(aW) &= C(a) + O(a, \overline{R}(W)) \end{aligned}$$

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

FM-index

Use a simplified FM-index where we don't compress B and store part of the occurrence array O

Outline

- Build FM-indices for reference and query sequences
- query sequence W: prefix DAWG $\mathscr{G}(W)$
- reference sequence X: prefix trie $\mathscr{T}(X)$

Prefix trie of a string X

A tree with each edge labeled with a symbol such that **the concatenation of symbols from a leaf to the root gives a unique prefix of X**

SA interval of a node

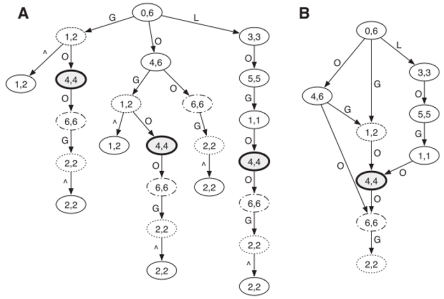
Path from a node to the root represents a substring of X

Prefix DAWG of a string X

Transformed from the prefix trie by **collapsing nodes having an identical suffix array interval**

An example

prefix trie and prefix DAWG of string ‘GOOGOL’



Prefix:

G	GOOGO
GOOG	GOO
GO	GOOGOL

Suffix array:

GOOGOL	2
OOGOL	6
OGOL	4
GOL	1
OL	5
L	3

1	2	3	4	5	6
4	1	6	3	5	2
G O L	G O O G O L	L	O G O L	O L	O O G O L

Dynamic programming:(Similar to Smith-Waterman)

Traverse both $\mathscr{G}(W)$ and $\mathscr{T}(X)$ in a nested way.

- $G_{uv} = I_{uv} = D_{uv} = 0$
  when u is the root of $\mathscr{G}(W)$ and v is the root of $\mathscr{T}(X)$

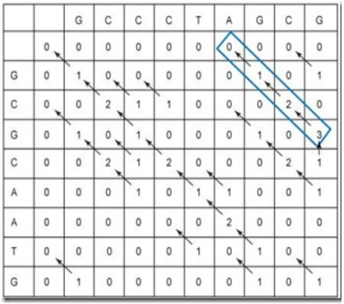
$$\begin{aligned} I_{uv|u'} &= \max\{I_{u'v}, G_{u'v} - q\} - r \\ D_{uv|u'} &= \max\{D_{uv'}, G_{uv'} - q\} - r \\ G_{uv|u'} &= \max\{G_{u'v'} + S(u', u; v', v), I_{u'v'}, D_{u'v'}, 0\} \end{aligned}$$

u': parent of u
v': parent of v
$S(u, u'; v, v')$: score between the symbol (u, u') and (v, v')
q, r: gap open and gap extension penalties

$matches$: if $G_{uv} > 0$

once u doesn't match v, u doesn't match any nodes descending from v

Smith-Waterman



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

$$H_{i,j} = \max\{H_{i-1,j-1} + s(a_i, b_j), \max_{k\geq 1}\{H_{i-k,j} - W_k\}, \max_{l\leq 1}\{H_{i,j-l} - W_l\}, 0\}$$

How to accelerate?

What makes it inefficient?

complex steps related to the travesal of prefix trie and prefix DAWG

A more efficient way

Use BWA-SW to find partial matches and apply the Smith-Waterman to extend

When G_{uv} is good enough and **the SA interval size of** v **is under a certain threshold**, save the (u, v) pair as a *seed interval pair* and don't go deeper from v node

Note:

- multiple seeds(typically 1kb alignments > 10-20 seeds)
- SA interval size (repetitive sequence)

Heuristics rules:

1. Restrict the dynamic programming algorithm around good matches only
2. Report only alignments largely non-overlapping

$\Rightarrow$ savings in computing time

Don't want to visit random matches or matches in low-complexity regions

Z best strategy

- Traverse $\mathscr{G}(W)$ in outer loop and $\mathscr{T}(X)$ in inner loop
- at each node u in $\mathscr{G}(W)$ only keep the top Z best scoring nodes in $\mathscr{T}(W)$ that match u rather than keeping all the matching nodes

miss a seed contained in the true alignment?

If the true alignment is long and contains many seeds, the chance of all seeds being false is very small.

Filtering seeds before Smith-Waterman extension

distinct alignment

if the length of the overlapping region on the query is less than half of the shorter query segment

Aim to find a set of distinct alignments which maximizes the sum of scores of each alignment in the set

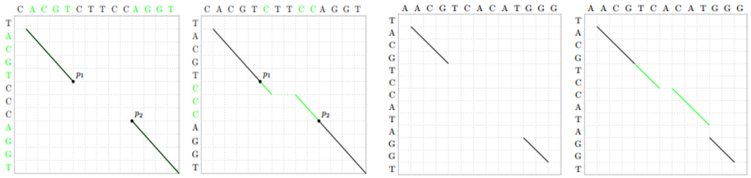
- sort the local alignments based on their alignment scores
- keep an alignment if it is distinct from all the kept alignments with larger scores

But how to discard those seeds that doesn't contribute to the final alignments? Chain seeds that are contained in a band

An example of chaining

Local Chaining

When a seed is found, we want to find a "close" seed that we found previously and combine it to form a longer seed.


C A C G T C T T C C A G G T
T A C G T C C C A G G T
p_1
p_2
C A C G T C T T C C A G G T
T A C G T C C C A G G T
p_1
p_2
A A C G T C A C A T G G G
T A C G T C C A T A G G T
A A C G T C A C A T G G G
T A C G T C C A T A G G T

- several gaps can be included
- a short chain is fully contained in a long chain on the query sequence and the number of seeds in the short chain is below one-tenth of the number of seeds in the long chain, discard all the seeds in the short chain.

Outline

Simulated results

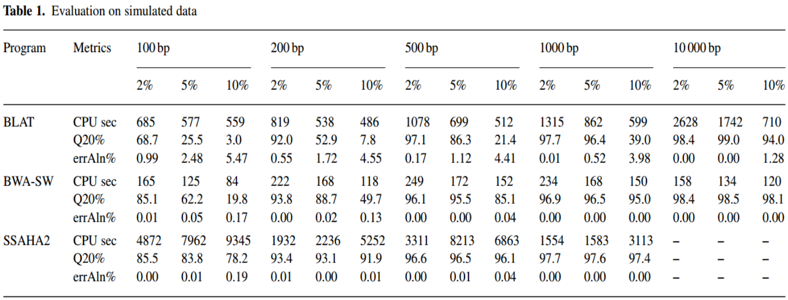


Table 1. Evaluation on simulated data

Program	Metrics	100 bp			200 bp			500 bp			1000 bp			10 000 bp		
		2%	5%	10%	2%	5%	10%	2%	5%	10%	2%	5%	10%	2%	5%	10%
BLAT	CPU sec	685	577	559	819	538	486	1078	699	512	1315	862	599	2628	1742	710
	Q20%	68.7	25.5	3.0	92.0	52.9	7.8	97.1	86.3	21.4	97.7	96.4	39.0	98.4	99.0	94.0
	errAln%	0.99	2.48	5.47	0.55	1.72	4.55	0.17	1.12	4.41	0.01	0.52	3.98	0.00	0.00	1.28
BWA-SW	CPU sec	165	125	84	222	168	118	249	172	152	234	168	150	158	134	120
	Q20%	85.1	62.2	19.8	93.8	88.7	49.7	96.1	95.5	85.1	96.9	96.5	95.0	98.4	98.5	98.1
	errAln%	0.01	0.05	0.17	0.00	0.02	0.13	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00
SSAHA2	CPU sec	4872	7962	9345	1932	2236	5252	3311	8213	6863	1554	1583	3113	–	–	–
	Q20%	85.5	83.8	78.2	93.4	93.1	91.9	96.6	96.5	96.1	97.7	97.6	97.4	–	–	–
	errAln%	0.00	0.01	0.19	0.01	0.00	0.01	0.00	0.01	0.04	0.00	0.00	0.00	–	–	–

Summary

BWA-SW is an efficient algorithm for aligning a query sequence of a few hundred base pairs or more against a long reference genome.

seed-and-extend paradigm, but:

- seeding strategy: long gapped seeds in unique regions; maintain a small fraction

high speed:

- FM-indices
- suppression of short repetitive matches contained in a better match

Discussion

1. Short-read alignment cannot be used for long-read alignment due to:
   - Full-length read vs local matches.
   - Ungapped or limited gap vs larger number of gaps.
2. BWA-short is more accurate, use less memory and competitively fast.
3. BWA-long is the best in market in speed, accuracy and memory.